Funded by the European Union

BEE WITH APEX

E-book on Low Code Programming with APEX

A Practical Guide for Better Employ-ability for Everyone

BeeAPEX Team



Copyright © 2023 BeeAPEX Team

PUBLISHED BY SOFTWARE QUALITY AND TESTING LAB, UM FOV

BEEAPEX.EU

Licensed under the Creative Commons Attribution-NonCommercial 4.0 License (the "License"). You may not use this file except in compliance with the License. You may obtain a copy of the License at https://creativecommons.org/licenses/by-nc-sa/4.0. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

1st edition, December 7, 2023

Contents

Acknowledgement 27

Preface 29

I How to in APEX

1 How to start Oracle APEX? 35

Vjeran Strahonja, Dijana Oreški . . . . .

1.1 What is Oracle APEX and what is it for? 35
1.1.1 What is Oracle APEX? 35
1.1.2 And what is the application? 35
1.1.3 How can Oracle APEX help in application development? 35
1.1.4 Application development cycle 36
1.2 How to start Oracle APEX? 42
1.2.1 What is your skill level? 42
1.2.2 Use on-premise APEX instance 43
1.2.3 https://apex.oracle.com 43
1.2.4 Virtual Box Appliance / Virtual Machine 44
1.2.5 APEX docker 44
1.2.6 APEX instance in Oracle Cloud Infrastructure 46
1.2.7 APEX instance in Oracle Academy 47
1.3 Questions 47
1.4 Answers 47

2 How to prepare a database? 49

E. Kapsammer, W. Retschitzegger, W. Schwinger . . . . .

2.1 How to Model the Data 50
2.1.1 Logical Model 50
2.1.2 Relational Model 54
2.1.3 Normalization of an RDB-Schema 59

2.2 How to Manage the RDBS-Schema – SQL-DDL 60
2.2.1 Automatic Table Generation using ODM 61
2.2.2 Automatic Table Generation using Quick SQL 63
2.2.3 Manual Table Creation using SQL-DDL 65
2.2.4 Manual Table Management using Object Browser 67
2.3 How to Manipulate Data – SQL-DML 68
2.3.1 Data Manipulation using SQL-DML 68
2.3.2 Data Manipulation using ORACLE's Object Browser (OB) 69
2.3.3 Data Manipulation using Quick SQL 69
2.4 How to Query Data – SQL-DQL 70
2.4.1 Data Querying using SQL-DQL 70
2.4.2 Data Querying using ORACLE's Query Builder ("QB") 75
2.5 Building up the DB-Layer – The Big Picture 78
2.6 Questions 78
2.7 Answers 79

3 How to Navigate in APEX? 81
E. Kapsammer, W. Retschitzegger, W. Schwinger . . . . . .
3.1 Web Application Development Process followed by APEX 81
3.2 The Start Menu of APEX 82
3.3 Overview of the App Builder – Create and Manage your Apps 83
3.4 Create Application – Three Use Cases 84
3.5 Create Application – Properties, Pages, Features and Settings 85
3.6 Specify Pages 87
3.7 Maintain and Modify a Page – Page Designer 89
3.8 Run Application or Page 89
3.9 Questions 91
3.10 Answers 91

4 How to exchange data in APEX? 93
E. Kapsammer, W. Retschitzegger, W. Schwinger . . . . . .
4.1 Importing and Exporting Data Using "Data Workshop" 94
4.1.1 Importing Data 94
4.1.2 Importing Steps 96
4.1.3 Exporting Data 100
4.2 Importing and Exporting Data Using "Object Browser" 101
4.3 Exporting the Result of a SQL-Command 102
4.4 Exporting Data from an Application Report 103
4.5 Enabling Data Exchange with RESTful Services 104
4.5.1 REST Architectural Pattern 105
4.5.2 Enabling a DB Schema for RESTful Access 106
4.5.3 Resource Modules | Templates | Handlers 107
4.5.4 Using "AutoREST" instead of Manually Defining Resources 109

4.6 Questions . . . 111
4.7 Answers . . . 111

5 How to generate a first draft of the application? . . . 113
A. Angeioplastis, G. Myllis, A. Tsimpiris, D. Varsamis

5.1 Why business need applications? . . . 113
5.2 Setting up the ORACLE APEX Environment . . . 113
5.3 SQL workshop example development . . . 117
5.3.1 Quick SQL . . . 120
5.3.2 SQL Script . . . 121
5.4 Data driven part of application . . . 124
5.5 Administration of application . . . 127
5.6 Access control . . . 127
5.7 Supplementary learning material . . . 133
5.7.1 Exported applications . . . 133
5.7.2 Video guides . . . 133
5.8 Questions . . . 133
5.9 Answers . . . 134

6 How to manage reports? . . . 135
Zuzana Žillová, Erik Malina, Matej Grochal, Andrej Staník, Andrea Meleková, Michal Kvet, Miroslav Potočár

6.1 Report . . . 135
6.2 Classic Report . . . 144
6.3 Column Toggle Report . . . 146
6.4 Interactive Report . . . 146
6.5 Questions . . . 155
6.6 Answers . . . 155

7 How to manage forms? . . . 157
Veronika Šalgová, Michal Kvet, Miroslav Potočár

7.1 Types of forms . . . 157
7.2 Editable Interactive Grid . . . 157
7.3 Form on a Table . . . 160
7.4 Master Detail Form . . . 160
7.5 Questions . . . 163
7.6 Answers . . . 163

8 How to transform text reports into charts? . . . 165
Ivan Pastierik, Michal Kvet, Miroslav Potočár

8.1 Chart . . . 165
8.2 Creating Bar Chart . . . 166

8.3 Adding Filtering to Bar Chart . . . 169
8.4 Adding Sorting to Bar Chart . . . 174
8.5 Creating Different Types of Charts . . . 178
8.6 Questions . . . 179
8.7 Answers . . . 180

9 How to manage menus? . . . 181
Veronika Šalgová, Michal Kvet, Miroslav Potočár

9.1 How to manage menus . . . 181
9.2 Side Menu . . . 181
9.3 Top Menu . . . 182
9.4 Mega Menu . . . 182
9.5 Editing Menu Lists . . . 182
9.6 Questions . . . 183
9.7 Answers . . . 183

10 How to collaborate in a team? . . . 185
P. Staniszewski, M. Sońta, A. Kierzkowski

10.1 Collaborative knowledge production is the essence of low-code development . . . 185
10.2 Being online together . . . 185
10.3 Tech savviness through collaboration . . . 186
10.4 Features description . . . 187
10.4.1 Page locking . . . 187
10.4.2 Comments . . . 188
10.4.3 Build options . . . 188
10.4.4 Team development . . . 189
10.4.5 Feedback . . . 192
10.4.6 Standardization . . . 193
10.5 Conclusions . . . 194
10.6 Questions . . . 195
10.7 Answers . . . 195

11 How to benefit from a gallery of applications and plug-ins? . . . 197
Vjeran Strahonja, Dijana Oreški

11.1 How to install sample and starter apps? . . . 197
11.2 Starter Apps . . . 198
11.3 Sample Apps . . . 199
11.4 Plug-ins . . . 201
11.5 Questions . . . 204
11.6 Answers . . . 204

12 How to manage packaged and multilingual applications? 205
R. Leskovar, U. Rajkovič, A. Baggia
12.1 Application and packaged application . . . 205
12.2 Application . . . 207
12.2.1 Scope of application . . . 207
12.2.2 Create tables . . . 207
12.2.3 Insert data . . . 208
12.2.4 Generate application . . . 208
12.3 Packaged application . . . 213
12.4 Multilingual application . . . 217
12.5 Supplementary learning material . . . 221
12.5.1 Exported applications . . . 221
12.5.2 Video guides . . . 221
12.6 Questions . . . 221
12.7 Answers . . . 222

II Constructing application in APEX

13 Intranet news for employees . . . 225
R. Leskovar, U. Rajkovič, A.Baggia
13.1 Business view of the case . . . 225
13.2 Problem definition . . . 225
13.3 Use cases . . . 226
13.3.1 Narrative description of use case . . . 226
13.3.2 Semi-structured description . . . 226
13.3.3 Use case diagram . . . 226
13.4 Data model . . . 228
13.4.1 Narrative description of data model . . . 228
13.4.2 Logical data model . . . 228
13.4.3 Relational data model . . . 230
13.5 Application interfaces . . . 231
13.6 Supplementary learning material . . . 233
13.6.1 Exported application . . . 233
13.6.2 Video guides . . . 233
13.7 Questions . . . 233
13.8 Answers . . . 233

14 GreenDi - Catalog of plants . . . 235
Vjeran Strahonja, Dijana Oreški, Darko Andročec, Ana Kutnjak
14.1 Business view of the case . . . 235
14.2 Problem definition . . . 235
14.3 Use cases . . . 236
14.3.1 Narrative description of use case . . . 236

14.3.2 Semi-structured description . . . 236
14.3.3 Use case diagram . . . 236
14.4 Data model . . . 238
14.4.1 Narrative description of data model . . . 238
14.4.2 Logical data model . . . 238
14.4.3 Relational data model . . . 239
14.5 Application interfaces . . . 240
14.6 Supplementary learning material . . . 241
14.6.1 Exported application . . . 241
14.6.2 Video guides . . . 242
14.7 Questions . . . 242
14.8 Answers . . . 242

15 GreenDi - User Authorisation and Management . . . 243

VJERAN STRAHONJA, DARKO ANDROČEC, ANA KUTNJAK, LARISA HRUSTEK . . .

15.1 Business view of the case . . . 243
15.2 Problem definition . . . 243
15.3 Use cases . . . 244
15.3.1 Narrative description of the use case . . . 244
15.3.2 Semi-structured description . . . 244
15.3.3 Use case diagram . . . 244
15.4 Data model . . . 246
15.4.1 Narrative description of a data model . . . 246
15.4.2 Logical data model . . . 246
15.4.3 Relational data model . . . 246
15.5 Application interfaces . . . 247
15.6 Supplementary learning material . . . 248
15.6.1 Exported application . . . 248
15.6.2 Video guides . . . 248
15.7 Questions . . . 248
15.8 Answers . . . 248

16 Small Innovation System . . . 251

R. LESKOVAR, U. RAJKOVIČ, A.BAGGIA . . .

16.1 Business view of the case . . . 251
16.2 Problem definition . . . 251
16.3 Use cases . . . 251
16.3.1 Narrative description of use case . . . 251
16.3.2 Semi-structured description . . . 252
16.3.3 Use case diagram . . . 252
16.4 Data model . . . 252
16.4.1 Narrative description of data model . . . 252
16.4.2 Logical data model . . . 252
16.4.3 Relational data model . . . 256

16.5 Application interfaces . . . 257
16.6 Supplementary learning material . . . 259
16.6.1 Exported application . . . 260
16.6.2 Video guides . . . 260
16.7 Questions . . . 260
16.8 Answers . . . 260

17 Business process management . . . 261
R. Leskovar, U. Rajkovič, A.Baggia

17.1 Business view of the case . . . 261
17.2 Getting started with Flows for APEX . . . 264
17.2.1 Install Flows for APEX . . . 264
17.2.2 Install Sample Process Flow Application . . . 264
17.2.3 Read and practice exercises . . . 264
17.3 Problem definition . . . 264
17.4 Use cases . . . 265
17.4.1 Narrative description . . . 265
17.4.2 Semi-structured description . . . 265
17.4.3 Use case diagram . . . 265
17.5 Workflow model . . . 268
17.6 Data model . . . 271
17.6.1 Narrative description of data model . . . 271
17.6.2 Logical data model . . . 271
17.6.3 Relational data model . . . 271
17.6.4 Quick SQL for generating SQL script . . . 272
17.6.5 Sequence and two stored functions . . . 273
17.7 Application interfaces . . . 274
17.7.1 List of values in Shared Components . . . 275
17.7.2 Plug-ins in Shared Components . . . 276
17.7.3 Sales manager . . . 276
17.7.4 Production manager . . . 280
17.7.5 Financial manager . . . 282
17.7.6 Chief executive officer - business manager . . . 283
17.8 Linking application with Flows for APEX . . . 285
17.9 Define user roles . . . 287
17.10 Testing and correcting errors . . . 287
17.11 Supplementary learning material . . . 288
17.11.1 Exported application . . . 288
17.11.2 Video guides . . . 290
17.12 Questions . . . 290
17.13 Answers . . . 290

18 GreenDi – Exchange of Plants and Seeds . . . 291
Vjeran Strahonja, Dijana Oreški, Darko Andročec, Ana Kutnjak

18.1 Business view of the case . . . 291

18.2 Problem definition . . . 291
18.3 Use cases . . . 292
18.3.1 Narrative description of use case . . . 292
18.3.2 Semi-structured description . . . 292
18.3.3 Use case diagram . . . 293
18.4 Data model . . . 293
18.4.1 Narrative description of data model . . . 293
18.4.2 Logical data model . . . 293
18.4.3 Relational data model . . . 294
18.5 Application interfaces . . . 294
18.6 Supplementary learning material . . . 295
18.6.1 Exported application . . . 295
18.6.2 Video guides . . . 295
18.7 Questions . . . 296
18.8 Answers . . . 296

19 Book review management system . . . 297
A. Kutnjak, L. Hrustek, A. Baggia and R.Leskovar . . .

19.1 Business view of the case . . . 297
19.2 Problem definition . . . 297
19.3 Use cases . . . 298
19.3.1 Narrative description of the use case . . . 298
19.3.2 Semi-structured description . . . 298
19.3.3 Use case diagram . . . 298
19.4 Data model . . . 300
19.4.1 Narrative description of data model . . . 300
19.4.2 Logical data model . . . 300
19.4.3 Relational data model . . . 300
19.4.4 SQL script . . . 301
19.4.5 Quick SQL . . . 302
19.5 Application interfaces . . . 303
19.5.1 Administrator . . . 303
19.5.2 User . . . 305
19.6 Define user roles . . . 307
19.7 Supplementary learning material . . . 307
19.7.1 Exported application . . . 307
19.7.2 Video guides . . . 308
19.8 Questions . . . 308
19.9 Answers . . . 308

20 Bill-of-material and cost calculation . . . 309
R. Leskovar, U. Rajković, A.Baggia . . .

20.1 Business view of the case . . . 309
20.2 Problem definition . . . 309

20.3 Use cases . . . 310
20.3.1 Narrative description . . . 310
20.3.2 Semi-structured description . . . 310
20.3.3 Use case diagram . . . 310
20.4 Data model . . . 313
20.4.1 Narrative description of data model . . . 313
20.4.2 Implementation of business rules in data base . . . 313
20.4.3 Logical data model . . . 314
20.4.4 Relational data model . . . 314
20.4.5 Objects in APEX . . . 316
20.5 Application interfaces . . . 319
20.6 Supplementary learning material . . . 322
20.6.1 Exported applications . . . 322
20.6.2 Video guides . . . 322
20.7 Questions . . . 322
20.8 Answers . . . 322

21 Nutrition and diet management . . . 325
A. Angeioplastis, G. Myllis, A. Tsimpiris, D. Varsamis . . .
21.1 Business view of the case . . . 325
21.2 Problem definition . . . 325
21.3 Use cases . . . 325
21.3.1 Narrative description . . . 325
21.3.2 Semi-structured description . . . 325
21.3.3 Use case diagram . . . 327
21.4 Data model . . . 327
21.4.1 Logical data model . . . 327
21.4.2 Relational data model . . . 328
21.4.3 QuickSQL and Script SQL Script . . . 329
21.5 Application interfaces . . . 332
21.5.1 How to build a dashboard . . . 336
21.5.2 Developers and application users . . . 340
21.6 Supplementary learning material . . . 343
21.6.1 Exported application . . . 343
21.6.2 Video guides . . . 343
21.7 Questions . . . 343
21.8 Answers . . . 344

22 Office Hours Scheduling . . . 345
J. Mańko, M. Sońta, R. Leskovar . . .
22.1 Business view of the case . . . 345
22.2 Problem definition . . . 346
22.3 Use cases . . . 346
22.3.1 Narrative description . . . 346
22.3.2 Semi-structured description . . . 347
22.3.3 Use case diagram . . . 347

22.4 Data model **349**
22.4.1 Narrative description of data model 349
22.4.2 Logical data model 350
22.4.3 Relational data model 350
22.4.4 Quick SQL 351
22.4.5 SQL script for creating tables 352
22.4.6 Query builder in APEX 352
22.5 Application interfaces **353**
22.5.1 Management application interfaces 353
22.5.2 Student office application interfaces 354
22.5.3 Student application interfaces 355
22.5.4 Teacher application interfaces 358
22.6 Supplementary learning material **360**
22.6.1 Exported application 360
22.6.2 Video guides 362
22.7 Questions **362**
22.8 Answers **363**

23 Telco case **365**

Veronika Šalgová, Jozef Kostolný, Michal Mrena, Michal Kvet, Miroslav Potočár . . . . . .

23.1 Business view of the case **365**
23.2 Problem definition **365**
23.3 Use cases **365**
23.3.1 Narrative description 366
23.3.2 Semi-structured description 366
23.3.3 Use case diagram 366
23.4 Data model **367**
23.4.1 Narrative description of data model 367
23.4.2 Logical data model 368
23.4.3 Relational data model 368
23.5 User authentication and user roles **369**
23.6 Application interfaces **371**
23.6.1 Application design 372
23.7 Scripts **380**
23.8 Creating a home page **382**
23.9 Creating a customer page **383**
23.10 Creating a manager page **385**
23.11 Supplementary learning material **386**
23.11.1 Exported application 386
23.11.2 Video guides 386
23.12 Questions **386**
23.13 Answers **386**

24 Car rental case . . . 389
A. Angeioplastis, G. Myllis, A. Tsimpiris, D. Varsamis . . .
24.1 Business view of the case . . . 389
24.2 Problem definition . . . 389
24.3 Use cases . . . 390
24.3.1 Narrative description of use case . . . 390
24.3.2 Semi-structured description . . . 390
24.3.3 Use case diagram . . . 390
24.4 Data model . . . 392
24.4.1 Narrative description of data model . . . 392
24.4.2 Logical data model . . . 393
24.4.3 Relational data model . . . 393
24.4.4 SQL Script . . . 394
24.5 Application interfaces . . . 395
24.6 Supplementary learning material . . . 400
24.6.1 Exported applications . . . 400
24.6.2 Video guides . . . 400
24.7 Questions . . . 400
24.8 Answers . . . 400
Bibliography . . . 401
Articles . . . 401
Books . . . 401
Index . . . 403

List of Figures

1.1 Application development cycle. 36
1.2 Business Model Canvas (6) 38
1.3 The basic concepts of the Use Case diagram. 39
1.4 Generalization/specialization of Actors. 40
1.5 Boundaries of the system and associations "include" and "extend". 40
1.6 Use Cases diagram of the Hotel Reservation system. 41
1.7 Searching for the APEX docker. 45

2.1 DBS as the Backbone of Web Applications. 49
2.2 Logical Model of our Running HR Example using an ER-Diagram. 50
2.3 Managing the Logical Model with ODM. 52
2.4 Managing Attributes with ODM. 53
2.5 Managing Relationships with ODM. 54
2.6 RDB-Schemata and Table Schemata. 55
2.7 Process for RDB-Schema Generation out of the Logical Model. 56
2.8 Automatically Generated RDB-Schema. 56
2.9 GUI-Elements for Managing Columns of the RM. 58
2.10 GUI-Elements for Managing the RM. 58
2.11 Normalization Steps – Overview. 59
2.12 Categories of SQL Statements. 61
2.13 Export of the RDB-Schema into a DDL-script. 62
2.14 Upload and Execution of a SQL-Script. 63
2.15 Accessing Quick SQL via SQL Workshop 64
2.16 Generated SQL-Script based on Quick SQL. 65
2.17 SQL Command Editor. 66
2.18 SQL Statement for Creating Table `Departments`. 66
2.19 SQL Statement for Dropping Table `Departments`. 67
2.20 SQL Statement for Altering Table `Departments` 67
2.21 Table Management with ORACLE's Object Browser. 67
2.22 SQL Statement for Inserting new Data into Table `Departments`. 68
2.23 SQL Statement for Updating the Salary of all Employees by 10% in Table `Employees`. 68
2.24 SQL Statement for Updating the Salary of Employee "Miller", only. 68
2.25 SQL Statement for Updating more than one attribute in Table `Employees`. 69

2.26 SQL Statement for Deleting all Data from Table Employees. 69
2.27 SQL Statement for Deleting some Data from Table Employees. 69
2.28 Random Data Insertion with Quick SQL. 70
2.29 SQL Statement for Retrieving all Data from Table Employees. 70
2.30 Excerpt from Retrieving all Data from Table EMPLOYEES. 71
2.31 SQL Statement for Retrieving all Data from Table Employees ordered by their Salary. 71
2.32 Excerpt from Retrieving all Data from Table Employees ordered by their Salary. . 71
2.33 SQL Statement for Retrieving certain Data from Table Employees only. 72
2.34 Excerpt from Retrieving certain Data from Table Employees only. 72
2.35 SQL Statement for Retrieving Data from Table Employees using TO_CHAR()-Function. 72
2.36 Excerpt from Retrieving Data from Table Employees using TO_CHAR()-Function. . 73
2.37 SQL Statement for Joining Data from Table Employees and Table Departments. . 73
2.38 Excerpt from Retrieving certain Data from Table Employees and Table Departments. 74
2.39 SQL Statement for Counting the number of Employees and Building the Sum of their Salaries from Table Employees. 74
2.40 Excerpt from Retrieving the Number of Employees and Building the Sum of their Salaries from Table Employees. 74
2.41 SQL Statement for Grouping the Data by Their Salary and Ordering the Result by the Number of Employees with that Salary from Table Employees. 75
2.42 Excerpt from Grouping the Data by Their Salary and Ordering the Result by the Number of Employees with that Salary from Table Employees 75
2.43 Overview of the ORACLE Query Builder (“QB”). 76
2.44 Using the Query Builder to build a Join. 77
2.45 Overview on the Options to Manage thee DB-Layer. 78

3.1 Overall Development Process in APEX. 82
3.2 Oracle APEX Workspace Homepage. 83
3.3 Overview of the App Builder. 84
3.4 Create Application Wizard: Three Use Cases when Creating an Application. . . . 85
3.5 Properties and Pages. 86
3.6 Features and Settings. 86
3.7 Application Homepage - Developer’s view. 87
3.8 Creating a Page. 88
3.9 Page Designer. 89
3.10 Running Entire Application. 90
3.11 Running Individual Pages. 91

4.1 Data Exchange Options in APEX. 94
4.2 Using Data Workshop to Load Data - Access through SQL Workshop. 95
4.3 Using Data Workshop to Load Data - Access through App Builder. 96
4.4 Provision of Data Source. 97
4.5 Loading Data into New Table. 97
4.6 Loading Data into Existing Table. 99
4.7 Exporting ("Unloading") Data. 100
4.8 Unload Data Wizard. 101
4.9 Import/Export of Table Data Using the Object Browser. 102
4.10 Data Export of the Result of a SQL-Query. 103
4.11 Export of Data From an Application Report. 104
4.12 RESTful Services for Data Exchange – Basic Architecture. 105
4.13 Enabling a DB Schema for RESTful Access. 107

4.14 Relationships Between the Different Components of ORDS RESTful Services. . . 108
4.15 Example RESTful Services for Table Employees. 109
4.16 Enabling REST Service in the Object Browser. 110

5.1 Type in APEX link and sign in. 114
5.2 Request an Oracle APEX Workspace initial steps. 114
5.3 Request an Oracle APEX Workspace completion. 115
5.4 Oracle APEX Workspace approved. 115
5.5 Activation email from APEX. 116
5.6 Set up a new password for Workspace APEX. 116
5.7 Workspace APEX environment. 117
5.8 Logical Model of our Running HR Example. 118
5.9 Relational Model of our Running HR Example. 119
5.10 Insert Quick SQL code to APEX workspace. 121
5.11 Run SQL script. 123
5.12 Create App starting process. 123
5.13 Create App from Script. 124
5.14 Web application created. 125
5.15 App builder environment. 125
5.16 App Login Page. 126
5.17 Your new App environment. 126
5.18 Manage users and groups environment. 127
5.19 Access control example. 128
5.20 Add new users. 129
5.21 Add roles to users. 130
5.22 Create button configuration. 131
5.23 Contributor access and rights. 132
5.24 Reader access and rights. 133

6.1 Selection of page type. 136
6.2 Creation of page with report. 137
6.3 Rendering. 137
6.4 Properties of region. 138
6.5 Various data sources. 138
6.6 Layout. 139
6.7 Appearance. 139
6.8 Templates. 140
6.9 Advanced. 140
6.10 Header and Footer. 140
6.11 Server-side Condition. 141
6.12 Read Only. 141
6.13 Security, Server Cache and Customization. 142
6.14 Pagination. 142
6.15 Number of rows to load. 142
6.16 Download and Printing. 143
6.17 Center of page designer. 143
6.18 Type of column. 144
6.19 Column settings. 144
6.20 Format Mask. 144

6.21 Customization. . . . 145
6.22 Option for sorting by column. . . . 145
6.23 Location of green arrow for page showcase. . . . 145
6.24 Classic report example. . . . 146
6.25 Possibility to change visibility of column in Column Toggle Report. . . . 146
6.26 Interactive Report. . . . 147
6.27 Search panel. . . . 147
6.28 Searching. . . . 148
6.29 Actions Button. . . . 148
6.30 Selection of columns to display. . . . 149
6.31 Column Filter. . . . 149
6.32 Row Filter. . . . 150
6.33 After clicking on Data button. . . . 150
6.34 Sort. . . . 151
6.35 Aggregate. . . . 151
6.36 Flashback. . . . 151
6.37 Control Break. . . . 152
6.38 Control Break result. . . . 152
6.39 Highlight. . . . 152
6.40 Highlight result. . . . 153
6.41 Row Per Page. . . . 153
6.42 Chart in the interactive report. . . . 153
6.43 Group By. . . . 154
6.44 Save Report. . . . 154
6.45 Download. . . . 155

7.1 Interactive Grid with Customers . . . 157
7.2 Actions of Interactive Grid . . . 158
7.3 Hide icon of the Phone column . . . 158
7.4 Displayed columns . . . 158
7.5 Freeze icon of the Address column . . . 159
7.6 Sorting the Address column . . . 159
7.7 Aggregate icon of the Address column . . . 159
7.8 Types of charts . . . 160
7.9 Form for inserting customers . . . 160
7.10 Stacked master detail form of Customer Flat Rates . . . 161
7.11 Side by Side master detail form of Customer Flat Rates . . . 161
7.12 Drill Down master detail form – first page . . . 162
7.13 Drill Down master detail form – second page . . . 162

8.1 Data model used in example application. . . . 165
8.2 Selecting and placing Chart region. . . . 166
8.3 Layout body section after placing Chart region. . . . 166
8.4 Tree overview showing error in chart series. . . . 167
8.5 Final configuration of sales series . . . 167
8.6 Data returned by executing SQL query. . . . 168
8.7 Chart shown when launching the application. . . . 168
8.8 Changing title attribute for x axis. . . . 169
8.9 Sales Chart after labelling axes. . . . 169

8.10 Selecting and placing Checkbox Group item. 170
8.11 Setting basic attributes of Checkbox Group item. 170
8.12 Setting list of values for Checkbox Group item. 171
8.13 Setting default values for Checkbox Group item. 171
8.14 Linking Sales Chart region with Checkbox Group item. 172
8.15 Creating Dynamic Action for P1_PRODUCTS_CHOICE item. 172
8.16 Setting basic attributes of on change Dynamic Action. 173
8.17 Setting attributes of Refresh action. 173
8.18 Dynamic Action to refresh Sales Chart in tree overview. 174
8.19 Application after implementing product filtering. 174
8.20 Selecting and placing Select List item. 175
8.21 Sales Chart region after placing and renaming Select List item. 175
8.22 Setting list of values for Select List item. 175
8.23 Setting static values for Select List item. 176
8.24 Adding refresh Dynamic Action for Select List item. 176
8.25 Setting Order By Item for Sales Chart region. 176
8.26 Setting Order By clauses for Select List item used for sorting. 177
8.27 Application after implementing chart sorting. 177
8.28 Changing the type of graph to Line with Area. 178
8.29 List of Values for Select List item used for filtering product sales by year. 178
8.30 Setting source SQL Query and column mapping for Monthly Sales Chart region. 179
8.31 Monthly Sales Chart. 179

9.1 Expanded Side Navigation Menu 181
9.2 Top Navigation Tabs 182
9.3 Top Navigation Menu 182
9.4 Top Navigation Mega Menu 182
9.5 Navigation menu attributes 183

10.1 Page Blocking feature in APEX. 187
10.2 Locking a specific page. 187
10.3 Adding comments. 188
10.4 Build options. 189
10.5 Team Development feature. 190
10.6 Activation of Team Development feature. 190
10.7 Customization of Team Development feature. 191
10.8 Issue-related communication inside Team Development feature. 191
10.9 Allowing feedback in Application definition. 192
10.10 Submitting feedback in application. 193
10.11 Utilities menu. 194

11.1 Oracle APEX workspaces come with Sample Apps and Starter Apps. 197
11.2 Starter Apps in Gallery. 198
11.3 Installation of starter app Team Calendar. 199
11.4 Sample Apps. 200

12.1 Application import and export wizard. 206
12.2 Transforming Quick SQL to SQL commands. 208
12.3 Script CH12INSERT insert data in three tables and commit transactions. 208
12.4 Create application from script. 209

12.5 Selecting application name and all features. 209
12.6 Selecting Shared components. 210
12.7 Create list of values from scratch. 210
12.8 Defining name and type: CH12_LOV_COMPETENCE_DESCRIPTION is dynamic. 211
12.9 Entering SQL SELECT command. 211
12.10 Define name and type: CH12_LOV_COMPETENCE_LEVEL is static. 212
12.11 Define display and return values: CH12_LOV_COMPETENCE_LEVEL is static. . 212
12.12 Select page 7 (Ch12 Job Competences). 213
12.13 Page 7 report (Ch12 Job Competences). 213
12.14 Select Supporting Objects. 214
12.15 Setting prerequisites. 214
12.16 Set check on existence of three tables. 214
12.17 Set prompt to rename application. 215
12.18 Set installation scripts. 215
12.19 Set deinstallation scripts. 215
12.20 Import the application into another workspace. 216
12.21 Licence agreement. 216
12.22 Rename imported application. 217
12.23 Adding a role to user. 217
12.24 Defined languages for translation. 218
12.25 Seed translatable text. 218
12.26 Export strings for particular language and page. 219
12.27 Translation of "target" tagged strings in lines 48, 52, 56, 60, 72 and 76. 219
12.28 Uploading XLDIFF translation files. 219
12.29 Applying changes and publishing. 220
12.30 Final publishing of the application translation. 220
12.31 Setting application primary language. 220
12.32 Translated page. 220

13.1 Use case diagram. 226
13.2 Logical data model. 230
13.3 Relational data model. 230
13.4 The Contributor (Publisher) dashboard. 231
13.5 Publishing news with attachments. 231
13.6 Viewer (Reader) dashboard. 232
13.7 Viewer (Reader) access to news. 232
13.8 Administrator dashboard. 232
13.9 Editing employee role by Administrator. 233

14.1 Use case diagram - Catalog of plants. 238
14.2 Logical data model. 239
14.3 Relational data model. 240
14.4 Interactive grid for Plant Form. 241
14.5 Public part – open browsing of plant data. 241

15.1 Use case diagram - GreenDi User Authorization and Management 244
15.2 Logical data model. 246
15.3 Relational data model. 247
15.4 User Form. 247
15.5 User History Form. 248

16.1 Use case diagram. . . . 252
16.2 Logical data model. . . . 256
16.3 Relational data model. . . . 256
16.4 The Employee dashboard. . . . 257
16.5 Submitting idea with attachments. . . . 257
16.6 Organizational Structure in the company. . . . 258
16.7 Small Innovation Idea overview. . . . 258
16.8 Reviewer dashboard. . . . 259
16.9 Page for reviewing the ideas. . . . 259

17.1 Integrating Flows for APEX with APEX application. . . . 263
17.2 Use case diagram. . . . 268
17.3 Workflow - processing inquiry - BPMN diagram. . . . 268
17.4 Creating instances. . . . 269
17.5 Creating instance of the flow CH17. . . . 269
17.6 Start demo instance. . . . 269
17.7 Complete first task in demo instance. . . . 270
17.8 Executing tasks in parallel gate. . . . 270
17.9 Completed demo instance. . . . 270
17.10 Logical data model. . . . 271
17.11 Relational data model. . . . 272
17.12 Home page of "CH17 Business Process Management" application. . . . 275
17.13 Sales - initiated process. . . . 276
17.14 Sales - flow report. . . . 276
17.15 Sales - flow diagram for selected instance. . . . 277
17.16 Sales - flow report for selected instance. . . . 277
17.17 Sales - inquiry report. . . . 278
17.18 Sales - list of documents. . . . 278
17.19 Sales - uploading document for inquiry. . . . 279
17.20 Sales - list of documents after uploading and refreshing. . . . 279
17.21 Sales - report for selected instance after uploading three documents. . . . 280
17.22 Sales - flow report for selected instance after uploading three documents. . . . 280
17.23 Manufacturing - flow report. . . . 281
17.24 Manufacturing - BPMN diagram - state of instance. . . . 281
17.25 Manufacturing evaluation. . . . 281
17.26 Manufacturing evaluation finished, instance waiting to financial evaluation. 282
17.27 Finance - flow report. . . . 282
17.28 Finance - BPMN diagram - state of instance. . . . 282
17.29 Financial evaluation. . . . 283
17.30 Financial evaluation finished, instance waiting to business evaluation. . . . 283
17.31 Finance - BPMN diagram - state of instance. . . . 284
17.32 Business - flow report. . . . 284
17.33 Business evaluation. . . . 284
17.34 Business evaluation finished, instance terminated. . . . 285
17.35 Showing the state of the instance on BPMN diagram. . . . 286
17.36 Setting page items. . . . 286
17.37 Completing step in Flows for Apex. . . . 287
17.38 Define roles and user roles in Application Access Control menu. . . . 287
17.39 Granting "create job privilege" to workspace. . . . 288

17.40 Import workflow called CH17 into Flows for APEX - step 1. 289
17.41 Import workflow called CH17 into Flows for APEX - step 2. 289

18.1 Use case diagram - Exchange of Plants and Seeds 293
18.2 Logical data model. 294
18.3 Relational data model. 294
18.4 Offers - interactive report. 295
18.5 Message form. 295

19.1 Use case diagrams. 298
19.2 Logical data model. 300
19.3 Relational data model. 301
19.4 Generating application out of script CH19CREATEINSERT - part 1. 303
19.5 Generating application out of script CH19CREATEINSERT - part 2. 304
19.6 Adding book by administrator. 305
19.7 Adding category by administrator. 305
19.8 User registration. 306
19.9 Browsing and adding reviews. 306
19.10 Form to comment a review 307

20.1 Use case diagram. 313
20.2 Logical data model. 314
20.3 Definition of unique index in Oracle SQL Data Modeler. 315
20.4 Relational data model. 315
20.5 Generating SQL script by using Quick SQL tool. 317
20.6 Application home page. 320
20.7 Managing basic data - report. 320
20.8 Managing basic data - form. 320
20.9 Managing structure data - report. 321
20.10 Managing structure data - form. 321
20.11 Page for calculation of bill of material. 322

21.1 Use case diagram. 327
21.2 Logical data model. 328
21.3 Relational data model. 329
21.4 Quick SQL tool to create SQL script. 330
21.5 SQL script procedure to build SEQUENCE. 332
21.6 Create new application. 333
21.7 Nutrition application. 334
21.8 Nutrition application- recipe page. 335
21.9 Nutrition application- ingredients page. 335
21.10 Nutrition application- recipe ingredients page. 336
21.11 Nutrition application- nutrition facts page. 336
21.12 Create a dashboard page for your application. 337
21.13 Configure Title and chat type on Dashboard reports. 338
21.14 Configure SOURCE with SQL QUERY. 340
21.15 Creation of new user account. 341
21.16 Create User account within application. 342
21.17 Configure pages with user rights. 343

22.1 Use case diagram. 349
22.2 Logical data model. 350
22.3 Relational data model. 351
22.4 Data model described with Quick SQL. 351
22.5 An example Query Builder usage. 352
22.6 Application home page. 353
22.7 Management dashboard. 354
22.8 Office hours interactive report for student office. 354
22.9 Rescheduling form for student office. 355
22.10 Office hours interactive report after rescheduling. 355
22.11 Calendar of student appointments. 356
22.12 Calendar of all office hours offered by teachers in student's study program. . 356
22.13 Enrollment to office hours through calendar - picking calendar slot. 356
22.14 Enrollment to office hours through calendar - selecting the purpose. 357
22.15 Enrollment to office hours through calendar - successful enrollment. 357
22.16 Checking the enrollment through the "View my appointments" menu item. . 357
22.17 Detailed interactive report of all student appointments. 358
22.18 Detailed report on teacher appointments. 358
22.19 Teacher calendar with visible student names, their languages and purposes. 359
22.20 Rescheduling teacher office hours with NONE enrolled - the calendar view. . 359
22.21 Rescheduling teacher office hours with NONE enrolled - new date entered. 359
22.22 The results of rescheduling teacher office hours with NONE enrolled. 360

23.1 Use Case Diagram. 367
23.2 Logical data model. 368
23.3 Relational data model. 369
23.4 Customer dashboard. 373
23.5 Customer dashboard – Invoice modal window. 374
23.6 Customer dashboard – Invoice in PDF. 374
23.7 Customer dashboard – Stats of minutes. 374
23.8 Customer dashboard – Stats of SMS. 375
23.9 Customer dashboard – Stats of data. 375
23.10 Manager dashboard – Add customer. 376
23.11 Manager dashboard – Manage customer. 377
23.12 Manager dashboard – View customers. 377
23.13 Manager dashboard – Customer export. 378
23.14 Manager dashboard – Customer export in XLS file. 378
23.15 Administrator dashboard. 379
23.16 A landing page with login. 379
23.17 Login page. 379
23.18 Regions of the body. 382
23.19 List of content. 382
23.20 HTML code of a static region. 382

24.1 Use case diagram. 392
24.2 Logical model of the Car Rental Project 393
24.3 Relational Model of the Car Rental Project 394
24.4 Log in to the app. 396
24.5 Home page of the app. 396

24.6 Cars template 397
24.7 How to make a column with photos. 398
24.8 Customers data. 399
24.9 Rent car template. 399



List of Tables

13.1 Use case description: publishing internet news. 227
13.2 Use case description: reading intranet news 228
13.3 Use case description: managing the intranet portal. 229

14.1 Use case description: browsing catalog of plants 237

15.1 Use case description: user Authorization and Management. 245

16.1 Use case description: idea submission. 253
16.2 Use case description: overview of ideas. 254
16.3 Use case description: idea evaluation. 255

17.1 Use case description: prepare inquiry documentation. 266
17.2 Use case description: evaluate manufacturing aspects of inquiry. 266
17.3 Use case description: evaluate financial aspects of inquiry. 267
17.4 Use case description: evaluate business aspects of inquiry. 267

18.1 Use case description: exchange of Plants and Seeds. 292

19.1 Use case description: book reviews management system 299

20.1 Use case description: report and maintain basic data. 310
20.2 Use case description: report and maintain structure data. 311
20.3 Use case description: calculate BOM. 312

21.1 Use case description: access to recipes, ingredients and nutrition facts calculation 326

22.1 Use case description: rescheduling of office hours by teacher. 347
22.2 Use case description: student enrollment for office hours. 348

23.1 Use case description: add service. 366
23.2 Use case description: show service status. 367
23.3 Sample data stored in the CH23_Person table. 370

24.1 Use case description: accessing cars, customers and car rent reservation . . . 391

Acknowledgement

Project partners and the BEE with APEX team:

- University of Maribor Faculty of Organizational Sciences,
- University of Zagreb Faculty of Organization and Informatics,
- University of Žilina,
- Kozminski University,
- International Hellenic University and
- Johannes Kepler University

would like to acknowledge the financial support given by the European Commission through the Action Erasmus + Better Employability for Everyone with APEX (project ID 2021-1-SI01-KA220-HED-000032218), co-funded by the Erasmus+ programme of the European Union.

The European Commission support for this publication's production does not constitute endorsement of the contents which reflects only the views of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Big thanks to:

- Oracle Academy *Program Manager Mr. Darko Jureković* for his continuous support in project results dissemination and
- THE RIGHT THING SOLUTIONS *CEO Mr. Aljaž Mali* for his valuable advices on APEX before and during the project.

BeeAPEX project team



Preface

Welcome to the exploring of Oracle Application Express (APEX) – an intuitive and powerful low-code development platform for creating data-driven web applications. This textbook is designed to equip you with the competences needed to harness the full potential of Oracle APEX and build cutting-edge applications to address real-world business challenges.

Part I of this textbook is “How to”, dedicated to the fundamental aspects of Oracle APEX. In these twelve chapters, you will embark on a journey to understand the core concepts, gain access to APEX, and explore various functionalities to build robust applications. Each chapter delves into a specific topic, providing clear instructions, and facilitating hands-on learning experience.

Chapter 1: “How to start Oracle APEX?” explains the fundamentals of what APEX is, along with what it can be used for, before describing various ways to prepare the APEX environment for a hands-on learning experience and application development.

Chapter 2: “How to prepare a database?” provides introduction to data modeling, managing data base, manipulating data and querying data. For beginners a big picture understanding of data base concepts is a must.

Chapter 3: “How to navigate in APEX?” gives a tour of APEX functionalities which enable the development, generation and customization of different web pages and it’s components. Running and testing the APEX application is just one tab away from development environment.

Chapter 4: “How to exchange data in APEX?” gives insight into importing and exporting data within APEX. The chapter covers data exchange with files such as spreadsheets and also through RESTful services.

Chapter 5: “How to generate a first draft of the application?” invites you to try the development power of APEX. You will find that once you decide what your data is, you can immediately generate appealing and functional application with no programming. It also how basic access control for end users with different roles can be instantly generated.

Chapter 6: “How to manage reports?” guides you through delivering views of the data through classic and interactive reporting wizards. APEX reports already include functionalities for end-user customization of reports with no programming at all.

Chapter 7: “How to manage forms?” introduce you to three common types of web form including master-detail. You will customize and generate form pages with no programming skills.

Chapter 8: “How to transform text reports into charts?” pave a path to utilize APEX capabilities to present data as charts directly beside text reports.

Chapter 9: “How to manage menus?” presents types of various navigation elements for your APEX application.

Chapter 10: “How to collaborate in a team?” offers insight into APEX functionalities which serve teams since it is a rare situation that application development will involve only one developer.

Chapter 11: “How to benefit from a gallery of sample applications and plug-ins?” invites you to apply powerful APEX capabilities through re-use of good patterns.

Chapter 12: “How to manage packaged and multilingual applications?” sets a path to distribute your application to other APEX environments for users that speak different languages.

Part I also covers topics which are vital for application security, deployment strategies, and readiness for the real-world.

Part II of this textbook takes you beyond the fundamentals, presenting twelve engaging business cases which require you to solve a problem. Each case is carefully documented to provide a holistic understanding of application development from a business, data and user interface perspective. This part includes applications:
for businesses:

- intranet news for employees,
- small innovation system,
- business process management with workflows,
- bill of material calculation,
- book review system,
- nutrition and diet management,
- office hours scheduling,
- telecommunication company billing,
- car rental business

for communities:

- catalog of plants,
- exchange of plants.

and general applicable user authorization and management. In each business case, you will explore:

- Business view of the case: an overview of the business situation.
- Problem definition: A search to answer who and why someone has a headache.
- Use cases: Three types of description are presented: narrative, semi-structured and graphical to prepare UML use case documentation.
- Logical and relational data model: APEX is fully featured to start fresh new data structures, to use and modify existing ones, to combine with other data modeling tools and to support forward or reverse engineering. Developers’ efforts to ensure appropriate chunks of data and the relationships between and considering business needs are a foundation to proceed with user interfaces.
- Application interfaces: textbook provides HTML pages, reports, forms, fields, menus, buttons, hyperlinks which materializes the business situation, the solution to the business problem, and the use cases and data with end-user in mind.
- Supplementary learning material: To enhance, accelerate and help you on the development path you will find links to exported applications, scripts, data and video tutorials for each chapter. These resources will provide you with practical insights, allowing you to reinforce your knowledge and apply it directly to real-world projects.

Whether you are an experienced developer seeking to expand your skill-set or a beginner eager to explore the world of APEX, this textbook is your definitive guide. Our hope is that also non-IT learners will also find it an invaluable companion on the journey to mastering Oracle APEX and building innovative applications that make a positive impacts.

The textbook and supplementary material are designed for approximately 75 hours of student effort (3 ECTS). We hope that different modes of study can be applied through:

- teacher lead class lectures and lab exercises,
- blended learning and also
- self-paced study.

Depending on the learner's background knowledge and available time to run the course, teachers can easily assemble a set of chapters that suits the learning situations like: extracurricular courses, summer schools, time limited events for low-code introductions for all students (not only in IT or CS), and practitioners in various branches of industry.

APEX versions 22 and 23 were used for developing this textbook and supplementary learning material. We believe that concepts and core technologies explained and applied in the contents will be beneficial also to the learners of the future APEX versions.

Get ready to embark on an exciting learning adventure with Oracle APEX! Enjoy using wizards and low-coding!

Professor Robert Leskovar
BeeAPEX project leader, Chair of IT Department, UM, Faculty of Organizational Sciences

How to in APEX

1 How to start Oracle APEX? 35
Vjeran Strahonja, Dijana Oreški . . . . . .

2 How to prepare a database? 49
E. Kapsammer, W. Retschitzegger, W. Schwinger . . . . .

3 How to Navigate in APEX? 81
E. Kapsammer, W. Retschitzegger, W. Schwinger . . . . .

4 How to exchange data in APEX? 93
E. Kapsammer, W. Retschitzegger, W. Schwinger . . . . .

5 How to generate a first draft of the application? 113
A. Angeioplastis, G. Myllis, A. Tsimpiris, D. Varsamis . .

6 How to manage reports? 135
Zuzana Žillová, Erik Malina, Matej Grochal, Andrej Staník, Andrea Meleková, Michal Kvet, Miroslav Potočár

7 How to manage forms? 157
Veronika Šalgová, Michal Kvet, Miroslav Potočár . .

8 How to transform text reports into charts? 165
Ivan Pastierik, Michal Kvet, Miroslav Potočár . . . . . .

9 How to manage menus? 181
Veronika Šalgová, Michal Kvet, Miroslav Potočár . .

10 How to collaborate in a team? 185
P. Staniszewski, M. Sońta, A. Kierzkowski . . . . . .

11 How to benefit from a gallery of applications and plug-ins? 197
Vjeran Strahonja, Dijana Oreški . . . . . .

12 How to manage packaged and multilingual applications? 205
R. Leskovar, U. Rajković, A. Baggia . . . . . .



1. How to start Oracle APEX?

VJERAN STRAHONJA, DIJANA OREŠKI

1.1 What is Oracle APEX and what is it for?

1.1.1 What is Oracle APEX?

Oracle APEX (also known as APEX or Oracle Application Express) is an enterprise low-code development platform, developed by Oracle Corporation, which enables users to create and deploy applications using a web-based interface. It is a set of tools for creating different types of applications that consist of programs for working with data in the database: including entering, storing, displaying, changing, calculating and logical operations with data, etc. Using this platform enables the development of applications for various purposes that include generating a database based on a set of structured data or data models, working directly with data in the database using the SQL language, creating user interfaces, generating various tabular and graphical reports and deploying web applications on Oracle databases.

1.1.2 And what is the application?

An application is generally a set of computer programs that has a specific purpose and works on a set of data in a database. We are particularly interested in business applications: those whose programs are used to process data on business processes and transactions. The development of an application, even a simple one, is a complex thought and creative process, which requires certain knowledge and skills. Firstly, an algorithmic way of thinking (algorithmic literacy) is required. It is the ability to conceptualize, and the awareness that an algorithm exists in a situation, along with the ability to apply it; the knowledge of how to design and use an algorithm, and the ability to critically evaluate algorithms. Knowledge and skills of creating high-quality records in a programming language (coding), testing and debugging, and other things we usually call programming are also required.

1.1.3 How can Oracle APEX help in application development?

When developing computer programs, we cannot avoid the algorithmic way of thinking. Using platforms and tools for low-code / no-code programming, such as Oracle APEX, allows non-programmers to easily program without the need to know the details of a programming language and advanced programming skills. So, they can focus on the problem and the algorithm rather than the programming language and technology. We are particularly interested in fast, user-friendly

application development for accessing, unifying, analysing and displaying data from open data sources and from corporate databases. Open data are freely available data for anyone to access, use and share. They can be used without restrictions (or with some), shared with others or used to create new works. They are made available by governments, companies and other organizations to promote transparency and cooperation. Today there are many easily accessible sources of meteorological, traffic, geological health and similar data, as well as statistics on education, health, economy, etc. Corporate databases contain data on business transactions and other data that are created as a result of business processes. Visualization, analysis, integration and different views of this data can be used to make business decisions or achieve some other benefits. Oracle APEX is used for prototype development, which is used to discuss what the program should do and show the IT specialist what the user wants. The prototype reflects the essential features of the system. Later, it is added to the system that will be used or serves as a basis for creating a system in another programming language, tool, or platform.

1.1.4 Application development cycle

Regardless of what kind of application we develop and what kind of methodological framework, development tools or platform we use, the application development cycle goes through several stages. These stages are different in different methodological approaches. Figure 1.1 shows one possible pattern of the application development cycle (see Figure 1.1).

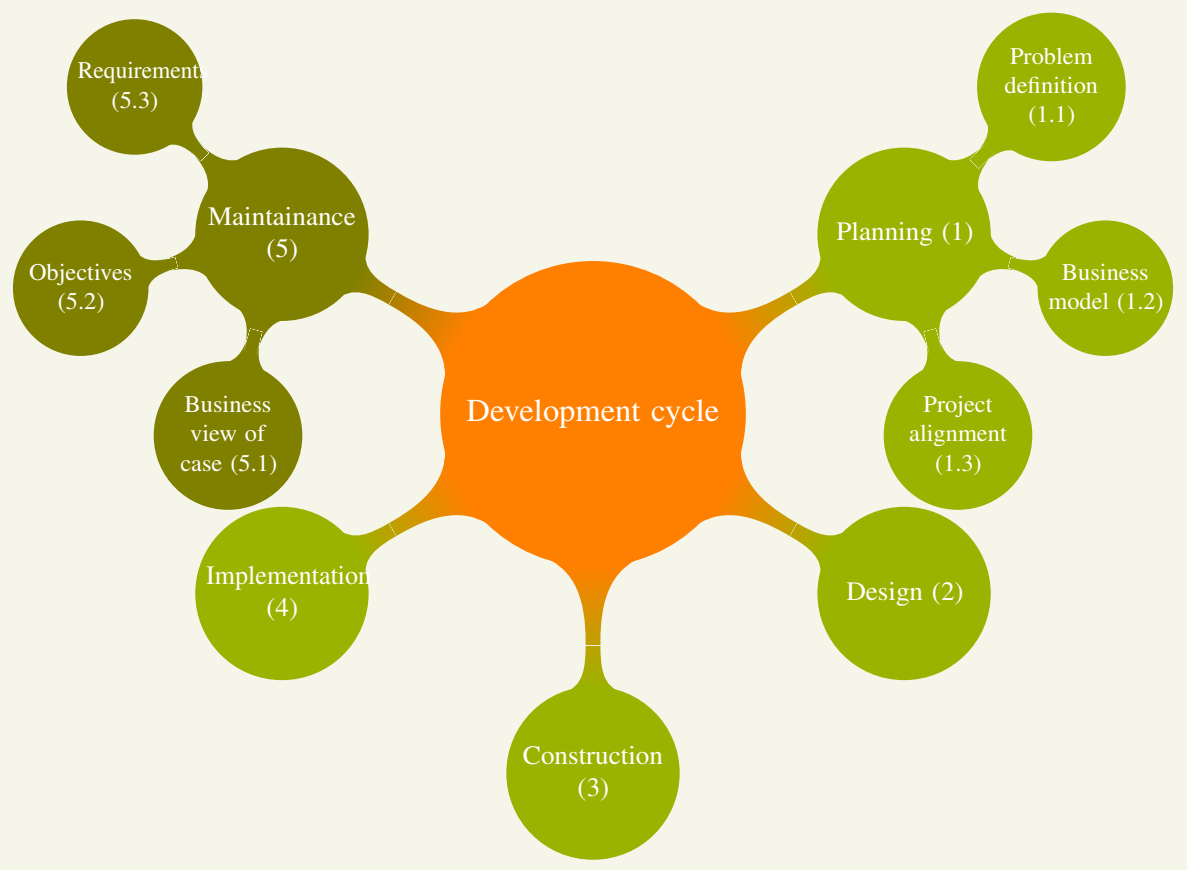


Figure 1.1: Application development cycle.

As shown in the figure, the application development cycle contains the following phases or processes:

1. Planning (defining the problem and project)

- definition and analysis of the business model
- requirement specification
- determination of basic processes and data
- technology and architecture decisions
- development planning

2. Application design
   - analysis and specification of requirements
   - logical application design (use cases and logical data model)
3. Construction of the application
   - database design
   - development (prototype) of the application
   - testing
4. Implementation of the application

The Planning is followed by Design, Construction and Implementation. These phases or processes can be performed sequentially or iteratively (evolutionary, incremental), so that we go through the entire development cycle several times and add a part of the functionality in each iteration.

Application Planning The application development cycle begins with the Planning. With simple applications, the initial information for the Planning of the application can be Problem definition, document which clearly states the problem. Who has a headache? Why is this a real headache? What is the manifestation of the problem? It can be a few sentences or pages of text. In the case of complex business applications, detailed specifications should consider:

- Business view of the case describes how people inside organization perceive situations or what they expect from the application.
- Objectives of the application development describe what the application wants to achieve terms of business, technological and/or other factors, present as the indicators of success.
- Business model describes values, customer segments, customer relationships and channels, key partners, activities and resources, structure and sources of revenues and costs, etc.
- Software requirements specification defines requirements regarding function, behavior, performance, technology etc.
- The project plan describes the scope and phases of the future project, the activities (tasks) that are performed in individual phases, the inputs and outputs of individual activities, the resources that are needed, the interconnection of activities and their duration, costs, risks, etc.

There are methods for creating each of the listed specifications, which are described in the literature. For example, Alexander Osterwalder [6] developed the Business Model Canvas method, which is used to develop, improve and document new and existing business models As shown on Figure 1.2, it is actually a visual chart of 9 building blocks that describe the value proposition provided by the organization, its customers, relationships and channels with them, key partners, activities and resources, as well as the cost structure and revenue streams.



Figure 1.2: Business Model Canvas [6]

.

While Business Model Canvas (BMC) serves to foster understanding, analysis and design of the business model, and especially what needs to be supported by the application we develop, and Software Requirements Specification (SRS) is a document that describes what the software will do and how it will be expected to perform. SRS describes the functionality, performance and other characteristics that the application must have in order to support the business model and needs of all users.

Application Design From the first computers and computer programs to today, hundreds of different methodological approaches, methods and techniques of application design have been developed. The methodology of program design has changed with technological development. Low-code / no-code program development is an approach that prefers simple methods, understandable to users who are not professional software developers. However, application development requires some design processes before program generation.

As we mentioned before, a computer application is a set of programs that work with data in a database. To develop an application, we need to design both the programs and the database. If the database already exists, we must know how to use it, and above all, understand the data model. For the needs of low-code / no-code program development, we will introduce the most necessary design methods. At the same time, the basis of the design is the creation of models that also have a graphic representation. We will use the following methods:

- Use Case (UC) model that defines the relationship between the application and the environment.
- An entity-relationship (ER) model that describes the structure of data at a logical level.

Use case will be explained in the continuation of this chapter, while entity-relationship (ER) will be explained in Chapter 2.

Use Case (UC) Use cases are one of the software development methods. They describe the relationship between the system (application) and the environment (users and other systems). The method is simple, so even non-professional users understand it. Use cases are an integral part of the Unified Modeling Language (UML), the most used modeling language today in the field of software engineering, which provides a standard way to visualize the design of a system. There is a rich literature UML and the Object Management Group (OMG) takes care of its development and standardization. UML contains 13 methods and diagrammatic techniques for modeling the structure, behavior and interaction of software, and we will only utilize Use cases.

The Use case model consists of two parts:

- Specifications (narrative description) of use cases
- Use case diagram

First, we will study the Use Case diagram. This diagram describes what the system does, from the point of view of an observer outside the system. It doesn't matter how the system works internally.

The basic concepts of the Use Case diagram are shown in Figure 1.3.

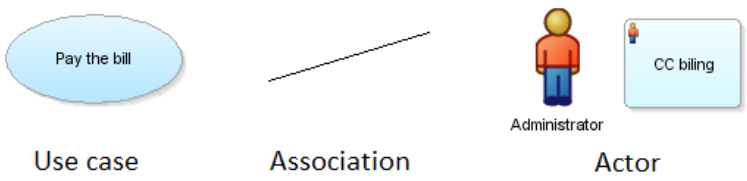


Figure 1.3: The basic concepts of the Use Case diagram.

A use case represents a set of actions that can be performed by a system (e.g., a software component) in interaction with external actors. It is a story that describes how actors use the system to achieve certain goals or perform tasks. It represents an abstract task that has a purpose. The link (Association) connects the participants in the communication, for example the actor and the use case, and represents their interaction and the relationship between the system and the environment (behaviour). An Actor represents a set of roles that interact with the system in the same way, for example a user class, some kind of external system or similar. An actor is someone outside the System under discussion who interacts with it but is not part of it. It can be a living being (User, Patient, Pilot), or another system (Billing System, Bank, Carrier).

The diagram can also show the classification structure (see Figure 1.4), which explains the generalization/specialization of the concepts, for example Actors. All subclasses (children) inherit properties and behaviour from the superclass (parent), which means that they also inherit links to use cases from the parent. At the same time, an individual subclass can enter a communication that neither the superclass nor other subclasses enter into.

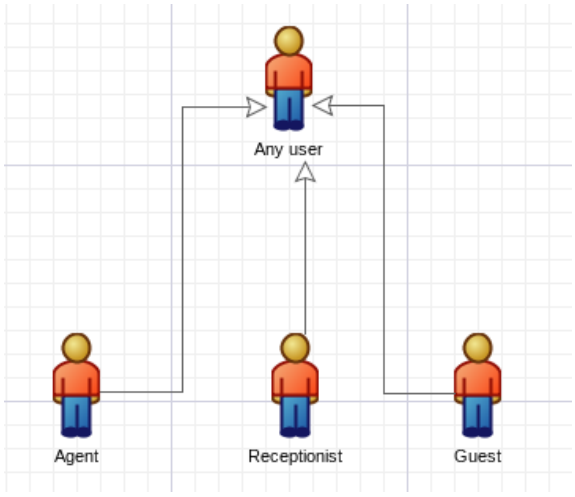


Figure 1.4: Generalization/specialization of Actors.

Figure 1.5 shows the boundaries of the system as a rectangle with the name of the system, inside which the use cases are located.

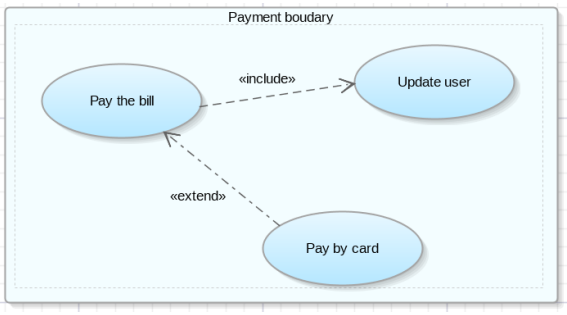


Figure 1.5: Boundaries of the system and associations "include" and "extend".

UC diagrams do not show workflow or sequence of use cases. However, there are two allowed types of associations between use cases which are also shown in Figure 1.5:

- "include" is an association from the base use case to the included use case, indicating that the base use case contains the behavior of the included use case. In this way, the functionality and behavior that is often used is separated into a use case that we will include as necessary in other use cases. Note that the basic use case is not complete without inclusion.
- "extend" is an association from a use case that is an extension to a base use case, indicating that the behavior of the base use case is extended by the behavior of the extension. At the same time, the extension is not part of the basic use case, nor should this communication always exist and function flawlessly.

The use of the aforementioned concepts of the Use cases diagram is best explained with example, such as the Hotel Reservation system, shown in Figure 1.6.

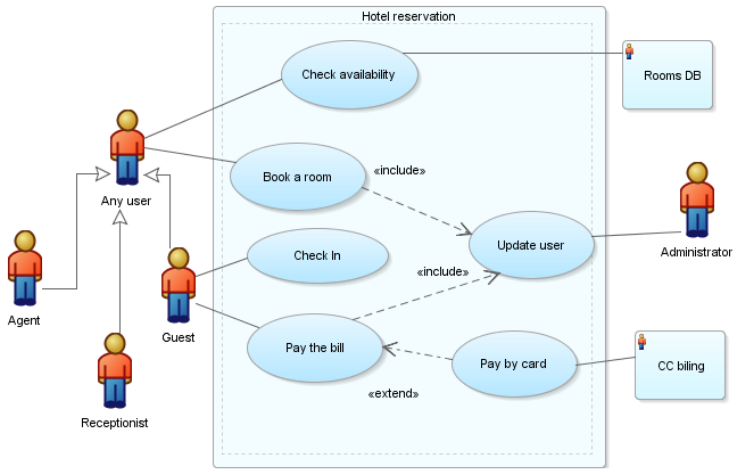


Figure 1.6: Use Cases diagram of the Hotel Reservation system.

The system whose behaviour we are observing is a Hotel Reservation. The system is all within the boundaries shown by the rectangle. The surroundings of the system are outside the rectangle. Within the system boundaries are six Use cases, shown by ellipses with names. Outside the system are Actors, who communicate with the system, that is, its Use cases - this communication is shown by connecting lines, and associations. Any User is an actor who communicates with Check Availability. This Use case communicates with an external actor, Rooms DB. It is some kind of external reservation system that has information about rooms and reservations in its database. Any User can Book the room. This UC includes another UC, Update User. The classification structure of the User actor is shown on the left. Agent, Receptionist and Guest are "a kind of actor" Any user. In the case shown, only Guest can communicate with UCs Check In and Pay the bill. UC Update User is included in Pay the bill, which means that user data can also be updated within the bill payment functionality. UC Pay by card extends the Pay the bill functionality, this means that card payment can be called up during bill payment. Pay by card communicates with an external credit card payment service CC billing. Administrator can communicate with UC Update user and update user information. As we already mentioned, the use case model consists of two parts:

- Specifications (narrative description) of use cases
- Use case diagram.

The Use Case specification describes scenarios and internal logic of Use Cases, initial states (preconditions) and final states, interfaces, system messages, specification of error and exception processing and similar. Usually, some kind of template is used for the Use Case specification, as we will use in this book.

Construction, implementation and maintenance of the application The construction of the application implies the creation of a program and the realization of a physical database. We will use the Oracle APEX platform for this, as will be explained later in the book. The first version of the application that works and can be tested is a prototype of the application. This prototype can be upgraded and expanded. Considering what will happen with the prototype later, we distinguish between two types of prototypes: throw away, and upgradeable. An upgradeable prototype is improved and upgraded to a state that is suitable for use. Whilst the "throw away" prototype serves as an illustration of the functionality and a model on which the application will be built, usually on another platform. If we intend to use the application prototype, we must implement it - that means install it in the production environment, educate users, prepare the database and similar.

The implementation phase is followed by the use of the and maintenance. These include fixing and removing errors, improving performance and minor extensions of functionality. If major

changes are needed, a new development cycle of the application is started.

1.2 How to start Oracle APEX?

Software development is considered to be complex due to the amount of programming required. Consequently, low code solutions are proposed in order to overcome complexity. Implementing a low code approach allows the development of sophisticated apps that are both functional and complex - without writing a line of code. Oracle Application Express is one of the best low-code options for this. There are various ways to start Oracle APEX. Oracle APEX can be used anywhere a database is running, both on-premises and in a private cloud. It can be a physical, dedicated server, a virtualized machine, or a docker image (which is particularly well-liked by Oracle Application Express developers and can be launched on a laptop, while traveling by bus, or plane). Additionally, Exadata can be utilized, a super-capable physical APEX server in the cloud. This chapter will cover all of the shortly describe all options how to access APEX development environment.

1.2.1 What is your skill level?

There are several possibilities to start with Oracle Application Express - APEX. Your current skill level determines the recommended way to start. Let's assume that you have specific domain competences in any organizational process (e.g., selling, hiring, manufacturing, lending, marketing, legal matters, health-care, constructing, education) and the following digital skill levels:

- Absolute beginner: comfortable with web browser, no programming experience at all; recommendation: use on-premise instance or apex.oracle.com or Oracle Academy
- Begginer: comfortable with web browser, little programming experience e.g. using spreadsheets and setting formulas; any other other programming language; recommendation: use on-premise instance or apex.oracle.com or Oracle Academy
- Fresh citizen developer: comfortable with web browser; little programming experience; understanding what is database, table, table column, primary key, foreign key; basic querying, inserting, updating and deleting; recommendation: use on-premise instance or apex.oracle.com or Oracle Academy or VBox appliance
- Skilled citizen developer: comfortable with web browser; moderate programming experience with procedures and functions in any database management system; intermediate querying, inserting, updating and deleting; recommendation: use on-premise instance or apex.oracle.com or Oracle Academy or VBox appliance or Apex docker
- On the road to professional developer: developing skills in PL/SQL and JavaScript; recommendation: use on-premise instance or apex.oracle.com or APEX docker or OCI APEX instance

Your user role and the location of Oracle APEX will determine how you log in and use the application. Oracle APEX can be installed locally on your computer or in a hosted environment like an Oracle Cloud service. Depending on the installation type, the sign-in procedure varies. Users need to create a workspace, add Oracle APEX users, then sign in to the workspace before developing or installing apps. Multiple users can collaborate on the same Oracle APEX installation using a workspace while maintaining the privacy of their objects, data, and applications. If Web browser supports JavaScript, users can log into a workspace to access the Oracle APEX home page. Each workspace has a distinct name and ID. Within Oracle APEX Administration Services, a workspace can be manually created by an instance administrator, or users can make requests. An independent program called Oracle APEX Administration Services is used to oversee a complete Oracle APEX instance.

1.2.2 Use on-premise APEX instance

This section describes how to install Oracle APEX in a on-premises (or local) installation. Oracle APEX installation involves several steps. Those steps are:

- **Choose between Full or Runtime Environment**. Full Environment gives access to the App Builder development environment completely. For production implementations where you wish to execute unchangeable apps, a Runtime Environment is good option. Oracle APEX enables the ability to install only a runtime version of Oracle APEX for testing and production instances. Since developers cannot accidentally update a production program in a runtime instance, this runtime environment reduces installed footprint and rights and enhances application security. Users can run production applications in an Oracle APEX runtime environment, but it does not have a Web interface for administration. A runtime environment is more secure because it just contains the components required to run the program.
- **Verify installation requirements**. Verify whether your system satisfy the minimum requirements for installation. There are five groups of requirements: a) Oracle Database Requirements (Oracle APEX release 22 requires an Oracle Database release 12.1.0.2 or later. Oracle APEX runs on all database editions, including Enterprise Edition (EE), Standard Edition (SE) and Express Edition (XE). Oracle APEX can be installed in single-instance database and in Oracle Real Application Clusters (Oracle RAC) database.), b) Browser Requirements (Oracle APEX requires a JavaScript-enabled browser and supports the current and prior major release of Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft Edge.), c) Web Listener Requirements (Oracle APEX requires Oracle REST Data Services (ORDS) 19.x or later), d) Disk Space Requirements (Oracle APEX disk space requirements 310 MB free space for APEX software files on the file system if using English only download and 705 MB if using full download, 220 MB free space in APEX tablespace, 100 MB free space in SYSTEM tablespace, 60 MB free space in APEX tablespace for each additional language (other than English) installed, e) Oracle XML DB Requirements (Oracle XML DB must be installed in the Oracle database that you want to use if you are installing a full development environment. If you are using a preconfigured database created either during an installation or by Database Configuration Assistant or DBCA, Oracle XML DB is already installed and configured.)
- **Install** the software. Install Oracle APEX by downloading a ZIP file from the Oracle APEX page and then download and install Oracle REST Data Services (ORDS).

1.2.3 https://apex.oracle.com

Navigate your browser to this URL. Requesting a free workspace is the fastest way to get started with Oracle APEX. It only takes a few seconds to join up, and workspace is prepared for users to begin developing apps. This approach doesn't require a cloud account and it is free. To start with Oracle APEX:

- Type in your web browser: apex.oracle.com
- Click on the Start for Free Today button
- On the web page that appears, select Request a free Workspace
- Put in your name, the e-mail address, give your workspace a name and click Next
- Complete the survey and put in why you are requesting workspace. Read through the terms and conditions and accept the terms, click Next
- At this point, you will get an email from the Oracle APEX, click on the link to register and set up a password.

1.2.4 Virtual Box Appliance / Virtual Machine

Navigate your browser to this URL. The main purpose of appliance is development and testing. The APEX version is usually behind the latest. Download Oracle Virtual Box and Extension manager. Setup Oracle Virtual Box and Extension manager. Import .ova file. Start appliance.

The Oracle APEX Development VM is a ready-to-use virtual machine that you may utilize by simply importing it into VirtualBox. Oracle Pre-Built Virtual Machines are available with Oracle accounts. All you need to do is install Oracle Virtual Box (Free Virtual Machine Client), and then import any appliance (Pre-Built VM). Oracle has several Pre-Built VM's available at on URL.

To set up VM, do the following:

- Download and install Oracle VM VirtualBox on your host system.
- Download the files (the use of a download manager is highly recommended)
- Import your VM: File > Import Appliance to launch Appliance Import Wizard. Click Choose... to browse to the directory you re-assembled all the files in and select the OTN_Developer_Day_VM.ova. Then click Next to begin importing the virtual machine. It will prompt you to agree to the appropriate developer licenses while importing. You will see 'Oracle Developer Days (Powered Off) when it is finished importing.
- Test your VM: Once the import has completed, double-click the OTN Developer Days VM. Click OK to close the Virtualbox Information dialogs. When you get to the Enterprise Linux 6 screen you can now login. (Username and password is oracle.) Allow the process to complete; it is ready when you see a terminal window, which you can close. Once you are finished working in the guest VM you can shut it down via System > Shut Down; this will return the guest VM to the Powered Off state.

The key benefits of utilizing Virtual Machines include:

- Utilizing new Oracle software without having to perform complex installations or needing vendor involvement.
- Experimenting without exposing your workstation.
- Free guided training provided by Oracle in the form of Hands On Labs.

1.2.5 APEX docker

Get official APEX docker on Oracle site. Navigate your browser to this URL.

Oracle APEX can be installed as Docker container. Dockers are very convenient way for developers since they can be installed on PCs and laptops. We strongly recommend to get an APEX docker only from trustworthy sources. Setup is usually quick and simple. For learning Oracle Database and APEX their version might not be important but for production use take attention to versions. The rest of this section describe one possible way to install APEX docker locally on your computer. In this process we clone the official Oracle docker image which contain development tools.

1. First step is to install docker on the machine. Docker can be easily downloaded and installed from the docker homepage: `https://docs.docker.com/desktop/install/windows-install/`
2. Second step is to sign up for a free account on the oracle container registry: `https://container-registry.oracle.com/ords/f?p=113:10` Docker images are pulled out from this registry (see Figure 1.7).

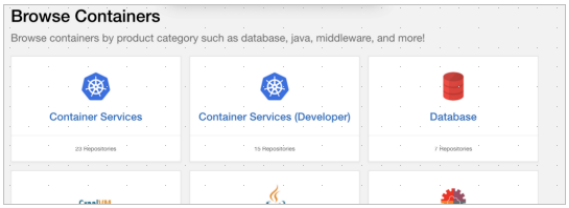


Figure 1.7: Searching for the APEX docker.

3. Third step is to open a terminal window and access the Oracle registry using the previously created user.

   ```
   docker login container-registry.oracle.com
   ```

   It's better to create a network in a docker environment so dockers can communicate with other dockers using a hostname.

   ```
   docker network create ords-database-network
   ```

   At this point, we have done all that we need and can execute the command to run our Oracle Database XE in a docker on our laptop.

   ```
   docker run -d --name testapex --hostname database
     --network=ords-database-network -p 1521:1521
     container-registry.oracle.com/database/express:latest
   ```

   Note that the parameter name, hostname and network were used here. Explanation:
   - The first flag -d will run the container in a detached mode.
   - The parameter –name specifies the container name.
   - -p maps the port 1521 on the host machine to the port 1521 in the container, so we can connect to the database.
   - –network connects the container to the network we created.
   - –hostname give a name to the DB server.
   - The last parameter is the image we want to use to spin up the container.

   If you want to connect by sqlplus you can execute this command:

   ```
   docker exec -it -u oracle testapex sqlplus / as sysdba
   ```

   With the *show pdbs* command you will check the status of pdbs. Change the default password using the below command (in this case the new password will be Welcome1!!). Before execute this command please be sure the pdb XEPDB1 is open for read and write).

   ```
   docker exec testapex ./setPassword.sh Welcome1!!
   ```

   To set up Apex for Oracle XE installation in docker, perform further steps. From the terminal session pull the image of ords.

   ```
   docker login container-registry.oracle.com
   ```

   Then execute the pull command to download the ords image.

   ```
   docker pull container-registry.oracle.com/database/ords:latest
   ```

   To configure the Apex installation, create a directory.

   ```
   mkdir ~/APEX
   ```

Put the string information inside a file on this directory.

```
echo 'CONN_STRING=sys/Welcome1##@database:1521/XEPDB1' >
~/APEX/conn_string.txt
```

Note that the parameter hostname of Oracle Database XE docker and the service of the pdb for the string connection were used. Define CONN_STRING variable as follows (needs to be in below shape, without single quote):

```
CONN_STRING=user/password@host:port/service_name
```

At this point, we can run the docker.

```
docker run --rm --name apex
-v /Users/lbindi_it/APEX:/opt/oracle/variables
--network=ords-database-network -p 8181:8181
container-registry.oracle.com/database/ords:latest
```

The same parameter network was used here. To monitor the installation, you can open another terminal session and execute this command:

```
docker run - rm - name apex
-v /Users/lbindi_it/APEX:/opt/oracle/variables
--network=ords-database-network -p 8181:8181
container-registry.oracle.com/database/ords:latest
```

Change the password for APEX_PUBLIC_USER user inside the database. To do that login in a Oracle XE docker by sqlplus.

```
sqlplus sys/Welcome1##@//localhost:1521/XEPDB1 as sysdba
```

Set the password.

```
alter user APEX_PUBLIC_USER identified by Welcome1##;
```

Now you can connect to APEX environment on your local computer with browser.

```
http://localhost:8181/ords
```

1.2.6 APEX instance in Oracle Cloud Infrastructure

Oracle Cloud Infrastructure (OCI) offers APEX low-code application development on the Autonomous Infrastructure as a fully managed service that is pre-configured and ready to use. OCI provides elastic scalability, security, high availability and global access via regional cloud data centres. In order to create an APEX Service instance, the procedures for gaining access to Oracle Cloud Infrastructure (OCI) are described in this section. The steps involved in accessing OCI are as follows:

1. **Get an OCI account**. User must have an OCI account or have access to an OCI account in order to use the APEX Service. User can use an existing OCI account if their organization has a sales agreement with Oracle. To seek access, first get in touch with the OCI administrator for your company. Consider Oracle Cloud Free Tier if you are a single user beginning from scratch or are unsure of where to begin. As well as a free initial allocation of Cloud Credits, this offers a free non-expiring OCI tenancy and account. During a trial term, these credits can be used to purchase paid OCI services, such as APEX Service. You must upgrade the account to paying status and buy more credits if you want to keep using APEX Service after the trial time has ended or the free credits have run out (whichever happens first).

Your OCI account will change to a condition where it can only utilize OCI services that have a tiny Always Free form available if you don't upgrade before the trial expires. A minor Always Free shape is there in APEX Service. Oracle advises that you convert your account to a paying one and buy more credits either during or after the trial period. Begin the signup process, by reviewing Oracle Cloud Infrastructure Free Tier. To start signup, go to https://signup.oraclecloud.com/.

2. **Sign in to the OCI Console.** Use a compatible web browser to find the OCI Console Sign-In Page. Enter your login and password, followed by the name of your cloud account (also known as your tenancy name). Your welcome email contains both your user name and cloud account name.

1.2.7 APEX instance in Oracle Academy

Oracle Academy provides Institutional members and their students with access to Oracle Application Express (APEX) for hands-on practice in the cloud. Oracle Application Express (APEX) is made available by Oracle Academy to Institutional members and their students to experiment using the cloud. In order to support labs and applied practice for curriculum, which includes Database Foundations, Database Design and Programming with SQL, Programming with PL/SQL, and Oracle Application Express—Application Development Foundations, Oracle Academy offers educators a dedicated instance designed specifically for classroom use with up to 99 students. To access Oracle APEX through Oracle Academy, users are required to log in to the Member Hub. If they are not members, users should sign up as an Institutional Members for free in order to access all material and receive other benefits. Please navigate your browser to https://academy.oracle.com/en/oa-web-overview.html.

1.3 Questions

1. What are the stages of application development cycle and what are activities of the third stage?
2. Which of the possibilities to start with Oracle Application Express is the best solution for absolute beginner?
3. What are benefits of using APEX instance in Oracle Cloud Infrastructure?

1.4 Answers

1. Stages of application development cycle are: (i) planning (defining the problem and project), (ii) application design, (iii) construction of the application, (iv) Implementation of the application. Activities of the third phase (construction of the application) are: database design, development (prototype) of the application and testing.
2. Recommendation for absolute beginner is to use on-premise instance or apex.oracle.com or Oracle Academy.
3. Oracle Cloud Infrastructure (OCI) offers APEX low-code application development on the Autonomous Infrastructure as a fully managed service that is pre-configured and ready to use. OCI provides elastic scalability, security, high availability and global access via regional cloud data centres.

2. How to prepare a database?

E. KAPSAMMER, W. RETSCHITZEGGER, W. SCHWINGER

Assume you work for *Walmart* and *Database Systems* (DBS) have not yet been invented. Assume further, you are asked to implement a *Web application* that can store, retrieve, visualize and further process every single sale in every of Walmart's 10.500 stores along with information about responsible departments, employees and their jobs which could be Petabytes of data. This task becomes unmanageable due to numerous challenges:

- How many files and disks do you need for storage?
- How do you find and retrieve data?
- How do you ensure adequate response times?
- How do you allow modifications and concurrent access to the data?
- How do you prevent unauthorized access to the data?

If you would employ a DBS as the constituting backbone of your Web application (cf. Figure 2.1) those challenges get manageable since DBS provide the proper functionality to deal with them.



Figure 2.1: DBS as the Backbone of Web Applications.

A DBS comprises:

1. the software to assist in managing and processing large collections of data which can be used by a Web application, called Database Management Systems (DBMS)

2. the actual storage of the data itself, called database, which provides the DB-layer where the Web application builds upon.

In the following, the main concepts needed for building up the DB-layer, i.e., *developing a DB*, are discussed in more detail. Although, ultimately the goal is "*creating appropriate DB tables, storing data, manipulating and querying it*", there are some important concepts which first have to be introduced in order to achieve this goal.

In particular, we will start in Section 2.1 with the process of modeling the data to be stored by introducing two different *abstraction levels* , comprising the *logical model* and the *physical model* in terms of the *relational model*[1]. Further on, in Section 2.2, DBS-specific mechanisms and tools for *creating the data structure within the DBS* are focused on, as prescribed by the modeling process. Based on that, Section 2.3 deals with data storage and manipulation issues and Section 2.4 aims at discussing the querying of stored data. Finally, Section 2.5 provides a summary of all options and the according tooling which is available in order to build up the DB-layer of a Web application.

Throughout all these sections, a *running example* is employed, inspired by the Walmart use case introduced above, emphasizing its *Human Resource (HR)* aspect in terms of *employees*, *departments* and *jobs* - a standard example which has been used by ORACLE for demonstration purposes since the first version of its DBS in 1979.

2.1 How to Model the Data

The first step when realizing the DB-Layer of a Web application is to focus on the real-world data which should be stored. Since this data can be quite comprehensive as we saw in our Walmart example above, it makes sense to deal with the data at a more abstract level first, not yet considering all low-level implementation details of a certain DBS. This is the focus of the logical model as discussed in Section 2.1.1. After that, Section 2.1.2 introduces the *relational model* at a more concrete and thus DBS-specific level.

2.1.1 Logical Model

Motivation. A *logical (data) model* describes "things" of the *real world* (i.e., the *problem domain*) about which, e.g., an organization like Walmart, wants to collect and process data. Logical models are visually described using diagrams, in practice often in terms of so-called *Entity-Relationship (ER)-diagrams*, without considering the specifics of a certain DBS. The benefits of this visual representation are manifold, e.g., enhancing understandability and facilitating communication within a development team and to the customer.

There are numerous different graphical formalisms and tools available for modeling ER-diagrams. We will focus on *ORACLE's SQL Developer Data Modeler (ODM)*, only. Figure 2.2 shows an ER-diagram of our HR example which will be explained in more detail in the following.



Figure 2.2: Logical Model of our Running HR Example using an ER-Diagram.

[1]Note that for realizing the physical model, not only the relational model can be used which is the focus of this Chapter, but also other data representation formats (aka. "data models") are available, like the *object-oriented data model* or *NoSQL data models*.

The same example is shown in Figures 2.3, 2.4 and 2.5, this time, however within ORACLE's ODM. All concepts described are annotated in order to explicate the most important graphical notations and symbols.

ER-Diagrams – Entities, Attributes, Relationships. *ER-diagrams* describe the concepts of interest and consist of (i) *entities* (e.g., Departments, Employees, Jobs) (ii) *attributes* (e.g., "departmentName" and "locationName" of departments or "lastName" and "salary" of employees) their characteristics, and (iii) *relationships* (e.g., a department "comprises" several employees) between entities .

Entities. An *entity* has a unique noun phrase assigned as its name and is visually represented as a rounded rectangle. Plural and singular forms are used however recently singular forms are preferable (and also standardized by ISO 11179-5) .

Attributes. Attributes are always associated with entities, i.e., they cannot exist independently. In contrast to entities, an *attribute* is "*atomic*", meaning that it cannot be further structured like an entity, being represented as a singular noun phrase. For each attribute, a certain *domain* (aka. *data type*) can be assigned, denoting the "nature" of the allowed values. The most common ones are numerical (`NUMERIC`), character (`VARCHAR`) and date/time (`DATE/TIME`).

Besides characterizing an entity, certain attributes can serve to *uniquely identify each instance* of an entity (e.g., a concrete employee or a concrete department). Such identifying attributes act as a so-called "*Primary Key (PK)*". For exampple, when descrining a department, some artificial ID can be defined as PK (which should be numeric, e.g., a "departmentID") or alternatively the combination of existing attributes like "departmentName" and "location", building up a so-called "*composite PK*". It is advisable to define a PK for each entity, since otherwise, entities with the same name (e.g., employees or departments) cannot be easily distinguished when retrieving the data. Thus, each value of a PK attribute has to be unique and of course, must exist.

Finally, it has to be mentioned that besides domains and PKs, also other so-called *constraints* on data values can be specified for particular attributes, e.g., `NOT NULL`, i.e., "mandatory" in order to require the existence of a value (e.g., for an attribute salary).

Relationship. Finally, a *relationship* associates (most commonly) two different entities, is visually depicted by a line between the entities' rectangles and named by a verb phrase above the line. There are foremost three different kinds of relationships which are commonly used in practice, distinguished by the *number of entity instances* which are allowed to be part in the relationship at each end called its "*cardinality*" (thus representing another kind of constraint on data that can be specified):

- **One-to-Many ("1:N"):** One entity instance can be related to many entity instances at the "many-side" of the relationship (e.g., one department has many employees, i.e., a "haveEmployees" relationship) and an entity on the "many-side" of the relationship (an employee) can be related to one entity at the "one-side"(a department) only. Such a relationship is called a "*Source-Target*" or "*Parent-Child*" *relationship*, the source entity at the "one-side" acting as the "Parent" and the arbitrary number of target entities at the "many-side" acting as its "Childs". In the above example, department is the Parent and employees are the Child entities.
- **Many-to-Many ("M:N"):** One source entity instance can be related to *many target entity instances* and one target entity instance can be related to *many source entity instances*. An example would be Employees having Jobs (i.e., a "haveJobs"-relationship), since an employee can have more than one job over time and a certain job is probably held by many employees. It should be noted that "M:N" relationship will always produce two "1:N" relationships as we will progress in database modeling.
- **One-to-One ("1:1"):** Finally, one entity instance can also be related to *a single entity instance* at the other end of the relationship, only and vice versa (e.g., a person "livesAt" a particular address and this address is the address of this particular person, only). It has to

be noted, however, that this kind of relationship is being often favoured by a more compact representation within one entity only (e.g., entity persons also cover address attributes like "street" and "city") and therefore not considered any further in this Chapter.

- **Optionality:** For each of these three different kinds of relationships, it is also possible to specify, at each end for source and target, if the *existence of at least 1 entity instance is optional or not*. For example, if a new department is founded, it could be that there are not yet any employees hired for that department, i.e., the target of the "haveEmployees"-relationship has to be defined as optional. This is also depicted by the dotted line in ODM.

Relationships are termed in ODM simply as "relations" which should not be confounded with "relations" of the "relational model" which are in fact a synonym for "tables" (cf. Section 2.1.2). Therefore, we further adhere to the more common term "relationship".

GUI-Elements of ODM for Managing ER-Diagrams. The following Figures provide a first impression of the most important GUI elements of the ODM. First, Figure 2.3 illustrates how to define new entities and new relationships, as well as the visualization of the logical model in form of a list view and a graphical view. Within the graphical view, the most important graphical notations and symbols are annotated.

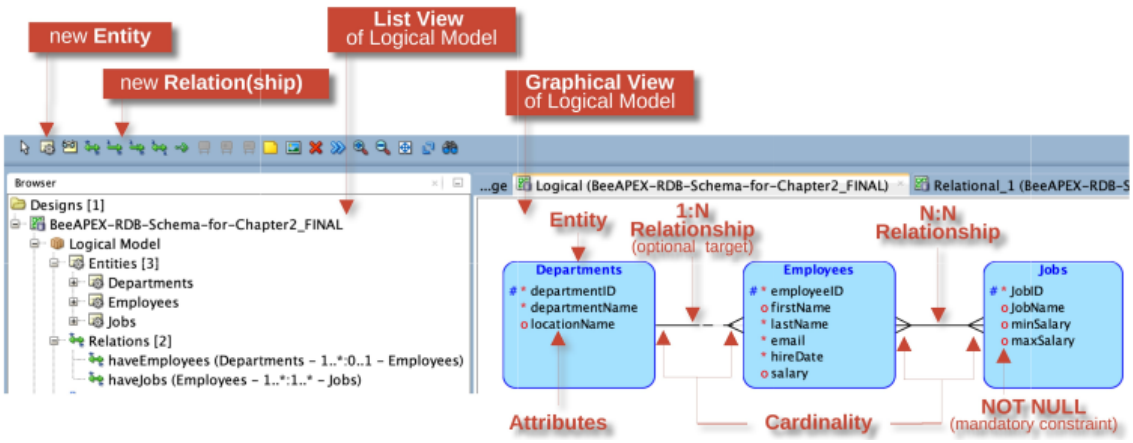


Figure 2.3: Managing the Logical Model with ODM.

Figure 2.4 shows the dialogue for the definition of attributes, which can be opened by simply double clicking on a certain entity.

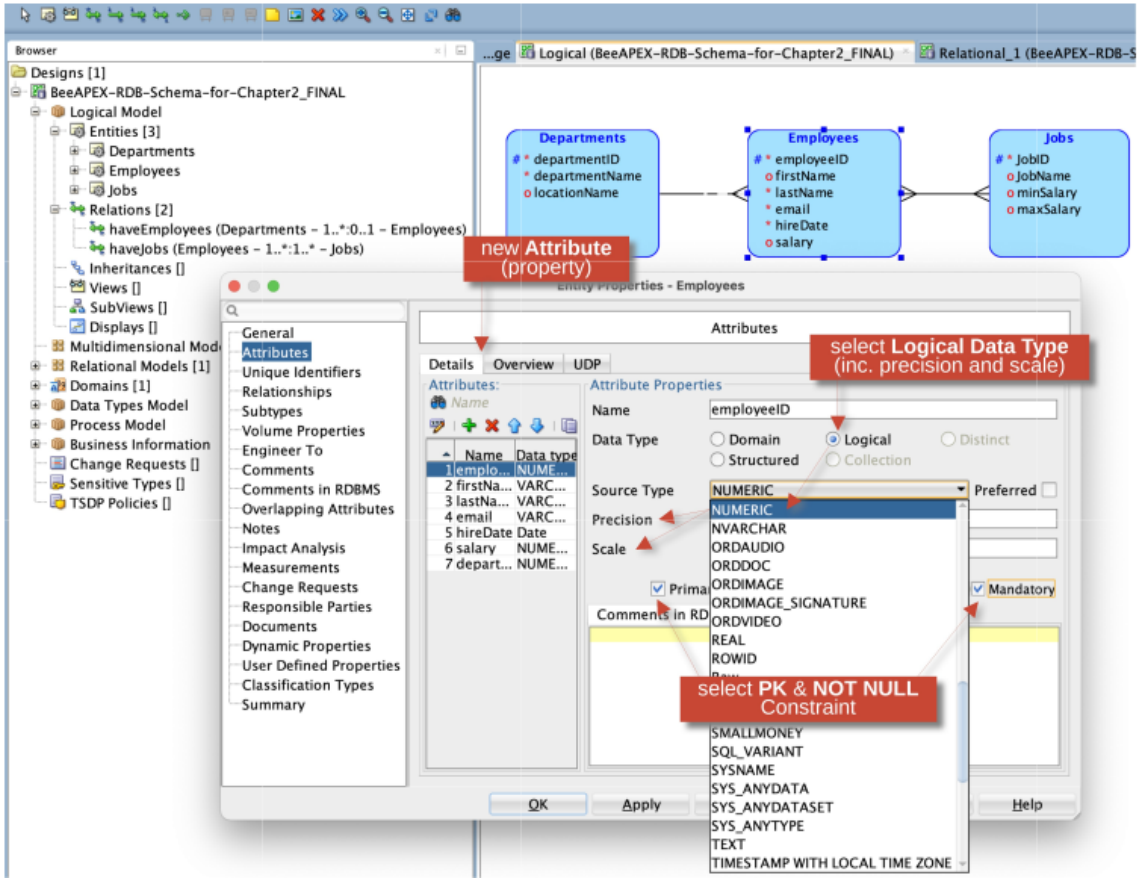


Figure 2.4: Managing Attributes with ODM.

Figure 2.5 shows the dialogue for the definition of relationships, which can be started by simply double clicking on a certain line, representing the relationship.

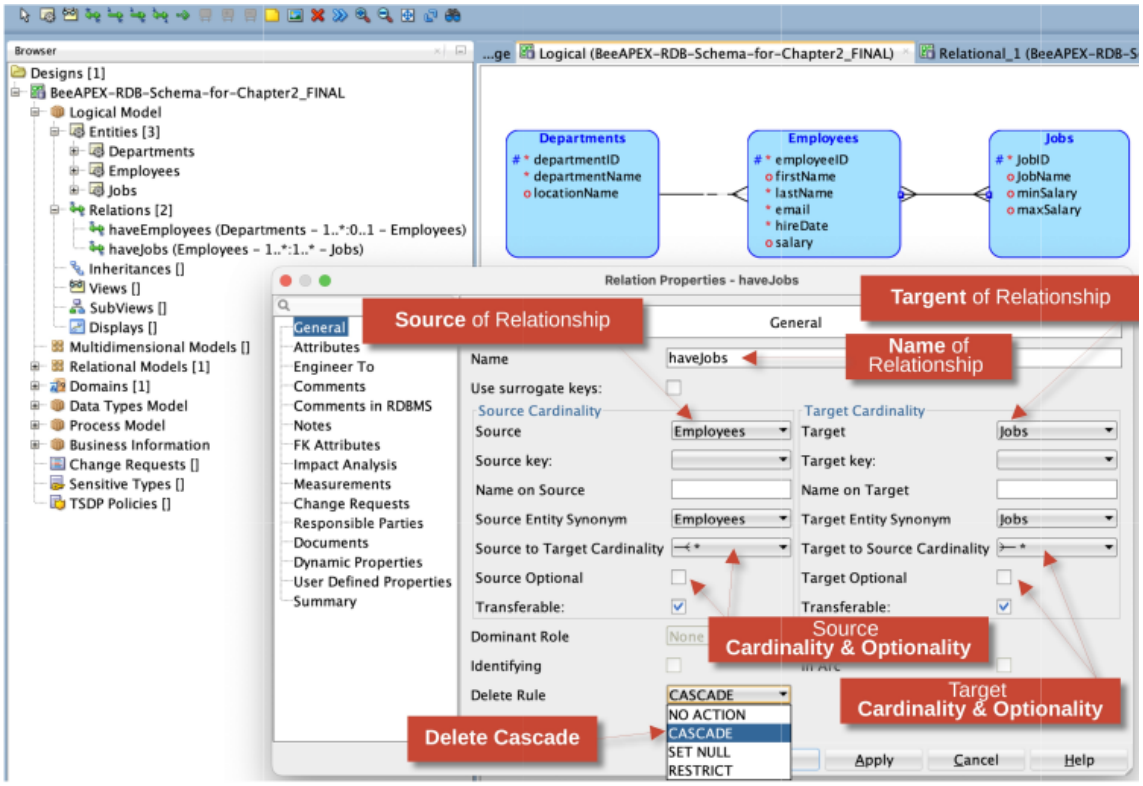


Figure 2.5: Managing Relationships with ODM.

2.1.2 Relational Model

DB Schema. The logical model in terms of an ER-diagram being independent of a specific DBS is the basis in order to derive the DBS-specific structure of the DB called a "*DB-schema*". Within a specific DBS, it is possible to create and store an arbitrary number of different schemes for different users and/or domains, e.g., one for managing HR data and another schema for managing data about product sales. A schema acts similar to a folder in a file system allowing to group data for specific purpose.

Relational DB Schema. A DB-specific kind of schema is the so-called *Relational DB (RDB)-schema* which uses the *Relational Model (RM)* as the basic formalism to describe the structure of data. The RM is quite simple since it represents the DB schema as a collection of *relations* aka. *tables*, which resemble the concept of entities. Tables are derived from entities in a straightforward way. Analogous to an entity, a table consists of a name, *columns* (resembling *attributes*), their data types and further (optional) constraints. All tables make up the *RDB-schema* (cf. Figure 2.6). The schema of a table is further on the "vehicle" in order to store the actual data, whereas each row in a table is called "*tuple*" which consists of a collection of related data values stored within the columns, thereby representing a real-world *entity instance* in terms of *relational* (*tabular*) data.

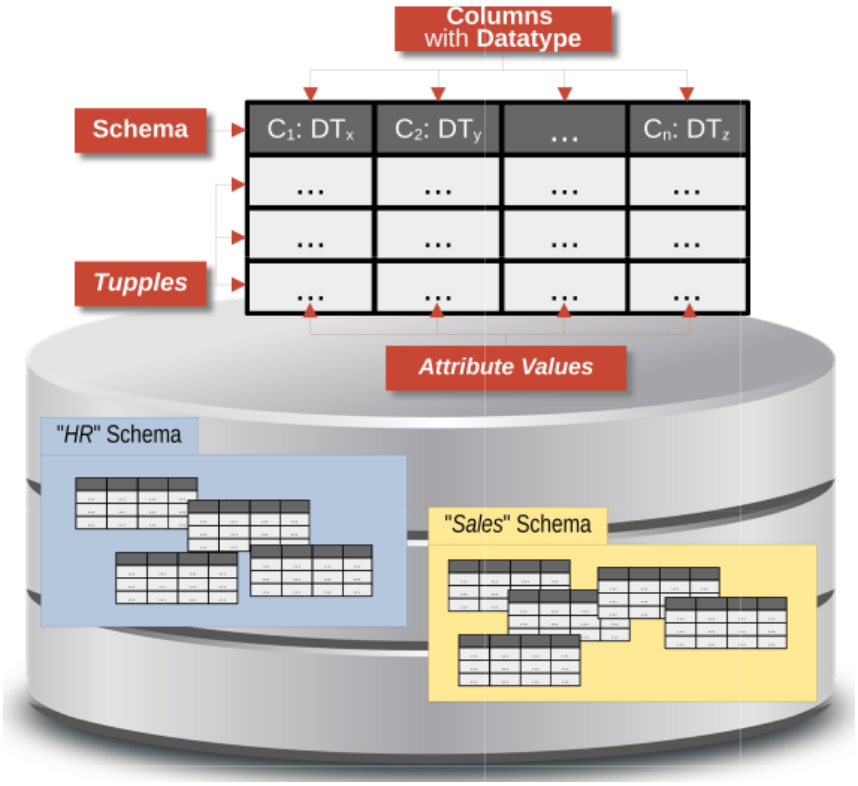


Figure 2.6: RDB-Schemata and Table Schemata.

It is interesting to note that *ODM* allows you to automatically generate a RDB-Schema out of the logical model in terms of tables, thus reflecting the entities of our logical model and all attributes and constraints of each entity. This is done by using the *"Engineer to Relational Model"* menu entry as depicted in Figure 2.7. The resulting RDB-Schema can be visualized graphically using ER-diagram formalism, based on a more "table-oriented-notation" than the ER-diagram of the logical model, showing now, e.g., also the data types of all attributes.

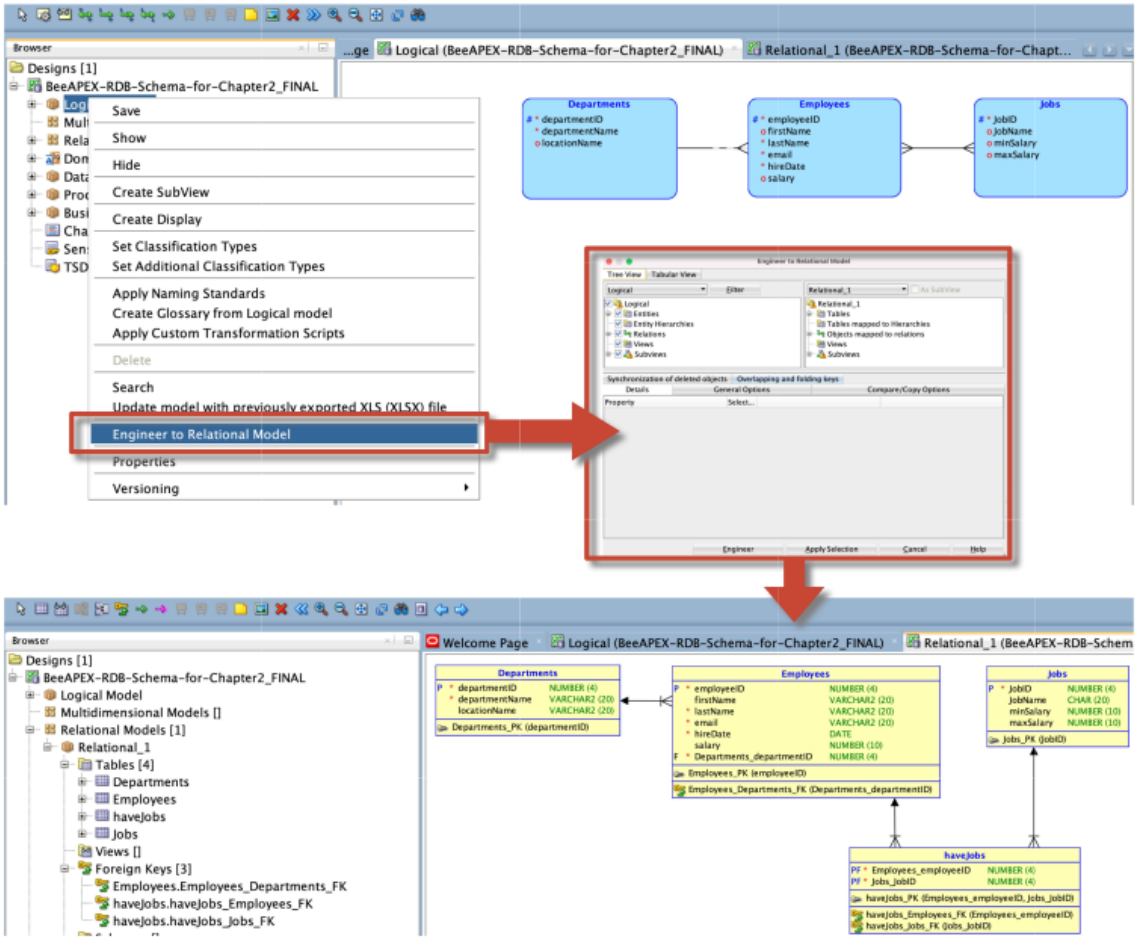


Figure 2.7: Process for RDB-Schema Generation out of the Logical Model.

Now, let's focus on our exemplary relational model depicted in Figure 2.8.

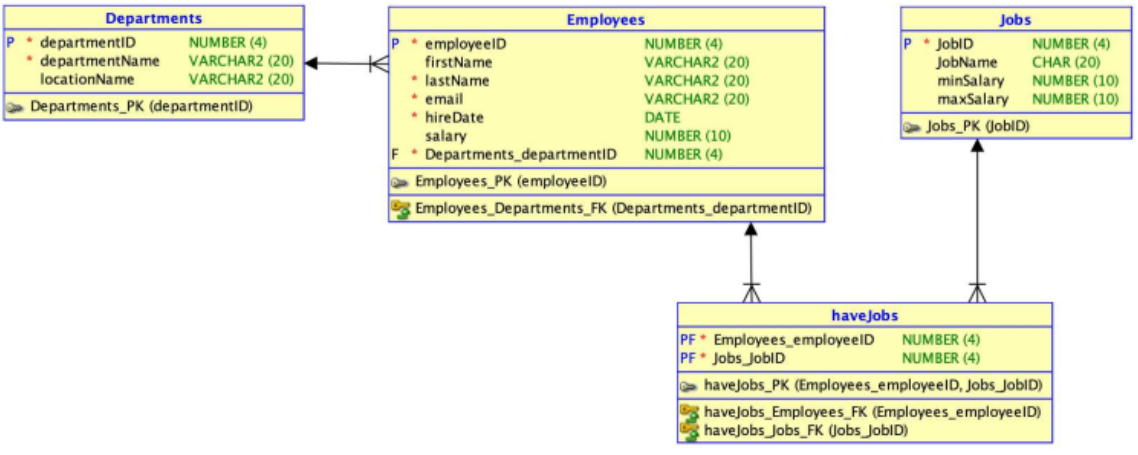


Figure 2.8: Automatically Generated RDB-Schema.

First of all, during the process of generating the relational model out of the logical model, besides transforming existing entities and attributes into tables and corresponding columns, PK's are automatically added to each table and given a name (cf., e.g., "Departments_PK (departmentID)". In addition, the relationships between entities within the logical model are automatically represented within the tables, using the so-called "*Foreign-Key (FK)*" *concept* in the following way:

- **One-to-Many (1:N) Relationship:** Within the table at the "Many-side" of the relationship (i.e., the Child table), an additional attribute is introduced and declared as FK, acting as a kind of reference to the values of the PK of the relation at the "One-side" (i.e., the Parent table). If there is, e.g., an employee with PK ID 4711, then within the FK attribute of the associated department, the value 4711 also has to be present. Consequently, the FK attribute is required to have the same data type as the PK. In our example, a FK "Departments_departmentID" as well as a FK named "Employees_Departments_FK" is automatically generated. As a consequence, each time data within these two relations is manipulated in the DBS, the FK ensures *consistency* in-between these relations, i.e., correct relationship data. For instance, if a new employee is inserted into the Employees table, the DBS ensures that it is also assigned to an existing department. If you attempt to delete a department, then the DBS rejects this operation by default in case there are still associated employees. Changing this default behaviour is possible by using the so-called "*Cascade-Delete Option*", when specifying the FK, resulting in an automatic deletion (maybe "firing") of all employees associated with the deleted department.
- **Many-to-Many (M:N) Relationship:** A many-to-many relationship between two entities is realized by introducing another, third table, acting as some kind of "*mapping table*" which mainly stores the references to each of the PKs of the associated entities in terms of two FK attributes. Regarding our running example of Employees and Jobs, a third relation is automatically generated named "haveJobs", now responsible for managing the correct associations between employees and their respective jobs. The PK of this mapping table is a composite one, consisting of the PKs of the two associated tables, i.e., "haveJobs_PK(Employees_employeeID, Jobs_jobID)". Each attribute of this PK is also used for defining an appropriate FK each referencing the PK of the respective associated table (i.e., "haveJobs_Employees_FK (Employees_employeeID)" and "haveJobs_Jobs_FK(Jobs_JobID)"). It has to be noted, that this mapping table can also have an arbitrary number of additional attributes, further characterizing the relationship, e.g., "startDate" and "endDate" of a job.
- **Optionality:** Per default, for each of these relationships, the existence of at least a single source and a single target instance is optional. Should it be mandatory, then a further NOT NULL constraint has to be associated with the according FK-attribute. Considering our example, all three FKs need a NOT NULL constraint since (i) Employees have to be associated to a certain Department (ii) as well as to at least one Job and (iii) a Job has to be associated to at least one Employee.

GUI-Elements of ODM for Managing the RM. The following Figures give an overview of the most important GUI elements of the *ODM* for managing the RDB-Schema which are quite similar to those for managing the logical model. First, Figure 2.9 illustrates how to define new tables and new FKs, as well as the visualization of the RM in form of a list view and a graphical view. Within the graphical view, the most important graphical notations and symbols are annotated.

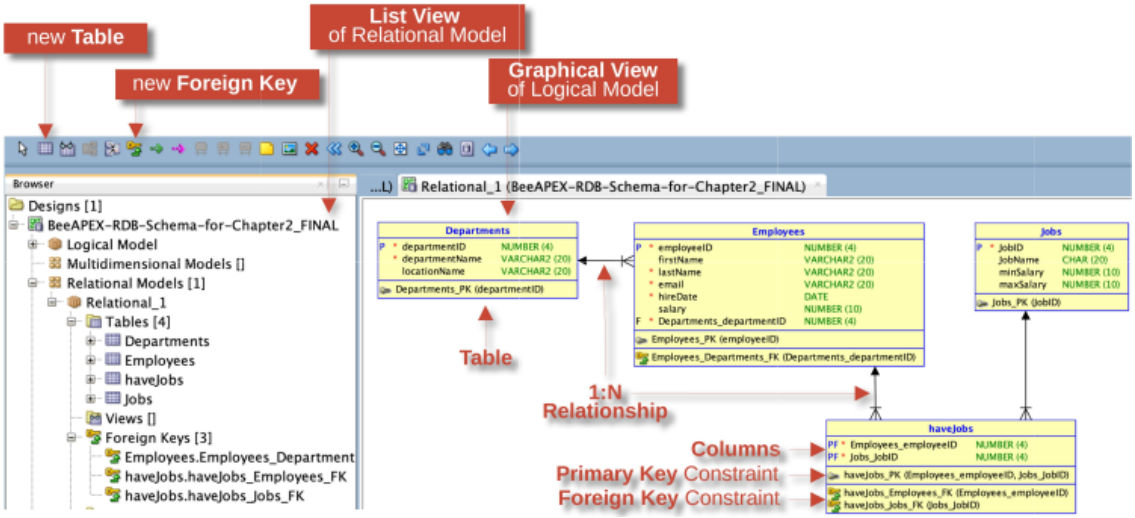


Figure 2.9: GUI-Elements for Managing Columns of the RM.

Figure 2.10 shows the dialogue for the definition of columns, which can be started by simply double clicking on a certain table. Note that this dialogue largely resembles the dialogue for managing attributes as already depicted in Figure 2.4.

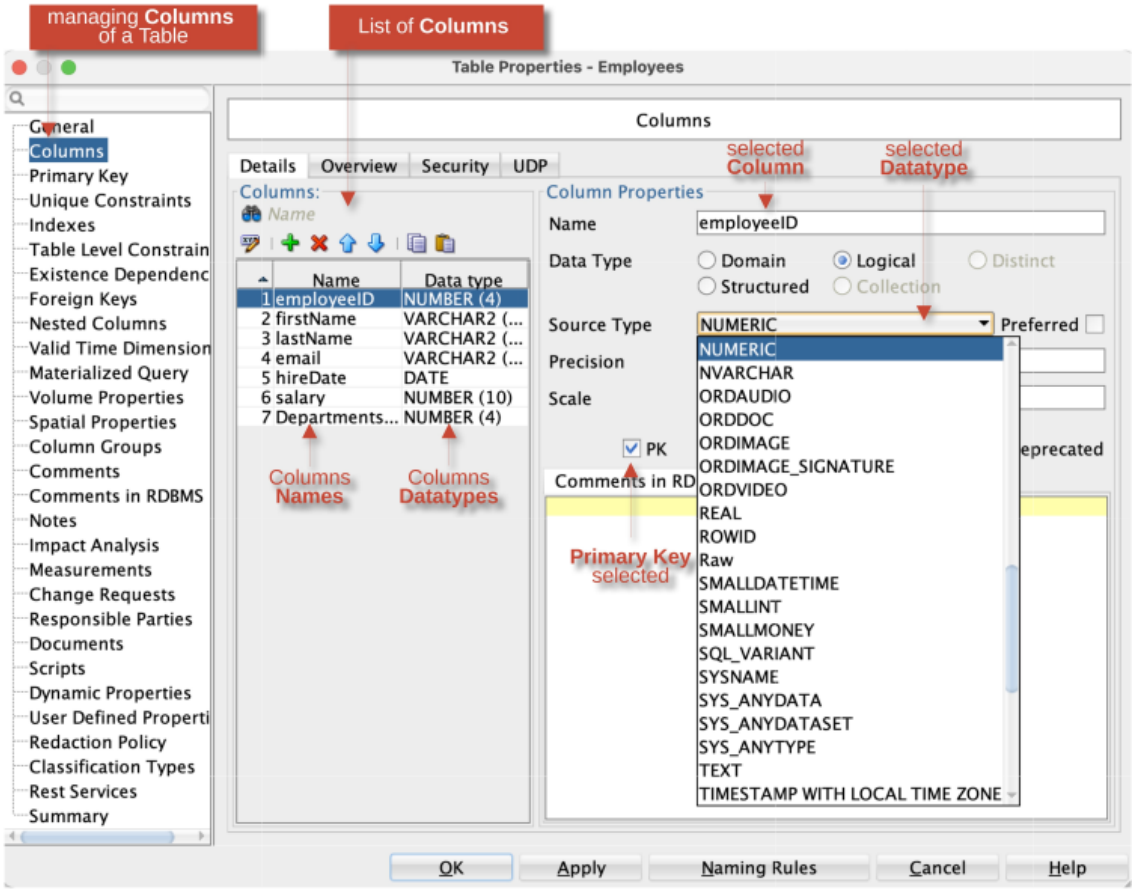


Figure 2.10: GUI-Elements for Managing the RM.

RDB-Schema independent of Physical Storage Aspects. Although a RDB-schema based on the relational model is already more concrete and DBS-specific than the logical model (e.g.,

"implementing" relationships based on the FK-concept) it has to be noted that it still hides the complexity of the underlying storage mechanisms of the DBS (aka. "*physical*" *level*), e.g., how many files are used in order to store the tables or on which servers the data is to be stored. Another big benefit of this abstraction from the physical storage aspects is that new RDB-schemas can be realized or existing ones changed without haven to care about the underlying physical data organization.

2.1.3 Normalization of an RDB-Schema

An RDB-Schema which has been derived from a logical one can often be further improved in order to better achieve certain *quality criteria* which are prevalent in DBS, being, on the one hand side best suited for *data read access* in terms of DB-queries but, providing on the other hand side also a proper basis for *data manipulation* in terms of inserting, updating and deleting data. Although these are, to some extent, conflicting goals, there is a well-known concept available for RDBS in terms of the so-called "*normalization theory*" which describes a systematic process of three steps named "*first normal form*", "*second normal form*" and "*third normal form*", leading to improved RDB-Schemas (cf. Figure 2.11 for an overview). In the following, these three normal forms are described in more detail, using slight variations of our running example.

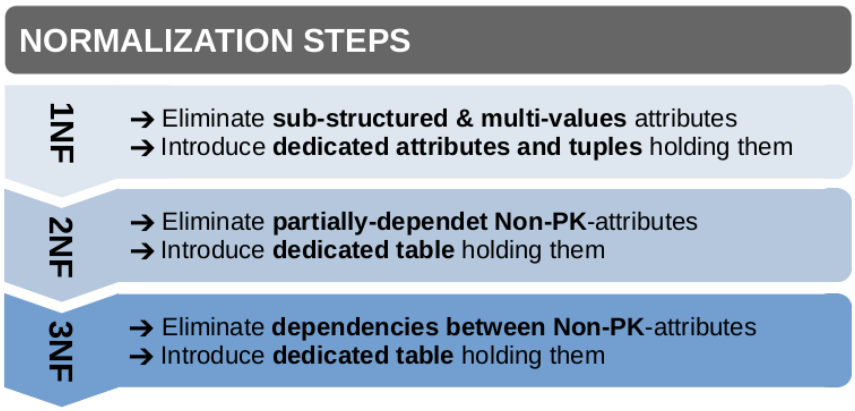


Figure 2.11: Normalization Steps – Overview.

First Normal Form. The first normal form requires that each attribute of a table is "atomic" meaning that neither (i) *sub-structures* are allowed (e.g., if, instead of our simple attribute "locationName", an alternative attribute "location" would contain values about street, postal code, city and country) nor (ii) *multi-values* are allowed (e.g., attribute "location" could also contain several addresses of a certain department. Sub-structures within values can be *eliminated* by simply representing the sub-structure with dedicated attributes (e.g., for each part of the address) and multi-values by simply using an own tuple for each of the values (i.e., putting each address in a separate row).

One main *benefit* of the first normal form is, that data queries can be more specific, since now, e.g., each part of the address as well as each address tuple can be separately accessed.

Second Normal Form. The second normal form requires (i) that a table is in the *first normal form* and (ii) in case of a composite PK, i.e., consisting of two or more attributes, all other non-PK attribute values are *uniquely identified by the whole composite PK only*. For example, assume that the table Departments contains a composite PK with the attributes "departmentName" and "locationName". This PK correctly determines the value of a non-PK attribute "annualRevenue", since we naturally assume that the value of "annualRevenue" depends on both, the department and its location. The values of another non-PK attribute "locationAddress", however, would already depend on a part of the PK, namely the location's name. In order to *eliminate* this *partial*

dependency, we have to *split up* the table Departments, factoring out the location information into a separate table "Locations" and connect both tables via a FK.

One main *benefit* of the second normal form is that it prevents *redundant data storage*, since in our case, a certain address is stored only once and can be reused by different departments. This reduces not only storage space but foremost also prevents incorrect data, so-called "*anomalies*" if, e.g., certain data manipulations are not processed on any duplicate data (e.g., in case of several departments moving to another address, the address has only to be changed once). Finally, the good thing is that having tables with non-composite PKs only (e.g., a numerical ID for each table), the RDB-Schema "automatically" fulfills the second normal form, since partial dependencies cannot exist.

Third Normal Form. The third normal form requires (i) that a table is in the *second normal form* and (ii) that each *non-PK attribute is uniquely identified by the PK only* and not by any other non-PK attribute, leading to *transitive dependencies*. For example, if the table Departments would contain besides the PK employeeID and the locationName also a locationAddress, then locationAdress would already be uniquely identified by locationName (if we assume that each locationName is associated with a different address) which is not allowed by the third normal form. In order to eliminate this situation, location information has to be factored out into a separate table Locations and connected via a corresponding FK to the Departments table.

The *benefit* is the same as already discussed for the second normal form – reducing redundancy and thus potential anomalies in case of data manipulations.

2.2 How to Manage the RDBS-Schema – SQL-DDL

In the previous Section, the RDB-Schema has been graphically modeled on different levels of abstraction in a *low-code development fashion*. Now it is time to discuss, how these graphical models can be implemented "programmatically" within a RDBS so that they really can be used in order to manage the actual data of a Web application. The questions which will be answered in the following are therefore:

- How to advise the DBS to *create the required tables*, ideally out of the graphical models and do we have any other alternatives if no such graphical models exist?
- How does the*DBS "Programming Language"* look like which is capable of creating the tables?

SQL = DDL | DML | DQL. In order to answer the second question first, the language which is used for that and also for other purposes is the *standardized Structured Query Language (SQL)*. SQL provides a wide range of different *"DBS programming statements"* which can be, according to their purpose, grouped into different categories (cf. Figure 2.12), the most important ones being (i) statements for managing the RDB-Schema (e.g., creating, altering and dropping tables), summarized as *DDL (Data Definition Language)*, which is the focus of this Section, (ii) statements for manipulating data within the schema (e.g., inserting, updating and deleting data within tables) called *DML (Data Manipulation Language)* which will be discussed in Section 2.3 and finally (iii) statements for querying the data called *DQL (Data Query Language)*, further dealt with in Section 2.4.

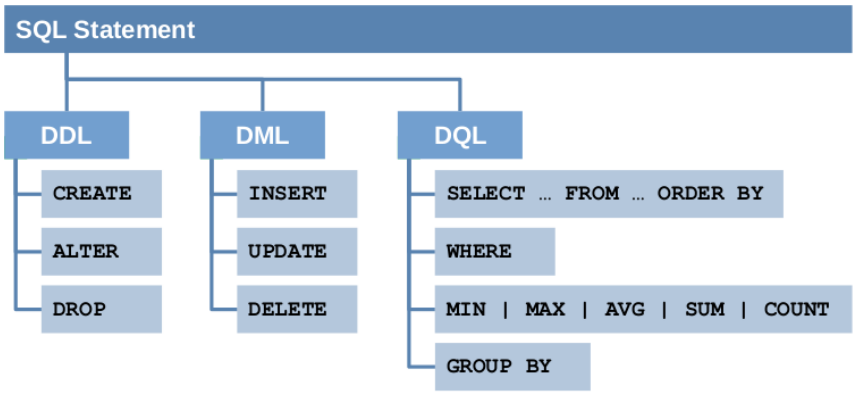


Figure 2.12: Categories of SQL Statements.

Five Options for Creating the Tables. According to the tooling provided by ORACLE, there are five different options for creating tables, i.e., implementing an RDB-Schema (cf. also Figure 2.45 for an overview):

1. **Automatic Table Generation using ODM.** This is the option with the least effort, since simply initiating an *automatic generation of the tables* by some button-clicks *using ODM* would suffice (cf. Section 2.2.1).
2. **Automatic Table Generation using Quick SQL.** Another alternative would be to use some kind of "shorthand-notation" for SQL, provided by ORACLES Quick SQL tool (cf. Section 2.2.2). This is the perfect choice if there are no graphical (logical and / or relational) models of the RDB-Schema available or if some quick testing is necessary.
3. **Manual Table Creation using SQL-DDL.** There is of course, also the possibility to ideogrammatically specify the tables using plain SQL, providing the benefits that every single detail concerning the table specification can be configured as needed and being not dependent on, sometimes sub-optimal automatic generation processes (cf. Section 2.2.3).
4. **Manual Table Creation using ORACLE Object Browser (OB).** For those users having no SQL knowledge, the Object Browser allows to create new tables and alter them if necessary in a form-based manner (cf. Section 2.2.4).
5. **Automatic Table Creation using ORACLE Data Workshop.** Finally, there is another automatic table creation possibility which can be, however, used only if there is already some data stored within external files (e.g., within EXCEL-files) using ORACLE Data Workshop. This option is not further dealt with in this Chapter but rather described in more detail in chapter 4.

In the following, the first four options are discussed in more detail.

2.2.1 Automatic Table Generation using ODM

ODM is abbreviation of the tool named Oracle SQL Developer Data Modeler. Being the table generation option with the least effort, just two steps are necessary to create the required DB tables in case that a RDB-Schema has been developed in *ODM*.

First, the modeled RDB-Schema has to be exported into a *SQL-script* (cf. Figure 2.13, steps 1 to 3), being in fact a simple text file which contains all the SQL-DDL-statements necessary to create the tables within the RDBS. Please note that the syntax of the generated SQL-statements is described in more detail in Section 2.2.3.

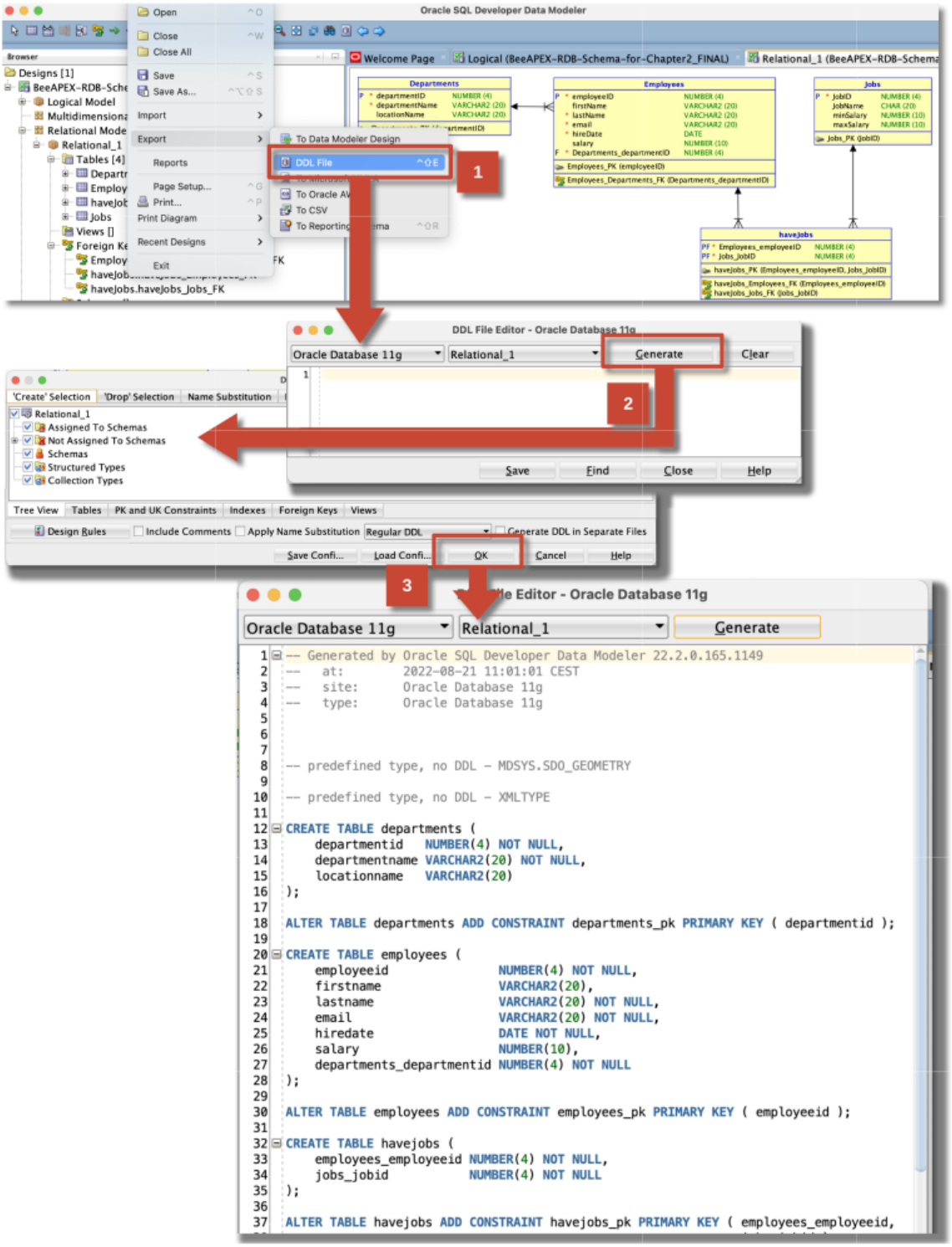


Figure 2.13: Export of the RDB-Schema into a DDL-script.

Second, the generated SQL-script has to be executed in order to advise the RDBS to actually build-up all tables as empty ones within its storage space. For that, as illustrated in Figure 2.14, the script has to be uploaded via ORACLE SQL Workshop (cf. 1-2) and simply executed (cf. 3).

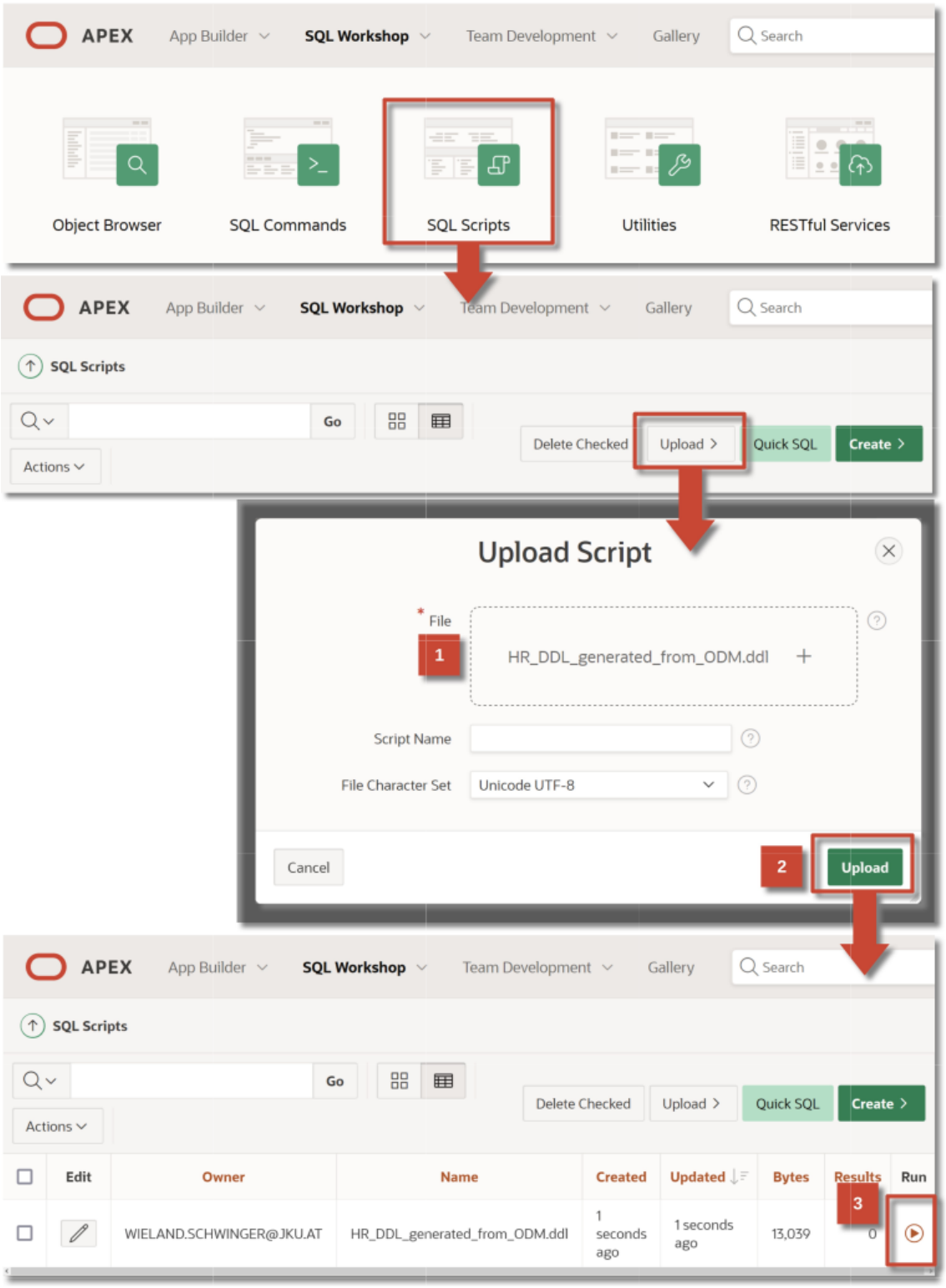


Figure 2.14: Upload and Execution of a SQL-Script.

2.2.2 Automatic Table Generation using Quick SQL

When building up the DB-layer of a Web application, it is highly advisable to follow the different abstraction levels in terms of logical and relational models. Nevertheless, *Quick SQL* provides a simple and intuitive alternative for creating tables, quite simpler than SQL, e.g., for quick testing purposes. Quick SQL further on allows to automatically generate an according SQL-script. Once

the SQL-script is generated it can be tweaked and expanded upon and finally executed to create the tables. It has to be emphasized again, that although Quick SQL is designed to reduce the effort required to create tables, it is definitely not designed to be a replacement for data modeling (cf. Section 2.1).

The ORACLE Quick SQL editor can be accessed via "Utilities" of SQL Workshop (cf. Figure 2.15.

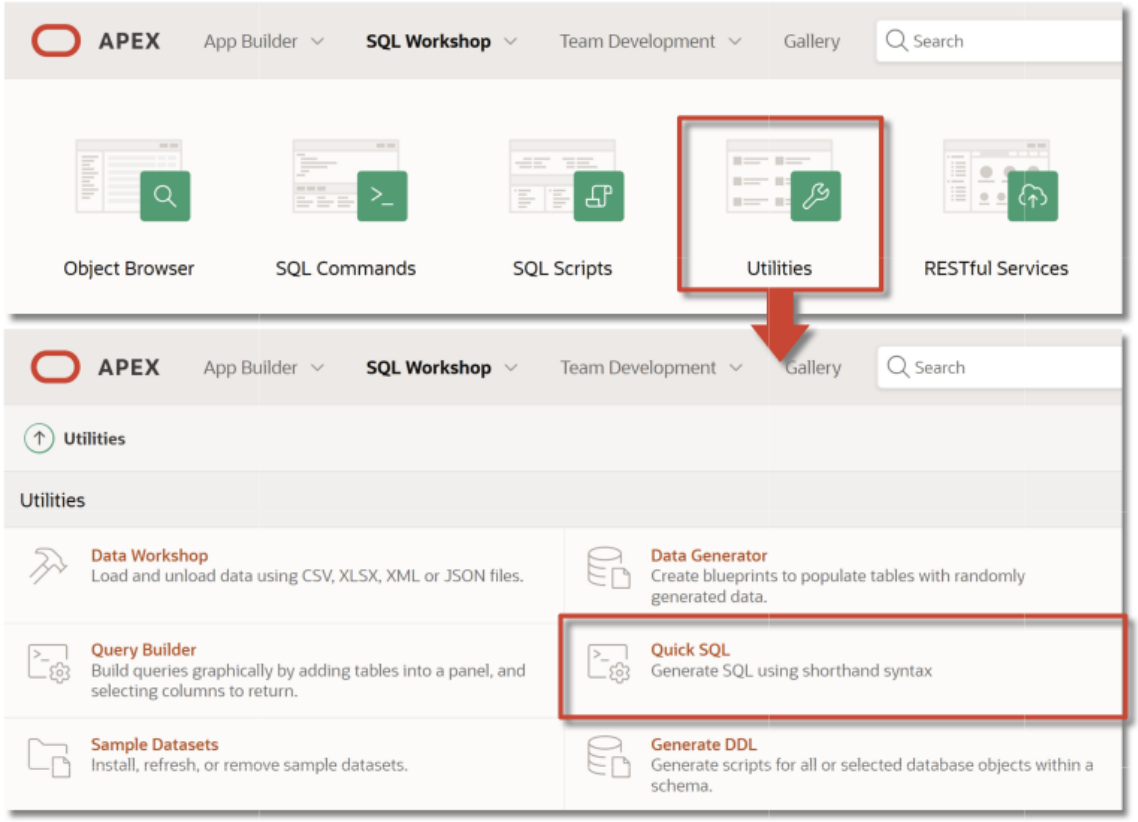


Figure 2.15: Accessing Quick SQL via SQL Workshop

Now let's stick to the simple syntax provided by Quick SQL, which basically uses a couple of formatting principles in order to specify the required table generations:

- **Parent tables:** Parent table names are entered without any indention. Table names are automatically formatted, replacing any spaces with underscores.
- **Attributes:** Attribute names are entered with a uniform indention of two or more spaces. As with table names, attribute names are automatically formatted, replacing any spaces with underscores.
- **Parent/child relationships:** Parent child relationships are entered by indenting child tables under parent tables. Child tables should be intended to the same level as the columns in the parent table.
- **Data Types:** Based on the English text contained in the column name, and in the absence of any data type specified, the RDBS automatically derives eventually appropriate data types. If a column should have a dedicated data type, the following ones are possible: `NUM`, `INT`, `VC` (for `VARCHAR2`) or `DATE`. The data type can be simply specified at the end of a column name, separated by a space. If a specific `VC` length should be defined, `VCn` has to be entered, where `n` is the length of the `VARCHAR2`. Overall, the available syntax is shown via the "Help"-menu.

Figure 2.16 shows an example of the Quick SQL notation, depicting our parent table "Departments" and the according child table "Employees on the left-hand side and the automatically

generated SQL-script in the SQL Output pane. SQL is immediately generated after each carriage return. The final SQL-script can be saved for further usage and an eventual adaption as well as executed to finally generate the empty tables.

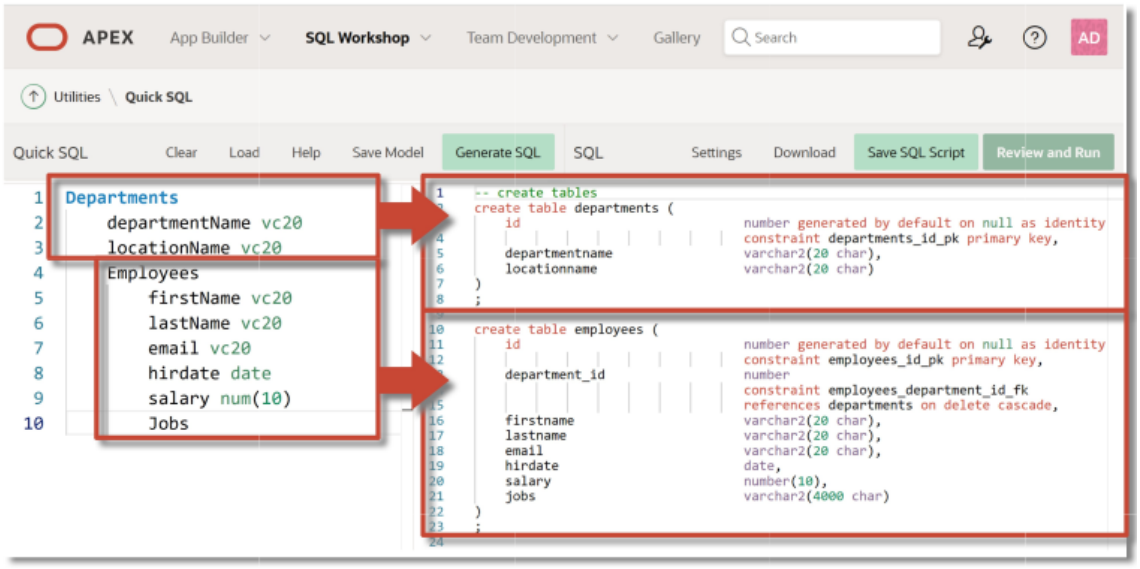


Figure 2.16: Generated SQL-Script based on Quick SQL.

Quick SQL has also syntax to define constraints (foreign key and allowed values) for specific data field as well as to generate test data.

2.2.3 Manual Table Creation using SQL-DDL

The programmatic way to create tables is to break away from any “low-code-fashioned” automatic generation of SQL-scripts (as discussed in Section 2.2.1 and Section 2.2.2) and do the work on your own. To be more concrete, one can of course also manually specify the SQL-scripts using the SQL-DDL `CREATE TABLE` statement within the SQL-Command-Editor provided by ORACLEs *SQL-Workshop* (cf. Figure 2.17). After manually defining the SQL-statements using plain SQL-DDL (1), the resulting SQL-script can be saved and executed (2) in order to create the actual empty tables (cf. Figure 2.17).

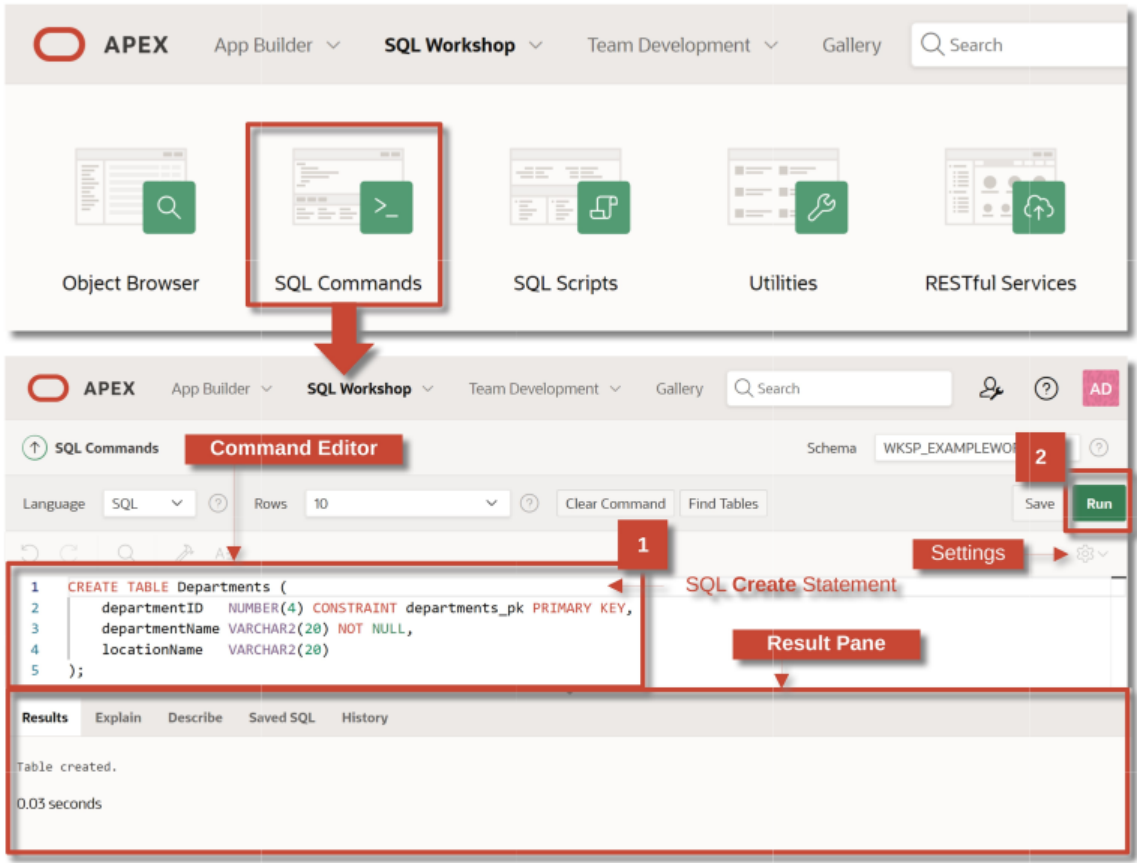


Figure 2.17: SQL Command Editor.

SQL-statements are more or less straightforward like plain English but with a specific syntax. The `CREATE TABLE` statement consists of *SQL-specific keywords* like "`CREATE`" itself as well as the *constraint definitions* (data types, primary key and foreign key). In Figure 2.18, all keywords as well as other SQL-specific syntactic symbols (like parenthesis and comma) are written in uppercase and colored in blue. It is important to note that SQL keywords are case-insensitive, meaning e.g., that "`CREATE`" means the same as "`create`". The attribute definition is enclosed within parenthesis, the attributes are separated by commas, each one ideally defined within a separate row (although these are informal formatting conventions only, not part of the SQL syntax). Constraints can be be defined using an explicit name (e.g., departments_pk), which is not obligatory (cf, e.g., `NOT NULL` for departmentName) but serves better readability purposes only. Each SQL-statement is, at the end, delimited by a semicolon.

```
CREATE TABLE Departments
(   departmentID   NUMBER(4), CONSTRAINT departments_pk PRIMARY KEY,
    departmentName VARCHAR2(20) NOT NULL,
    locationName   VARCHAR2(20)
);
```

Figure 2.18: SQL Statement for Creating Table `Departments`.

Complementing the `CREATE TABLE` statement, there is of course also a SQL-statement for deleting a table – it works properly, no matter if the table is empty or if it already contains some data, which is then deleted too (cf. Figure 2.19).

```
DROP TABLE Departments;
```

Figure 2.19: SQL Statement for Dropping Table `Departments`.

And finally, the definition of already existing tablescan also be changed, e.g., adding additional attributes or changing a data type (cf. Figure 2.20):

```
ALTER TABLE Departments
      ADD annualRevenue NUMBER(10);
```

Figure 2.20: SQL Statement for Altering Table `Departments`

If the table which should be changed already contains data, then modifications are somewhat restricted, since, e.g., changing a data type can lead to unintentional effects on this data. This issue, is however, not further dealt with in this Section.

2.2.4 Manual Table Management using Object Browser

The Object Browser in SQL Workshop allows not only to view all generated tables, so-called *Objects*in ORACLE's terminology, together with their according attributes and constraints, but also to alter them and create new ones, all based on a simple GUI-Wizard (cf. Figure 2.21).

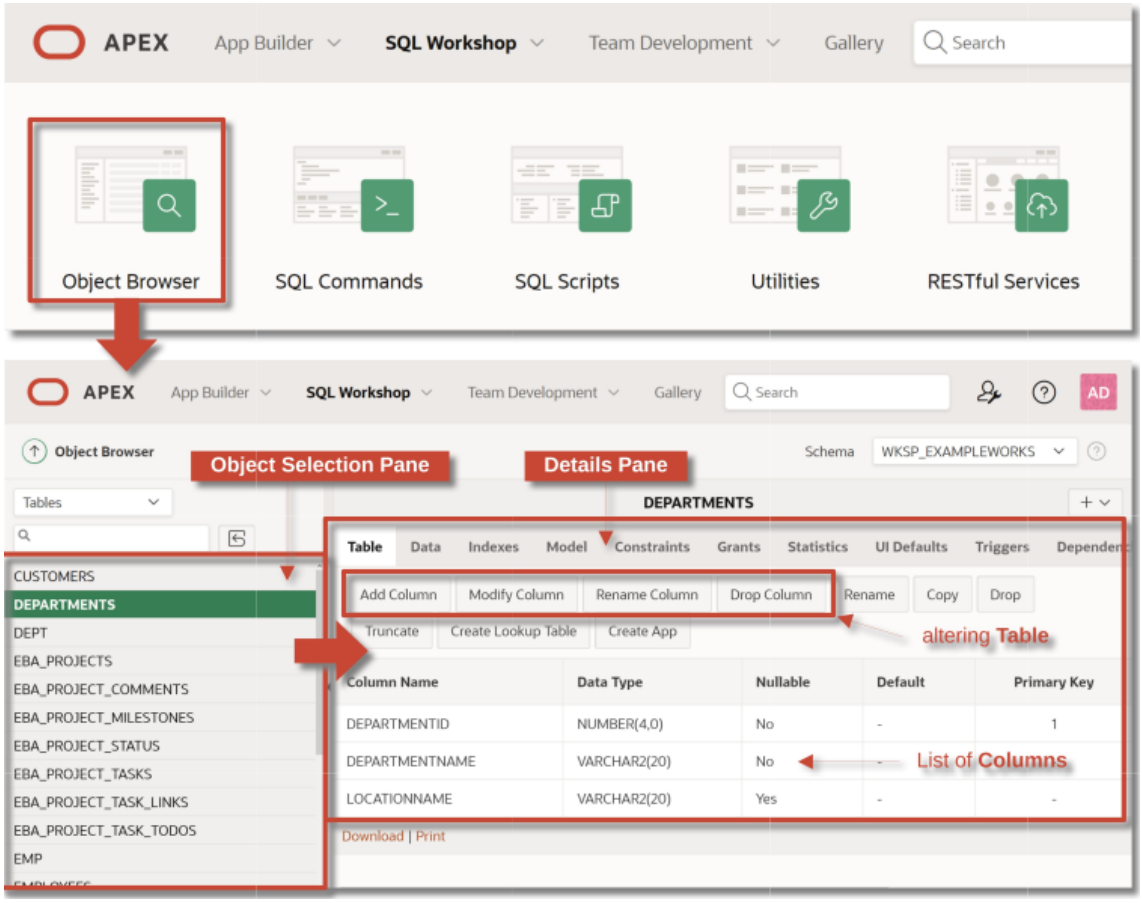


Figure 2.21: Table Management with ORACLE's Object Browser.

2.3 How to Manipulate Data – SQL-DML

The empty tables created in the previous Section, are now "waiting" to be fulfilled with according data, which could be further on modified and eventually also deleted. This is exactly the responsibility of the SQL-DML statements `INSERT`, `UPDATE` and `DELETE`, which will be briefly introduced in the following Section 2.3.1 by means of our running example. Besides that, there is also a no-code alternative available through ORACLE's Object Browser (OB), which allows to manually insert data into tables, being appropriate for users without any SQL-knowledge (cf. Section 2.3.2). Finally, a quite limited data insertion alternative is provided by Quick SQL (cf. Section 2.3.3). All these alternatives are also depicted in Figure 2.45.

2.3.1 Data Manipulation using SQL-DML

For programmatic data manipulation using SQL-DML, the SQL Command Editor is used (cf. Figure 2.17) the same which was used for creating tables. In the following, some examples are illustrated in order to give a first impression about the syntax and the functionality provided by SQL-DML.

`INSERT` is used to *add new rows* to a table as can be seen in the following example (cf. Figure 2.22):

```
INSERT INTO Departments (departmentID, departmentName, locationName)
     VALUES (10. 'ACCOUNTING', 'NEW YORK');
```

Figure 2.22: SQL Statement for Inserting new Data into Table `Departments`.

It hast to be noted, that besides this programmatic option we mentioned, in Section 2.3.3 also the possibility to use Quick SQL for inserting data, which is however, restricted to insert random data only.

`UPDATE` is used to update one or more rows of data within a table. The following example is the simplest form of an `UPDATE` statement, increasing the salary of all employees by 10% (cf. Figure 2.23):

```
UPDATE Employees
     SET salary = salary * 1.1;
```

Figure 2.23: SQL Statement for Updating the Salary of all Employees by 10% in Table `Employees`.

A slight extension is shown in the next example, adding a so-called "`WHERE`-clause" in order to apply the salary increase to the employee having "Miller" as last name (cf. Figure 2.24):

```
UPDATE Employees
     SET salary = salary * 1.1
     WHERE lastName = 'Miller';
```

Figure 2.24: SQL Statement for Updating the Salary of Employee "Miller", only.

Finally, also more than one attribute of a table can be updated by means of a single statement as can be seen in the following example (cf. Figure 2.25):

```
UPDATE Employees SET
      jobID  = '4711',
      salary = salary + 1000,
      departmentID = 120;
```

Figure 2.25: SQL Statement for Updating more than one attribute in Table `Employees`.

`DELETE` is used to remove one or more rows of data from a table. The following example is the simplest form of a `DELETE` statement, deleting all data stored in the table storing employees (cf. Figure 2.26).

```
DELETE FROM Employees;
```

Figure 2.26: SQL Statement for Deleting all Data from Table `Employees`.

Extending the previous statement with a "`WHERE`-clause" allows to selectively delete the employee with the jobID "4711" (cf. Figure 2.27).

```
DELETE FROM Employees
      WHERE jobID  = '4711';
```

Figure 2.27: SQL Statement for Deleting some Data from Table `Employees`.

2.3.2 Data Manipulation using ORACLE's Object Browser (OB)

Besides this programmatic option to manipulate data, it would also be possible to manually insert data into tables as well as to change or delete existing data using an editor provided by the *Object Browser*, cf. Section 2.2.4, Figure 2.21, using the "Data" tab. It has to be noted that all data manipulations which are done using the OB are again automatically translated by ORACLE into according SQL-DML statements.

2.3.3 Data Manipulation using Quick SQL

Using Quick SQL, it is possible to automatically insert some data into the tables. An example is shown in Figure 2.28 using the statements "`/insert 2`" and "`/insert 1`". With these statements, the number tuples having random values is defined which should be inserted into the tables.

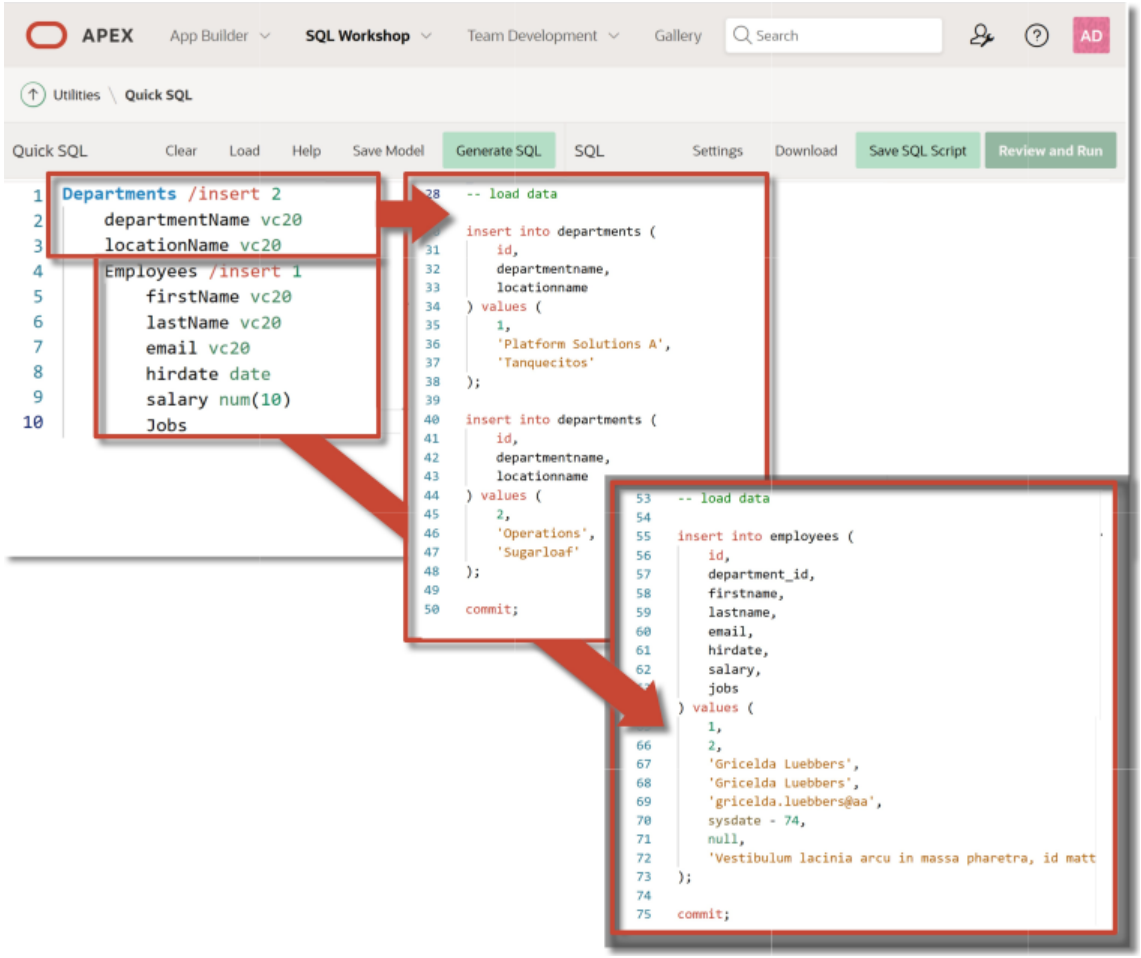


Figure 2.28: Random Data Insertion with Quick SQL.

2.4 How to Query Data – SQL-DQL

After discussing different possibilities for the creation of empty tables in Section 2.2 and the options for manipulating data in Section 2.3, we now turn to the retrieval of data out of the tables by first discussing query capabilities of SQL in terms of the "SELECT" statement (cf. in Section 2.4.1), again by means of our running example, followed by a brief overview about the ORACLE tool Query Builder ("QB") providing a "low-code" querying alternative for users having little or no SQL knowledge (cf. Section 2.4.2). These alternatives are also depicted in Figure 2.45.

2.4.1 Data Querying using SQL-DQL

First of all, it has to be noted that the expressiveness of SQL-DQL is immense, so that we will focus on a quite small subset of concepts, only. Nevertheless, we aim to give a first impression of the expressiveness and broad applicability of SQL-DQL, by introducing different querying possibilities by example, ordered in the following by increasing complexity. Regarding tooling, again the SQL Command Editor is used (cf. Figure 2.17) the same which was used for creating tables and manipulating data.

(1) Let's start with the simplest query – *"Retrieve all Employees"* (cf. Figure 2.29).

```
SELECT * FROM Employees;
```

Figure 2.29: SQL Statement for Retrieving all Data from Table `Employees`.

The target table of the query (`Employees`) is specified within the `FROM`-clause, the required result of the query is defined right after the `SELECT`-clause - in this example using the star as a joker sign – meaning we would like to get the values of all columns of the table.

An excerpt of the result of this query looks as follows (cf. Figure 2.30):

EMPLOYEEID	FIRSTNAME	LASTNAME	EMAIL	HIREDATE	SALARY	DEPARTMENTS_ DEPARTMENTID
100	Steven	King	SKING	06/17/2003	24000	90
101	Neena	Kochhar	NKOCHHAR	09/21/2005	17000	90
102	Lex	De Haan	LDEHAAN	01/13/2001	17000	90
103	Alexander	Hunold	AHUNOLD	01/03/2006	9000	60
104	Bruce	Ernst	BERNST	05/21/2007	6000	60
105	David	Austin	DAUSTIN	06/25/2005	4800	60
106	Valli	Pataballa	VPATABAL	02/05/2006	4800	60
107	Diana	Lorentz	DLORENTZ	02/07/2007	4200	60
108	Nancy	Greenberg	NGREENBE	08/17/2002	12008	100
109	Daniel	Faviet	DFAVIET	08/16/2002	9000	100
110	John	Chen	JCHEN	09/28/2005	8200	100
111	Ismael	Sciarra	ISCIARRA	09/30/2005	7700	100
112	Jose Manuel	Urman	JMURMAN	03/07/2006	7800	100
113	Luis	Popp	LPOPP	12/07/2007	6900	100
114	Den	Raphaely	DRAPHEAL	12/07/2002	11000	30

Figure 2.30: Excerpt from Retrieving all Data from Table `EMPLOYEES`.

If we would like to order the result, e.g., according to the employee's salaries, we have to simply add an `ORDER BY`-clause (cf. Figure 2.31):

```
SELECT * FROM Employees ORDER BY salary;
```

Figure 2.31: SQL Statement for Retrieving all Data from Table `Employees` ordered by their Salary.

The result of this query looks as follows (cf. Figure 2.32):

EMPLOYEEID	FIRSTNAME	LASTNAME	EMAIL	HIREDATE	SALARY	DEPARTMENTS_ DEPARTMENTID
100	Steven	King	SKING	06/17/2003	24000	90
101	Neena	Kochhar	NKOCHHAR	09/21/2005	17000	90
102	Lex	De Haan	LDEHAAN	01/13/2001	17000	90
108	Nancy	Greenberg	NGREENBE	08/17/2002	12008	100
114	Den	Raphaely	DRAPHEAL	12/07/2002	11000	30
103	Alexander	Hunold	AHUNOLD	01/03/2006	9000	60
109	Daniel	Faviet	DFAVIET	08/16/2002	9000	100
110	John	Chen	JCHEN	09/28/2005	8200	100
112	Jose Manuel	Urman	JMURMAN	03/07/2006	7800	100
111	Ismael	Sciarra	ISCIARRA	09/30/2005	7700	100
113	Luis	Popp	LPOPP	12/07/2007	6900	100
104	Bruce	Ernst	BERNST	05/21/2007	6000	60
105	David	Austin	DAUSTIN	06/25/2005	4800	60
106	Valli	Pataballa	VPATABAL	02/05/2006	4800	60
107	Diana	Lorentz	DLORENTZ	02/07/2007	4200	60

Figure 2.32: Excerpt from Retrieving all Data from Table `Employees` ordered by their Salary.

(2) What about querying certain columns and certain rows only – *"Retrieve name, address and salary of employees having a salary higher than 5.000, only"*

First, in order to get back the values of certain attributes only, we just have to specify them instead of the star, separated by commas. Second, to selectively retrieve certain rows only, we need to introduce a `WHERE`-clause and specify a logical condition on the corresponding attribute (salary in our case). Note that, there can be several such conditions on different attributes, separated by the keywords `AND` / `OR` (cf. Figure 2.33).

```
SELECT lastName, hireDate, salary FROM Employees
WHERE salary > 5000;
```

Figure 2.33: SQL Statement for Retrieving certain Data from Table `Employees` only.

The result of this query looks as follows (cf. Figure 2.34):

LASTNAME	HIREDATE	SALARY
King	06/17/2003	24000
Kochhar	09/21/2005	17000
De Haan	01/13/2001	17000
Hunold	01/03/2006	9000
Ernst	05/21/2007	6000
Greenberg	08/17/2002	12008
Faviet	08/16/2002	9000
Chen	09/28/2005	8200
Sciarra	09/30/2005	7700
Urman	03/07/2006	7800
Popp	12/07/2007	6900
Raphaely	12/07/2002	11000

Figure 2.34: Excerpt from Retrieving certain Data from Table `Employees` only.

If we would like to get the values of the "`hireDate`" attribute in another format, the `TO_CHAR()`-function" provided by ORACLE SQL can be used. This function takes a `DATE` value as first input, and a desired format as second input. In the following, an extended version of the above example is shown (cf. Figure 2.35), now using the `TO_CHAR()`-function for "`hiredate`" (a full list of possible formatting options is, of course, available in the Oracle documentation):

```
SELECT lastName, TO_CHAR(hireDate, 'Month, DD, YYYY') hireDate, salary
FROM Employees
WHERE salary > 1000;
```

Figure 2.35: SQL Statement for Retrieving Data from Table `Employees` using `TO_CHAR()`-Function.

The result of this query looks as follows (cf. Figure 2.36):

LASTNAME	HIREDATE	SALARY
King	June, 17, 2003	24000
Kochhar	September, 21, 2005	17000
De Haan	January, 13, 2001	17000
Hunold	January, 03, 2006	9000
Ernst	May, 21, 2007	6000
Austin	June, 25, 2005	4800
Pataballa	February, 05, 2006	4800
Lorentz	February, 07, 2007	4200
Greenberg	August, 17, 2002	12008
Faviet	August, 16, 2002	9000
Chen	September, 28, 2005	8200
Sciarra	September, 30, 2005	7700
Urman	March, 07, 2006	7800
Popp	December, 07, 2007	6900
Raphaely	December, 07, 2002	11000

Figure 2.36: Excerpt from Retrieving Data from Table `Employees` using `TO_CHAR()`-Function.

(3) Let's focus on querying several tables – "*Retrieve all employees and their departments*" Retrieving the data of two or more tables within a single `SELECT`-statement requires a quite prominent SQL-concept, i.e., a "*Join*". A join combines, in fact, the rows from two or more tables which have to be specified in the `FROM`-clause. Additionally, we need a `WHERE`-clause which plays the role of a *matchmaker* between these two tables – since we have to somehow define that the DBS should, for each employee in the `Employees` table, take its Departments_ departmentID value (which is the FK), go further on to the `Departments` table, match this value with the according PK value of departmentID and give back the departments data together with the employees data. More formally, this matchmaking is specified by a so-called "*join-condition*" as can be seen in the following example (cf. Figure 2.37):

```
SELECT lastName, salary, departmentName
FROM Employees, Departments
WHERE Employees.Departments_departmentID = Departments.departmentID;
```

Figure 2.37: SQL Statement for Joining Data from Table `Employees` and Table `Departments`.

The result of this query looks as follows (cf. Figure 2.38):

LASTNAME	SALARY	DEPARTMENTNAME
King	24000	Executive
Kochhar	17000	Executive
De Haan	17000	Executive
Hunold	9000	IT
Ernst	6000	IT
Austin	4800	IT
Pataballa	4800	IT
Lorentz	4200	IT
Greenberg	12008	Finance
Faviet	9000	Finance
Chen	8200	Finance
Sciarra	7700	Finance
Urman	7800	Finance
Popp	6900	Finance
Raphaely	11000	Purchasing

Figure 2.38: Excerpt from Retrieving certain Data from Table `Employees` and Table `Departments`.

Note that the according join-attributes are also *"qualified"* using the according table names (separated by a "dot"), thus striving for better readability and - in case the name of the FK would be the same as the name of the PK - ensuring a unique identification.

(4) Let's stick to aggregating values – "*Retrieve the number of employees and the sum of their salaries*"

To retieve the number of all employees in the Table `Employees` we can use the SQL `COUNT`-function over different `employeeIDs`. For building a sum of all values we can employ the `SUM`-function over an attribute, in our case, as we want to calculate the sum of all salaries, the attribute `salary` (cf. Figure 2.39):

```
SELECT COUNT(employeeID), SUM(salary)
FROM Employees;
```

Figure 2.39: SQL Statement for Counting the number of Employees and Building the Sum of their Salaries from Table `Employees`.

The result of this query looks as follows (cf. Figure 2.40):

COUNT(EMPLOYEEID)	SUM(SALARY)
15	149408

Figure 2.40: Excerpt from Retrieving the Number of Employees and Building the Sum of their Salaries from Table `Employees`.

(5) Finally, what about grouping certain rows – "*Retrieve the number of employees grouped by their salaries in descending order*"

To group same data together we can use the `GROUP BY`-function in an `SELECT`-statement. Combined with a SQL `COUNT`-function over different `employeeIDs` we are now able to determine the number of rows for each group. For sorting the result we can employ the `ORDER BY`-function stating which attribute should be used for ordering. Thereby, `ASC` declares ascending order whereas `DESC` declares a descending order (cf. Figure 2.41):

```
SELECT COUNT(employeeID), SUM(salary)
FROM Employees
GROUP BY salary
ORDER BY COUNT(employeeID) DESC;
```

Figure 2.41: SQL Statement for Grouping the Data by Their Salary and Ordering the Result by the Number of Employees with that Salary from Table `Employees`.

The result of this query looks as follows (cf. Figure 2.42):

COUNT(EMPLOYEEID)	SUM(SALARY)
2	9600
2	34000
2	18000
1	6900
1	7700
1	12008
1	11000
1	24000
1	6000
1	7800
1	4200
1	8200

Figure 2.42: Excerpt from Grouping the Data by Their Salary and Ordering the Result by the Number of Employees with that Salary from Table `Employees`

2.4.2 Data Querying using ORACLE's Query Builder ("QB")

As already mentioned, ORACLE Utilities in form of the so-called Query Builder ("QB") allows users with little or even no SQL knowledge to query DB tables and to save the queries for further usage. The QB can be accessed in the following way (cf. Figure 2.43):

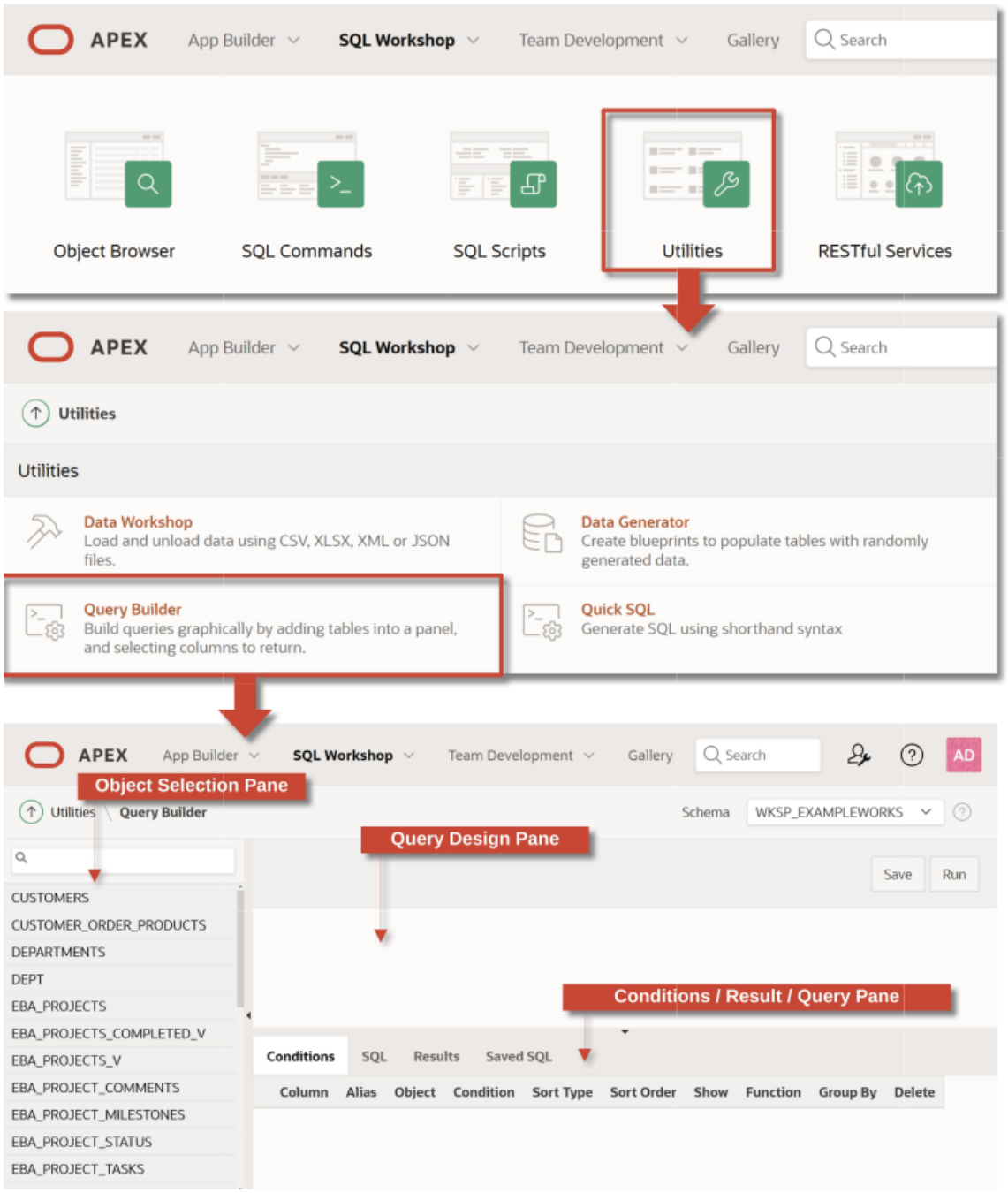


Figure 2.43: Overview of the ORACLE Query Builder (“QB”).

Overall, the steps which are necessary in order to build queries are exemplary depicted in Figure 2.44, using our "Join-example" shown in Figure 2.37 and briefly described in the following:

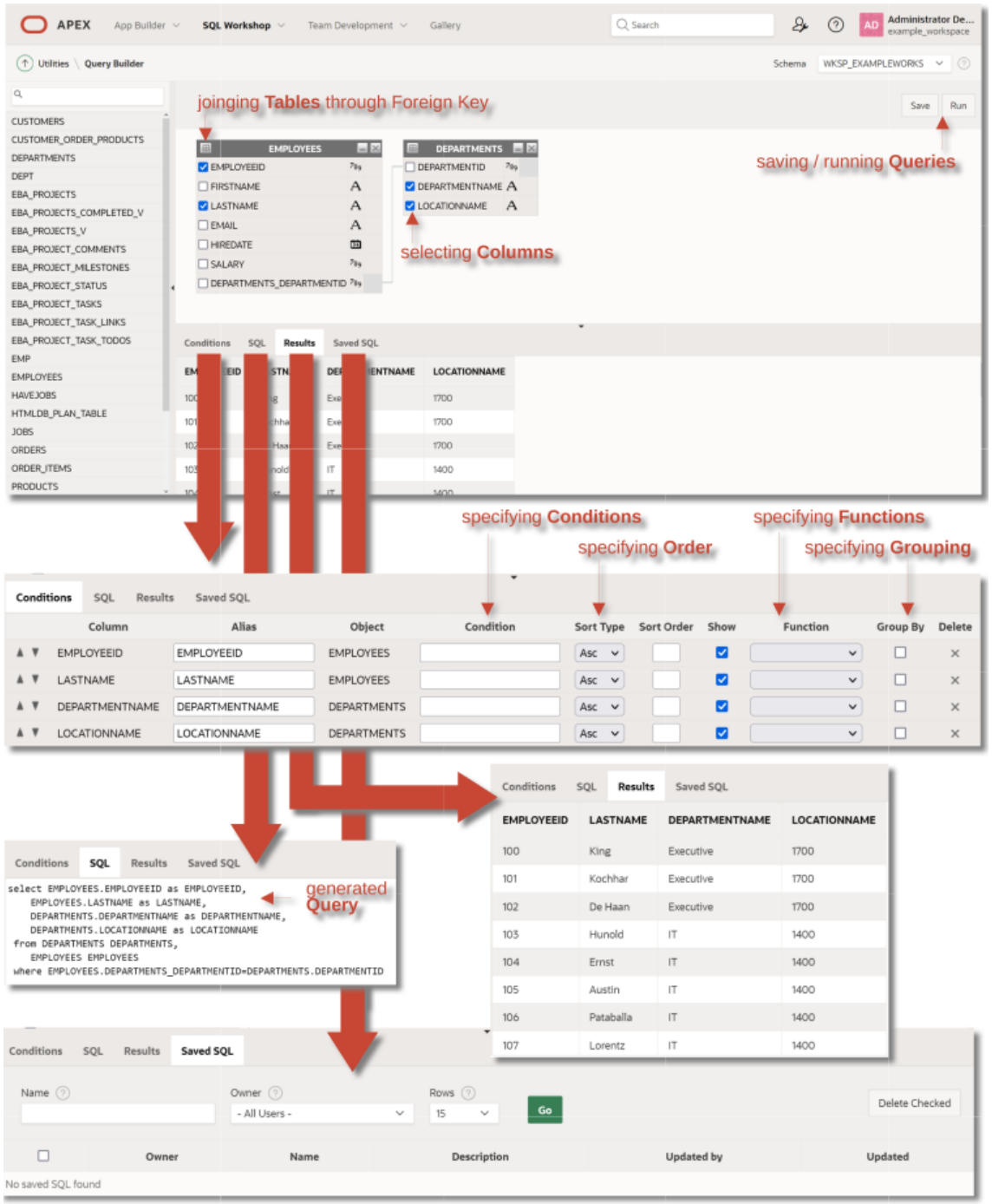


Figure 2.44: Using the Query Builder to build a Join.

- Select the *target of the query* in terms of one or more DB tables (called „objects“ in the QB) from the „Object Selection pane“ (cf. Figure 2.43). This forms the “`FROM`”-part of SQL-DQL.
- Add the selected objects to the “Design pane” and further select desired columns. This makes up the “`SELECT`”-part of SQL-DQL.
- Optional: Create query conditions, i.e., the “`WHERE`”-part of SQL-DQL.
- Execute the query and view results (click the “Run”-button).
- Optional: View the SQL-DQL code which is automatically generated (click on the “SQL-tab”).

- Optional: Save the query for further usage (click the “Save-button”).

Finally, it has to be noted that the queries defined using QB are again automatically translated by the RDBS into proper SQL-DQL statements.

2.5 Building up the DB-Layer – The Big Picture

After reading the previous sections, one is maybe completely puzzled by all those different options and alternatives which are provided by ORACLE’s tooling landscape in order to manage the DB-layer of Web applications. Therefore, in order to cut a path through this tooling jungle, Figure 2.45 provides a summarizing overview, illustrating all options and according tooling which is available in order to finally manage the DB-layer for a Web application.

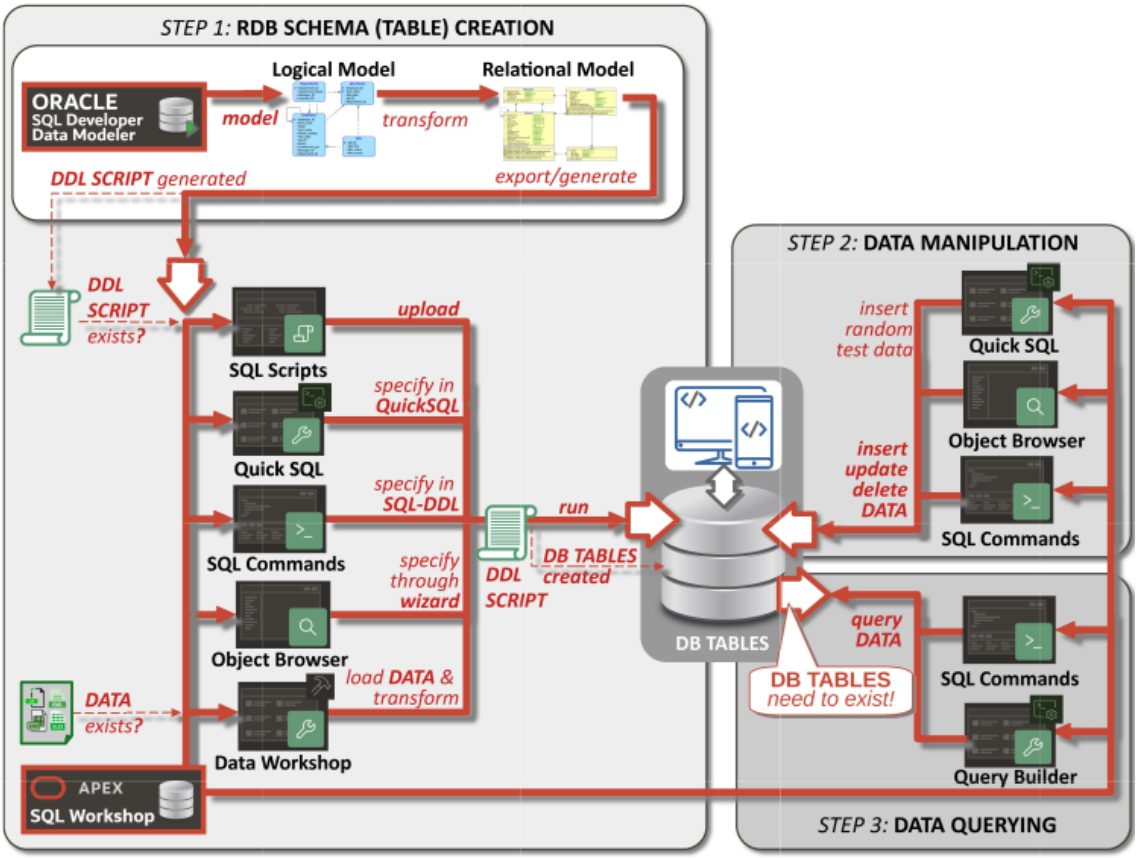


Figure 2.45: Overview on the Options to Manage thee DB-Layer.

Overall, at the left hand side, two different tooling environments are shown, the ODM and the APEX SQL Workshop. The ultimate goal of creating the RDB Schema in terms of tables (Step 1) can be reached through different paths. All of them, however, leading to the generation of SQL scripts which are finally executed to generate empty tables. From there it is possible to insert data into the tables, to update, to delete (Step 2) or to query data (Step 3) using the tooling of the SQL Workshop workbench as described in this chapter.

The Web application could be generated on empty tables. This process, however, is discussed in detail in the following Chapter 3.

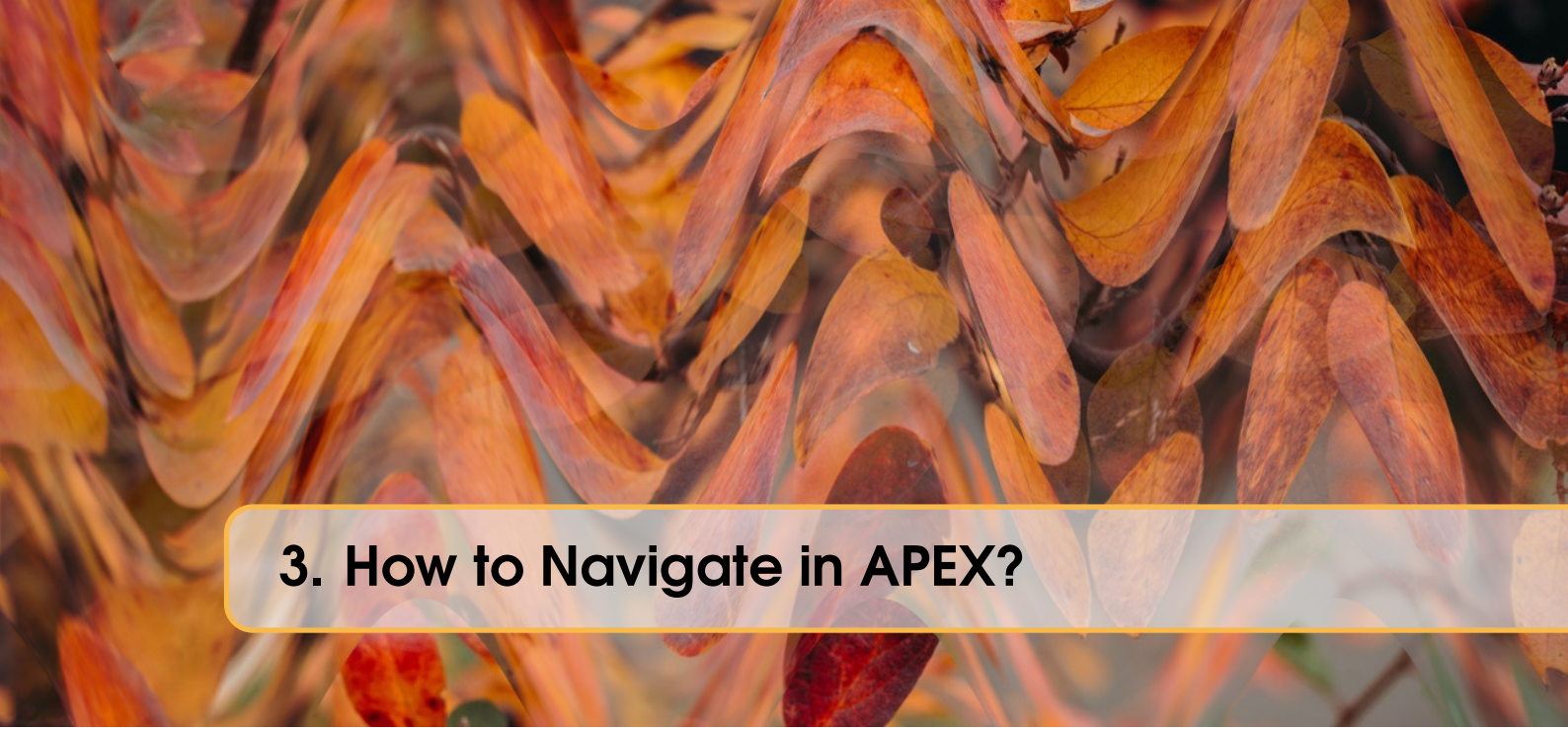
2.6 Questions

1. What is the difference between a logical model and a relational model? What are the elements of the two models?

2. What is the normalization of an RDB-Schema?
3. Describe the options for creating tables using the Oracle toolset.

2.7 Answers

1. The logical model is more abstract. The elements of the logical model are entities, attributes, and relationships. The relational model represents a more concrete and therefore DBS-specific level. The elements of the relational model are tables, table columns (data fields) and relationships.
2. The normalization of a RDB-Schema is a systematic process with three steps called "first normal form", "second normal form" and "third normal form", leading to improved RDB-Schemas.
3. There are at least five ways to create the tables: automatic table generation with Oracle Data Modeler, automatic table generation with Quick SQL in Oracle APEX, manual table creation with SQL-DDL in Oracle APEX SQL Commands, manual table creation with Oracle APEX Object Browser, and automatic table creation with Oracle APEX Data Workshop.



3. How to Navigate in APEX?

E. KAPSAMMER, W. RETSCHITZEGGER, W. SCHWINGER

What is Oracle APEX? Oracle Application Express (Oracle APEX) is a Web browser-based low-code development environment for Oracle DB-driven Web applications. "Low-code" means that only few programming skills are required for building-up a full-fledged Web application – for a first, simple Web application prototype, programming skills are not even necessary at all.

For Which Purposes can APEX be Used? Overall, APEX allows not only to build-up the DB-layer of Web applications using APEX "*SQL Workshop*", as has been discussed in Chapter 2, but also to build-up desktop or mobile Web applications using APEX "*App Builder*". APEX is employed by large and small customers alike, across a broad number of application domains, coping with a wide spectrum of business needs. These business needs may range from a simple transformation of a local spreadsheet into a Web-based one, to the realization of a full-fledged Web application (cf. the running example introduced in Chapter 2) allowing to store, retrieve, visualize and further process every single sale in every of Walmart's 10.500 stores together with responsible departments, employees, and information about their job.

What is the Rationale Behind this Chapter? The rationale behind this chapter is to first discuss the overall Web application development process followed by Oracle APEXs "App Builder". This process forms the basis for the navigation possibilities between the different App Builder tools called "development components" in order to build up a Web application. Second, a rather high-level overview of these different development components is given, whereas details will be discussed in the further chapters.

3.1 Web Application Development Process followed by APEX

Before going into details regarding the different development components of APEXs' "App Builder", it is useful to take a look into the Web application development process followed by APEX.

APEX Follows a Multi-Step and Cyclic Development Process. The overall development process supported by APEX, which is illustrated in Figure 3.1, is organized in a *multi-step, partly cyclic manner* allowing for *incremental development*, i.e., a stepwise refinement of the Web application and eventually the DB-layer. Each of the development steps is supported by appropriate Wizards of the development components. This means that from menu options at the top-level, one is guided deeper into the functionality of *App Builder* in order to specify each detail of a Web application, stepping back and forth in order to perform incremental refinements if needed.

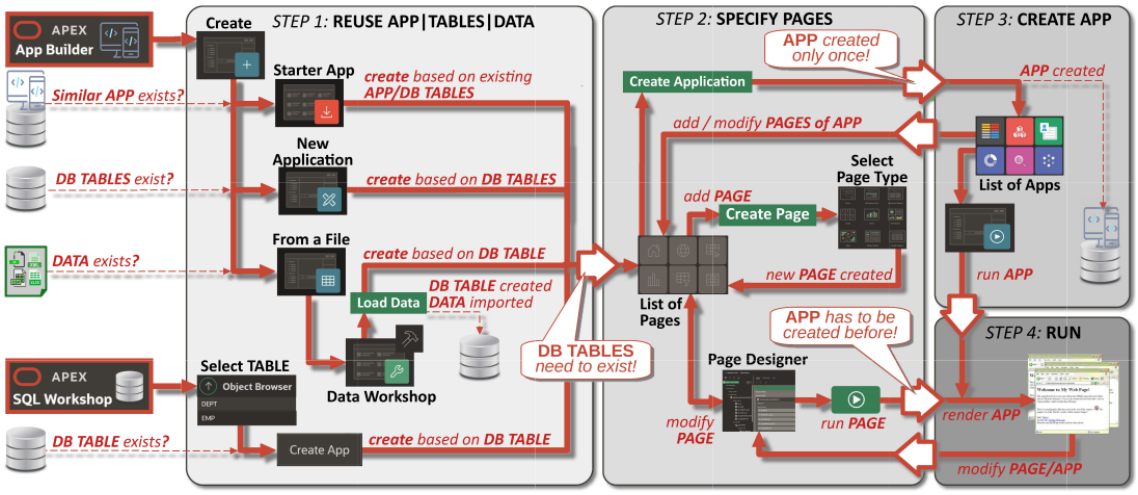


Figure 3.1: Overall Development Process in APEX.

APEX Follows a "DB-Layer First" Development Process. It has to be emphasized that, as APEX focuses on the development of DB-driven Web applications, it is necessary to always start by developing the DB-layer first (cf. Figure 3.1 "STEP 1: REUSE APP | TABLES | DATA") . This can be done either as described in Chapter 2 or directly within the App Builder, if, e.g., some external data files already exist, which can be reused for creating the DB tables (cf. Figure 3.1 "From a File"). In case that the DB tables already exist, the development process can either be started (i) from within the App Builder, thereby eventually reusing (and adapting) an existing application together with its DB tables (cf. Figure 3.1 "Starter App") or by creating a new application from scratch, again based on existing DB tables (cf. Figure 3.1 "New Application") or (ii) from within SQL Workshop, by simply selecting a DB table within the Object Browser (cf. Figure 3.1 "Select TABLE") and starting the application development process on basis of this table.

APEX Follows a "Page-Driven" Development Process. As soon as the DB tables for storing the data of the Web application exist, they can be used for creating a new Web application, following a *page-driven process* (cf. Figure 3.1 "STEP 2: SPECIFY PAGES"). This means that one or more Web pages can be defined, making up the HTML-pages of the final Web application. Each of these pages can be based on one or more DB tables, allowing to visualize their data in terms of different *page types* comprising, e.g., interactive reports, forms, lists, charts or calendars, but also to interactively "play" with the data (e.g., zoom into details), and even to manipulate them, and store the changes back into the DB. These *pages can be linked together* using navigation menus, tabs, buttons, or hypertext links.

As soon as one or more pages have been created, it is necessary to create the overall application (cf. Figure 3.1 "STEP 3: CREATE APP"), thereby further specifying some overall properties of the whole Web application (e.g., appearance of the App) or just using the default settings. Finally, the App and/or each of the specified pages has to be rendered in order to generate the final HTML-pages (Figure 3.1 "STEP: 4 RUN"), which can further on be interactively tested, occasionally stepping back to further refine the Web application.

3.2 The Start Menu of APEX

When you sign in to Oracle APEX, the so-called *"Workspace Homepage"* appears. This Workspace Homepage provides access to four different development components (cf. Figure 3.2):

- *"App Builder"* is mainly used for building up the Web pages and will be the focus of this Chapter.
- *"SQL Workshop"* is used for building up the DB-layer and has been already described in

Chapter 2.

- *"Team Development"* is used, e.g., to interact with other developers via a ticketing system to set milestones, etc.
- *"Gallery"* is used to simply install pre-built demo applications, which can then be further modified and adapted towards own needs, again using the "App Builder".

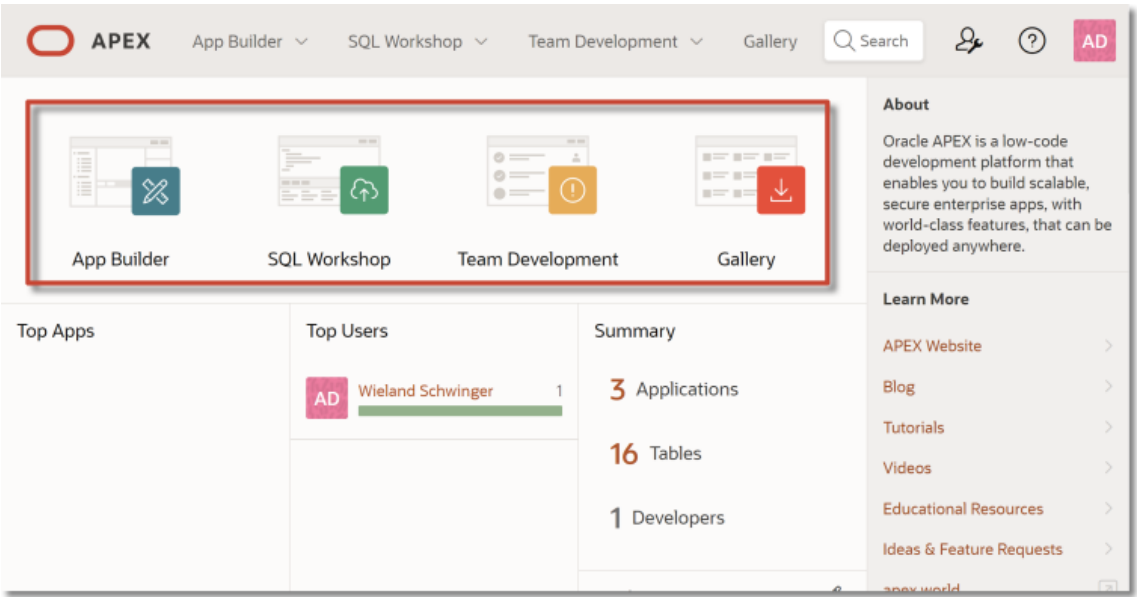


Figure 3.2: Oracle APEX Workspace Homepage.

At the bottom of the Workspace Homepage (cf. Figure 3.2), the regions "*Top Apps*", "*Top Users*", and "*Summary*" offer real time information about development activities in the current DB workspace, whereas at the right-hand side further APEX learning resources are referenced.

3.3 Overview of the App Builder – Create and Manage your Apps

When navigating to the App Builder Homepage, one has the following options (cf. Figure 3.3):

- *Create* a new Web application.
- *Import* previously exported Web applications.
- View the "Dashboard" providing statistics on already developed Web applications.
- *Access "Workspace Utilities"* providing a wide range of services like remote data access or backup functionality.

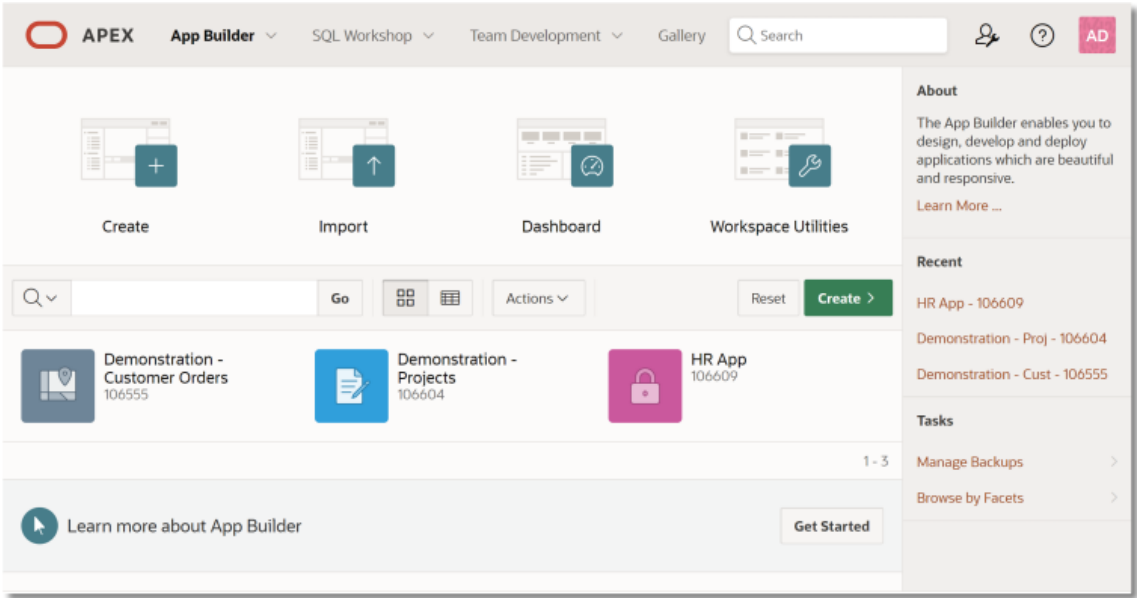


Figure 3.3: Overview of the App Builder.

In the bottom half of the window (cf. Figure 3.3), already existing Web applications are listed which can be selected for further editing or rendering (cf. forthcoming sections).

3.4 Create Application – Three Use Cases

When choosing *"Create"* in the App Builder, the "*Create Application Wizard*" brings up three further options (cf. Figure 3.4), supporting in fact real-world use cases which take into account the possible reuse potential regarding the data and the application itself:

- *"New Application"* should be selected if the DB-layer (or part of it) already exists or if the DB-layer should be developed "on-the-fly" during application development.
- *"From a File"* is the option of choice, if the DB-layer should be automatically generated from an external file, e.g., an EXCEL-file.
- *"Starter App"* finally allows to reuse (and probably further modify) an already existing application together with its DB-layer from the "Gallery" of applications already mentioned above.

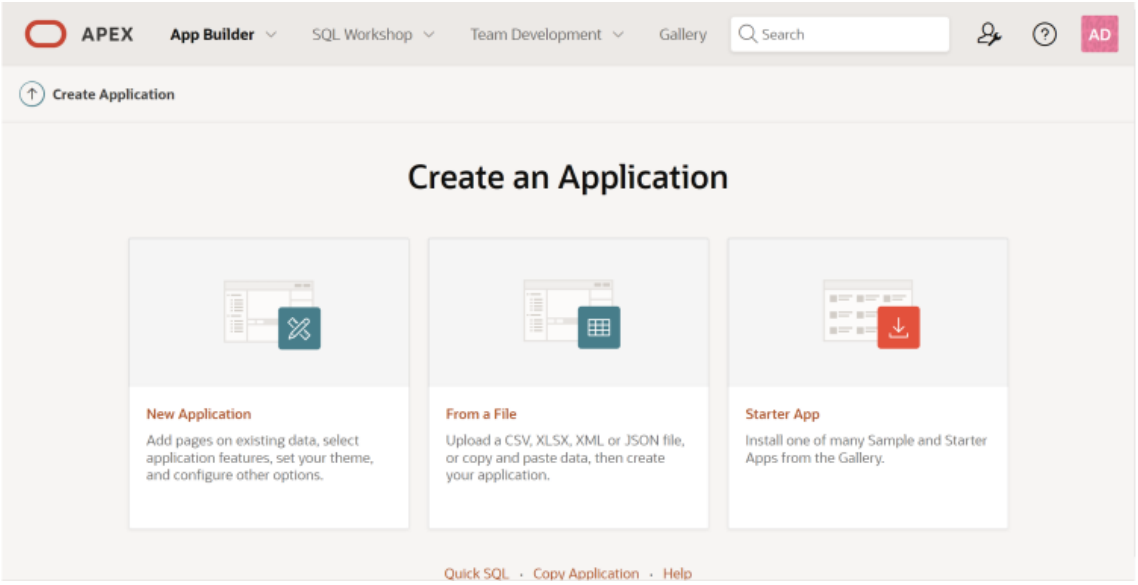


Figure 3.4: Create Application Wizard: Three Use Cases when Creating an Application.

In the following, the focus will be on the first option *"New Application"*, the second option *"From a File"* will be described in more detail in Chapter 4, the third option *"Starter App"* will not be further dealt with.

3.5 Create Application – Properties, Pages, Features and Settings

When creating a new Web application, some *common properties* have to be defined, the *pages* have to be created, and some additional quite useful *features* and *settings* can be chosen. In the following, all these definitions are discussed in more detail, wheres Figure 3.5) shows the upper part of the GUI and Figure 3.6) shows its lower part.

Common Properties – Icon, Name and Appearance of the Application. Regarding the common properties, as illustrated in Figure 3.5, (1) an arbitrary *icon* can be chosen or uploaded, used as the Favicon and the icon for PWA, Apple touch or App Builder, (2) a *name* has to be entered and (3) the *appearance* of the Web application can be chosen from some pre-defined options, determining its "look and feel" in terms of a theme style (e.g., color, font size, etc.) and the *kind of navigation menu* (e.g., side menu or top menu), whereby also a default appearance is available.

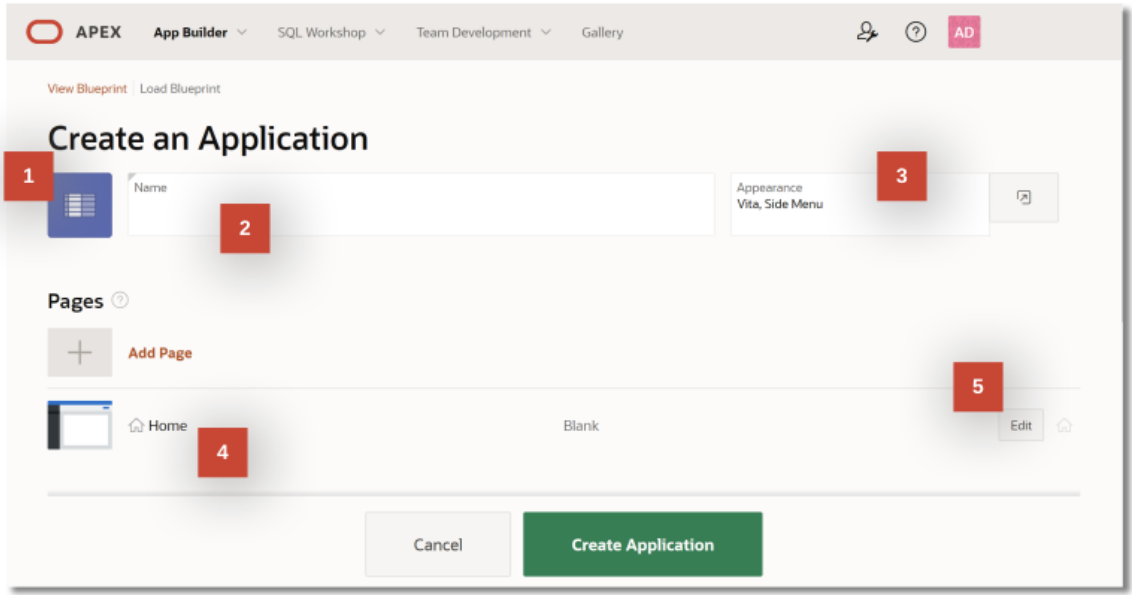


Figure 3.5: Properties and Pages.

Page Creation for the Application. The main task, however, is to incrementally build-up the Web application by successively *adding new pages* using the *"Create Page Wizard"* (started by *"Add Page"*, cf. (4) in Figure 3.5. Once a page is created, one can, at any time during application development, *edit* the *composition* of the page, *alter* the page order, and *delete* them (cf. (5) in Figure 3.5). At the bottom part of Figure 3.5, an already created page "Home" is shown. Further details about the "*Create Page Wizard*" are given in Section 3.6.

Features and Settings of the Application. Finally, the application can be enhanced by certain quite useful *features* (cf. (1) in Figure 3.6) such as an "About Page", a page for automatic "Access Control" or a page for "User Feedback" as well as by some common *settings*, e.g., the name of the schema holding all DB-tables and thus forming the DB-layer for the application or the used language (cf. (2) in Figure 3.6).

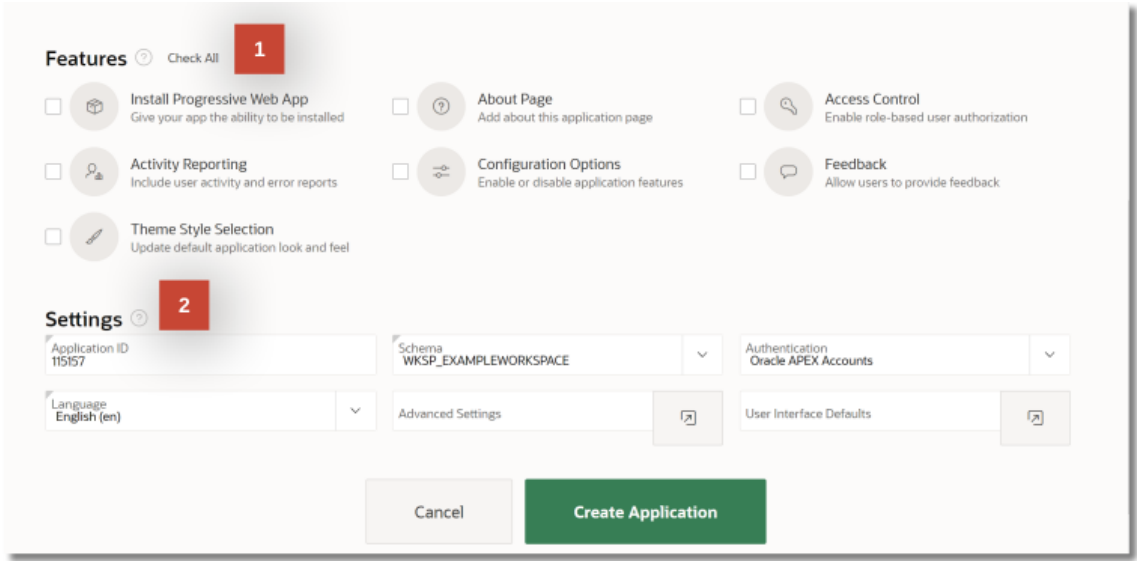


Figure 3.6: Features and Settings.

After clicking the "*Create Application*" button the "*App Builder Homepage*" appears showing

the newly created application with its pages (cf. Figure 3.7).

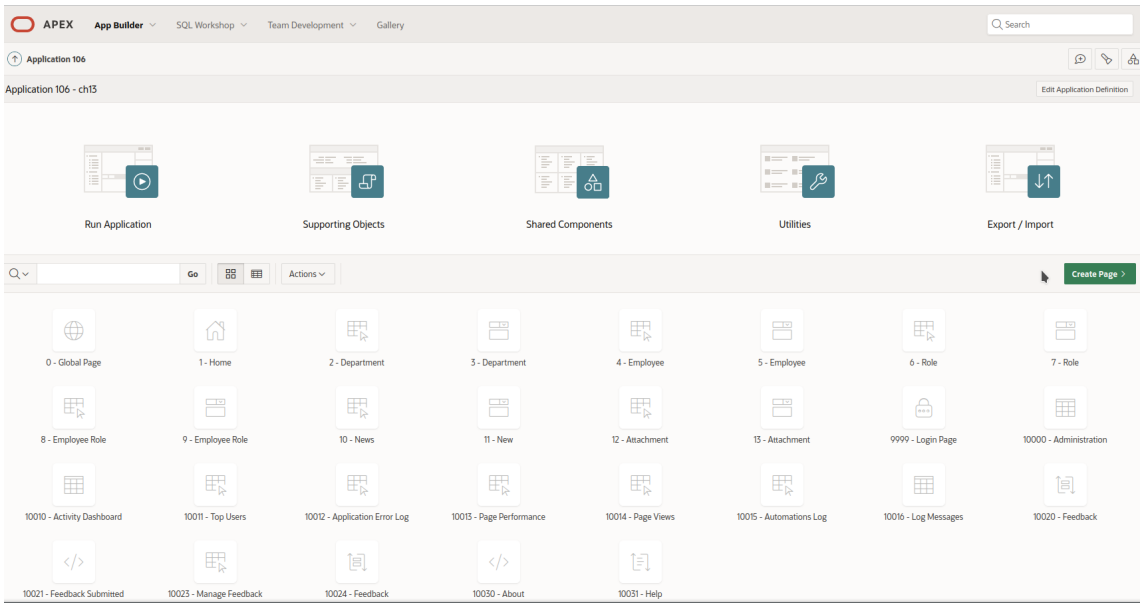


Figure 3.7: Application Homepage - Developer's view.

3.6 Specify Pages

What makes up a Page? As already mentioned, a page is the basic building block of a Web application, i.e., every application consists of at least one or multiple pages. Each page can have buttons and fields (called *"items"*) which are grouped into containers called *"regions"*. Pages can also include *application logic*, e.g., perform *calculations* (called *"computations"*) , and perform *validations* such as correctness checks during data manipulation.

Choose from 18 Different Page Types. A page can display data of DB tables in various different forms determined by so-called "page types", comprising, e.g., interactive reports , lists , calendars or charts . A page type has to be selected, whenever a new page is created, either by using the "Add Page" button from within the "Create Application" dialogue (cf. Figure 3.5) or by using the "Create Page" button from within the developer's Application Homepage (cf. 3.8). Overall, 18 different page types are provided as illustrated in Figure 3.8. A certain page type determines *contents*, *composition*, and *layout* of a page as can be recognized by taking a look at the page type icons in Figure 3.8. Details about each of these page types are given in the forthcoming chapters.

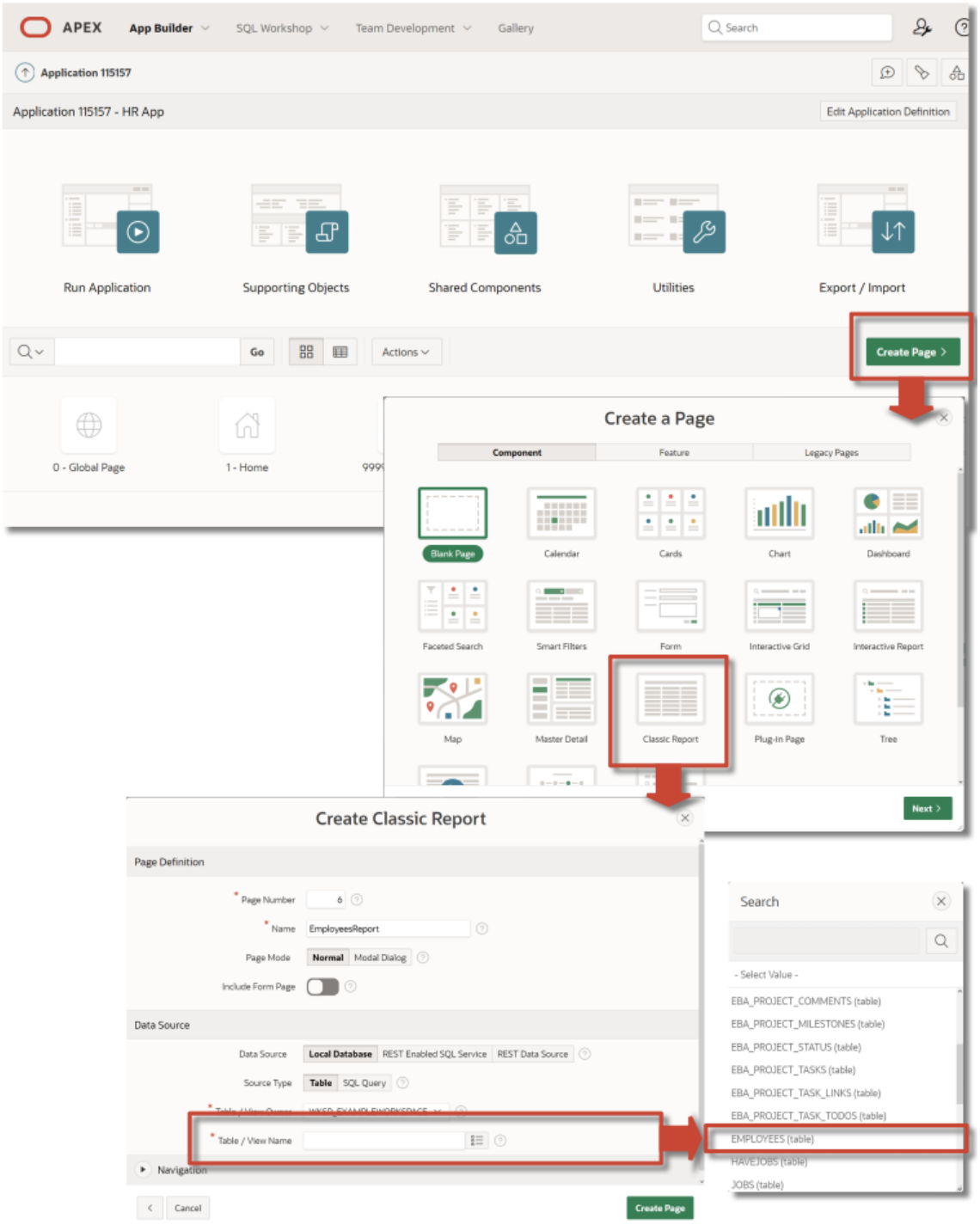


Figure 3.8: Creating a Page.

Page Specification Dialogue – Table-to-Page Mapping. When deciding for a certain page type, the concrete dialogue for page specification which is presented by the "Page Creation Wizard" is naturally dependent on the selected page type. Nevertheless, since nearly all of the available page types are based on DB-tables, the main task is commonly to map appropriate DB-tables to the page, providing the source for data visualization and the target for data manipulation. Figure 3.8 shows an example dialogue for the quite common page type "*Classic Report*". It can be seen that, besides some basic information like "Name", also the mapping to the DB-layer has to be specified by selecting the "Data Source" in terms of the DB table which should be the source for this report

(in Figure 3.8 we select the "Employees" table of our running example). It has to be noted that in order to display the data from the underlying DB table, APEX inherently utilizes SQL-DQL in the background as soon as an HTML-page is rendered.

As soon as a new page is added, the "*Page Designer*" component of APEX (cf. Section 3.7) is automatically opened for this page, allowing to view and edit the page specification. At the same time, the page can also be immediately and automatically rendered into a HTML-page, e.g., for testing and incremental development purposes, using the "Run" button (cf. Section 3.8).

3.7 Maintain and Modify a Page – Page Designer

Already created pages may, of course, also be further maintained and enhanced using the so-called "*Page Designer*". The Page Designer is a full featured *Integrated Development Environment (IDE)* that includes a toolbar and multiple panes (cf. Figure 3.9). The functionality provided by Page Designer is huge, allowing, among others, to fully modify the composition/layout of a page and its components (cf. (1), (2) and (5)), including items, buttons and regions (cf. (3)) but also to modify the rendering of the page (cf. (or application logic in terms of computations and validations (cf. (4)).

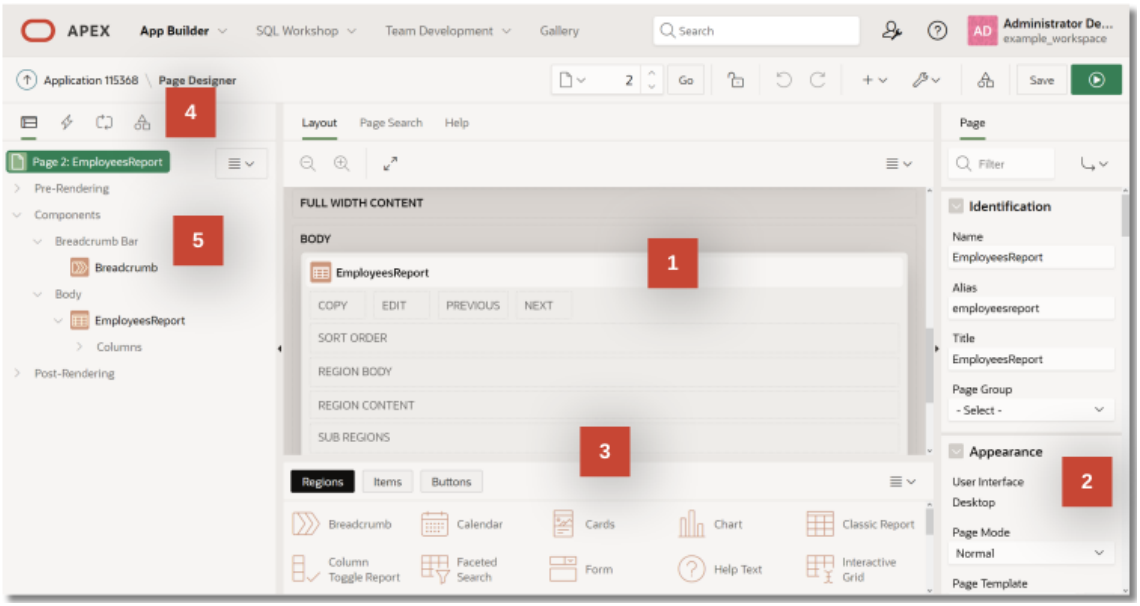


Figure 3.9: Page Designer.

Details about the functionality provided by Page Designer can be found in the forthcoming chapters.

3.8 Run Application or Page

To view a rendered version of an application or an individual page, it has to be submitted to the so-called *Oracle APEX engine* by clicking the "*Run*" button at the *application home page* (cf. Figure 3.10 and Figure 3.11). The APEX engine dynamically renders and processes pages into viewable *HTML pages* based on data about the application which is stored in "internal" DB tables. There are two different options available on the *Application Homepage*, depending on if you would like to run the entire application, i.e, all pages, or just a single page for, e.g., testing purposes.

- **Run the Entire Application**, i.e., all pages: This option is available on the Application Homepage as can be seen in Figure 3.10.

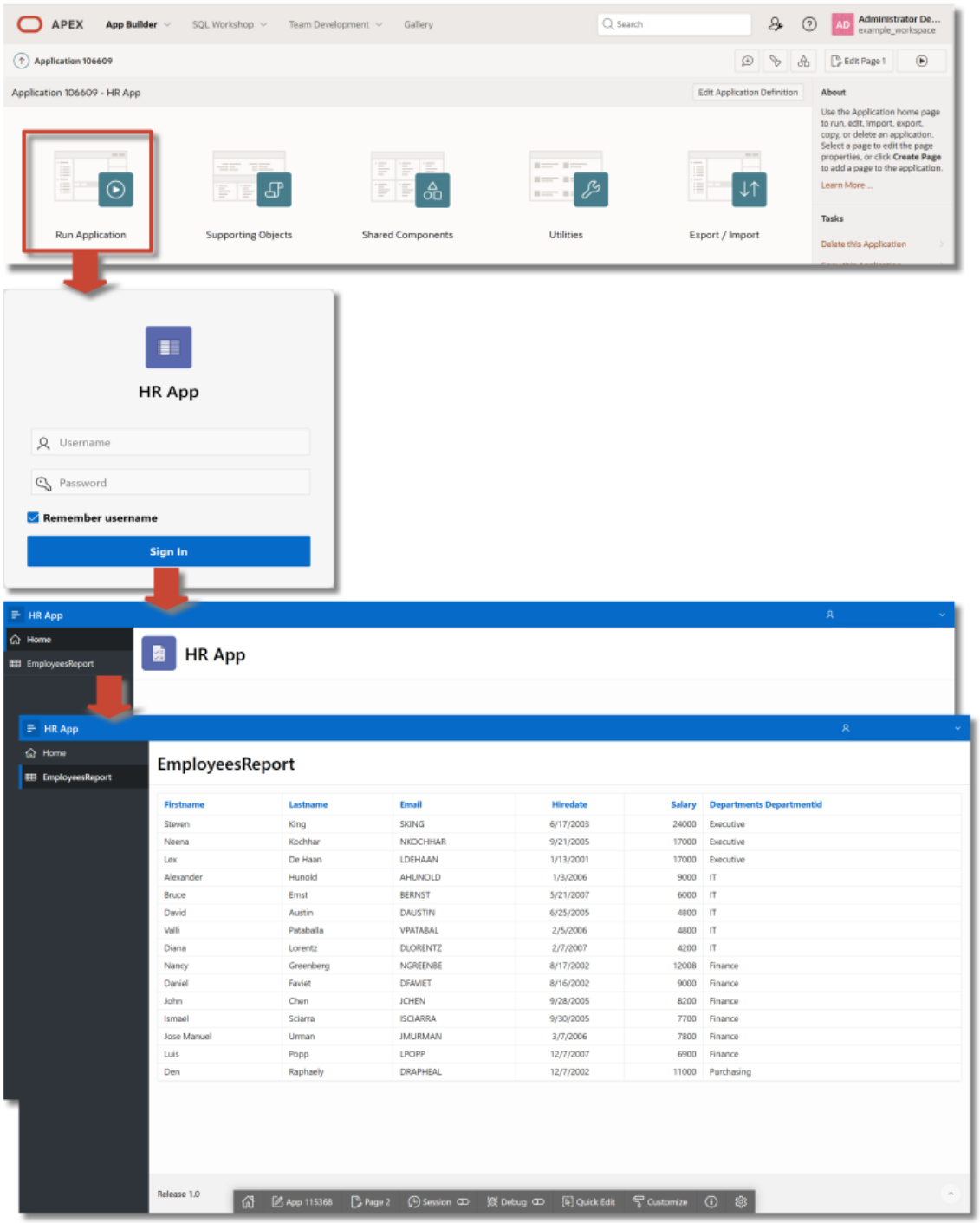


Figure 3.10: Running Entire Application.

- **Run Individual Pages**. As you create new pages, it is also possible to run each page individually to get an immediate impression of the look and feel of the actual page. The "Run Page" button resembles a small, black play icon and displays at the right-hand side of each of the pages of your application as can be seen in Figure 3.11. It has to be noted, however, that even if one just runs an individual page, also all other pages are available.

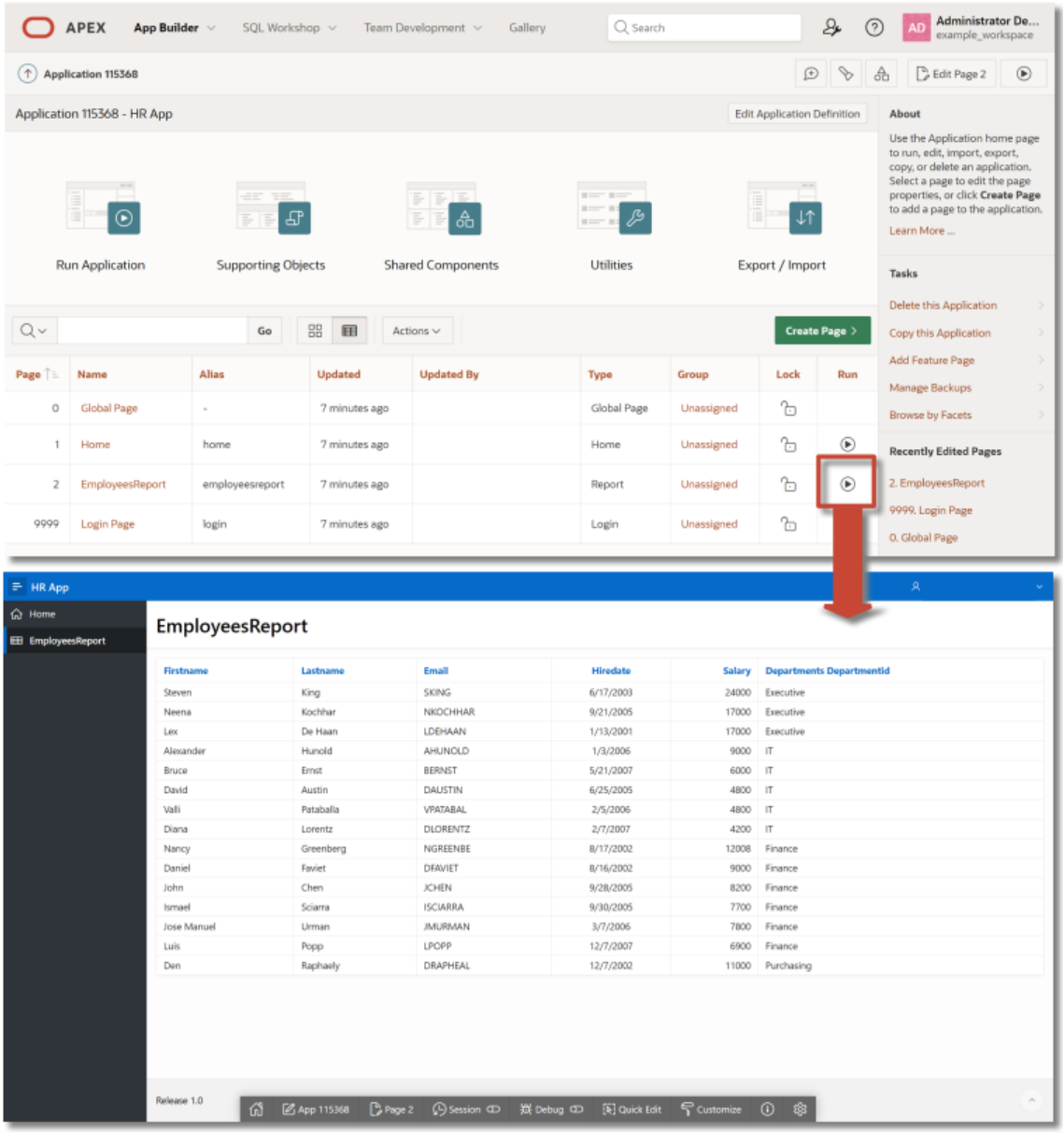


Figure 3.11: Running Individual Pages.

Note that the other options which are provided at the Application Homepage besides “Run Application” (e.g., Supporting Objects) are dealt with in forthcoming chapters.

3.9 Questions

1. What options does the application creation wizard provide and what are they used for?
2. What features does the "New Application" wizard provide?
3. What kind of pages can APEX generate?

3.10 Answers

1. There are three options: a) "New Application" allows the developer to add pages for existing data, select application features, set the theme and configure it, b) "From a File" allows the developer to upload files in formats such as CSV, XLSX, XML and JSON or copy/paste data and initiate the creation of the application and c) "Starter App" allows the developer to install

one or many Sample and Starter Apps from Gallery.
2. There are several useful features such as automatic creation of an "About Page", providing application administrator a page to grant privileges to users in the "Access Control" feature, automatic creation of a "User Feedback" page, activity reporting, theme style selection, configuration options, and the choice of installing a progressive web application.
3. APEX can generate a wide range of pages: form, report (interactive grid, interactive report, classic report, master detail), blank page, calendar, cards, chart, dashboard, faceted search, smart filter, map, search page, plug-in page, tree, data loading, wizard and unified task list.



4. How to exchange data in APEX?

E. KAPSAMMER, W. RETSCHITZEGGER, W. SCHWINGER

The major task when building up the DB-layer for a Web application is the creation of the DB-tables and the insertion of appropriate data (see Chapter 2). This can, however, not only be done manually if the data does not yet exist. In the case there is already existing data available within external files it can simply be reused and *imported* into the Oracle DB. At the same time, it could be desirable to *export* already existing data into a file from time to time, so that it can be reused in a second step by some other, external application. And finally, in case that some external clients (e.g., other Web/mobile/legacy applications or cloud-based services) should be provided with *online access* to the data within our DB, instead of just exchanging data files from time to time, there is also the possibility of providing indirect online access to our DB data.

These three use cases for data exchange are illustrated in Figure 4.1, providing a summary of the different options Oracle APEX provides for importing, exporting and indirectly accessing DB data. This chapter is intended to give an overview about these different options.

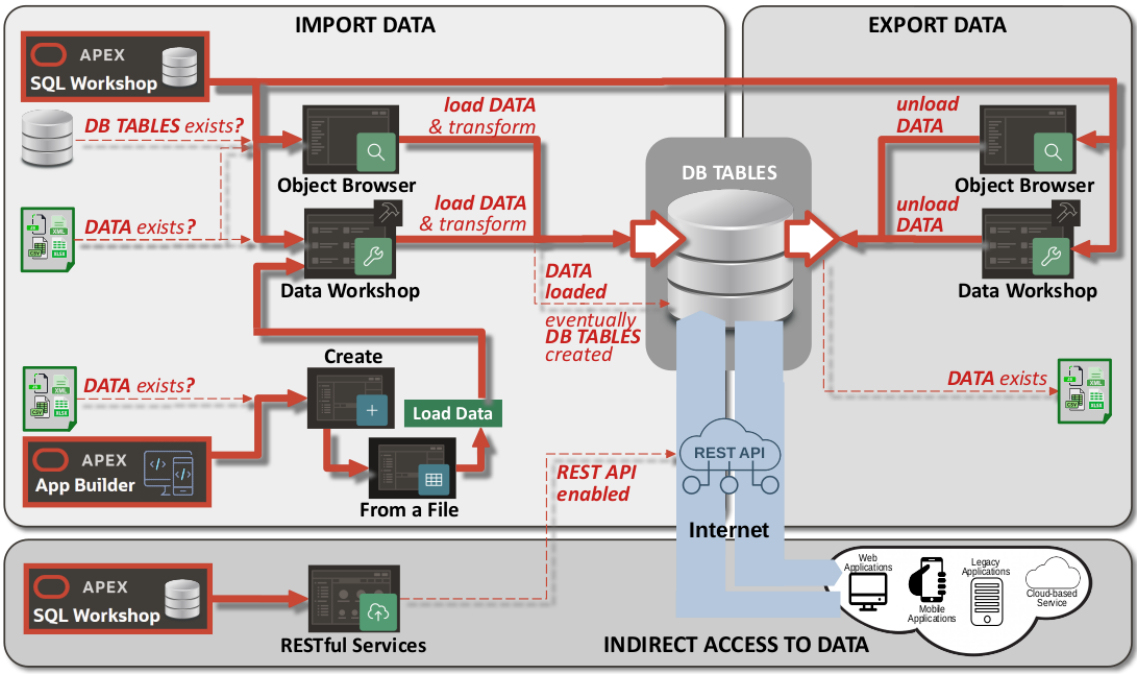


Figure 4.1: Data Exchange Options in APEX.

4.1 Importing and Exporting Data Using "Data Workshop"

Application Scenario for Data Workshop. Data Workshop is an easy-to-use tool, providing a "Wizard" in order to simplify the task of importing/exporting data from/to external files. Data Workshop is especially suited for application scenarios dealing with data of a moderate size (fewer than 10 tables) having simple standard data types only (e.g., no multi-valued fields or nested structures). It has to be noted that for reusing and importing *huge and complex data sets*, Oracle provides other appropriate tooling, e.g., in terms of the the so-called "*SQL*Loader Utility*". In the following, first of all, Section 4.1.1 and Section 4.1.2 deal with the import of data, while Section 4.1.3 describes the export of data.

4.1.1 Importing Data

File Formats. Overall, Data Workshop allows to load data from external files adhering to the following formats:

1. Any standard *delimited format* being *tab-delimited* or *comma-delimited* ("*CSV*" – *Comma Separated Values*).
2. *XLSX files* (i.e., *Excel workbooks*). Note that, if the uploaded XLSX file contains multiple worksheets, the first sheet is picked by default. To load another sheet, it can be picked from the "Select Sheet" select list.
3. *JSON files* (*Java Script Object Notation*). Note that only one nesting level is supported.
4. *XML files* (*eXtended Markup Language*). Note that similar to JSON-files, only one nesting level is supported.

Accessing Data Workshop for Imports. Data Workshop can be accessed in two alternative ways:

1. Start Data Workshop out of SQL Workshop as illustrated in Figure 4.2.

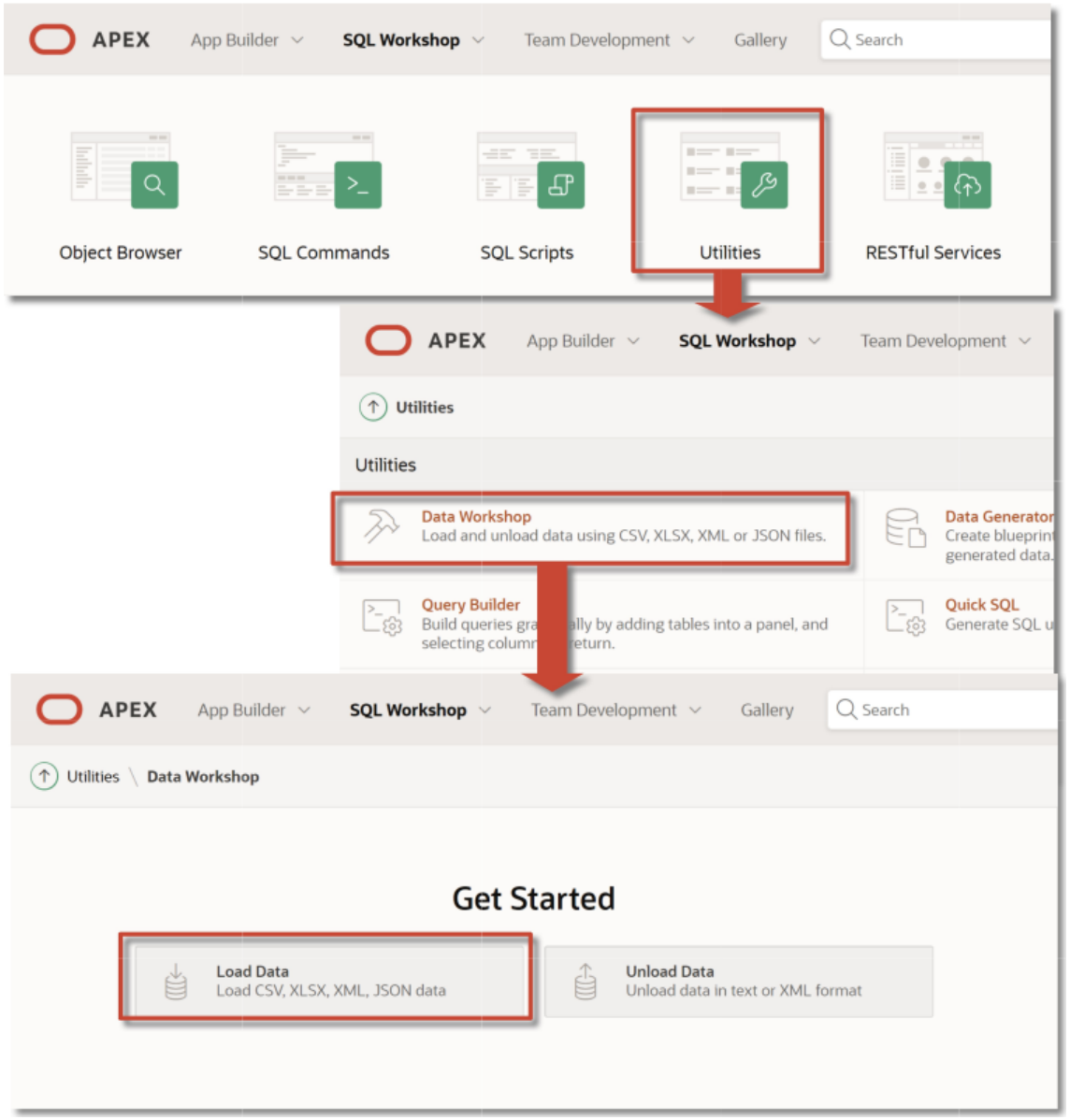


Figure 4.2: Using Data Workshop to Load Data - Access through SQL Workshop.

2. Alternatively, Data Workshop can also be accessed from within the APEX App Builder when creating an application as already mentioned in Chapter 3 (cf. Figure 4.3), using the "From a File" option (cf. Figure 4.1).

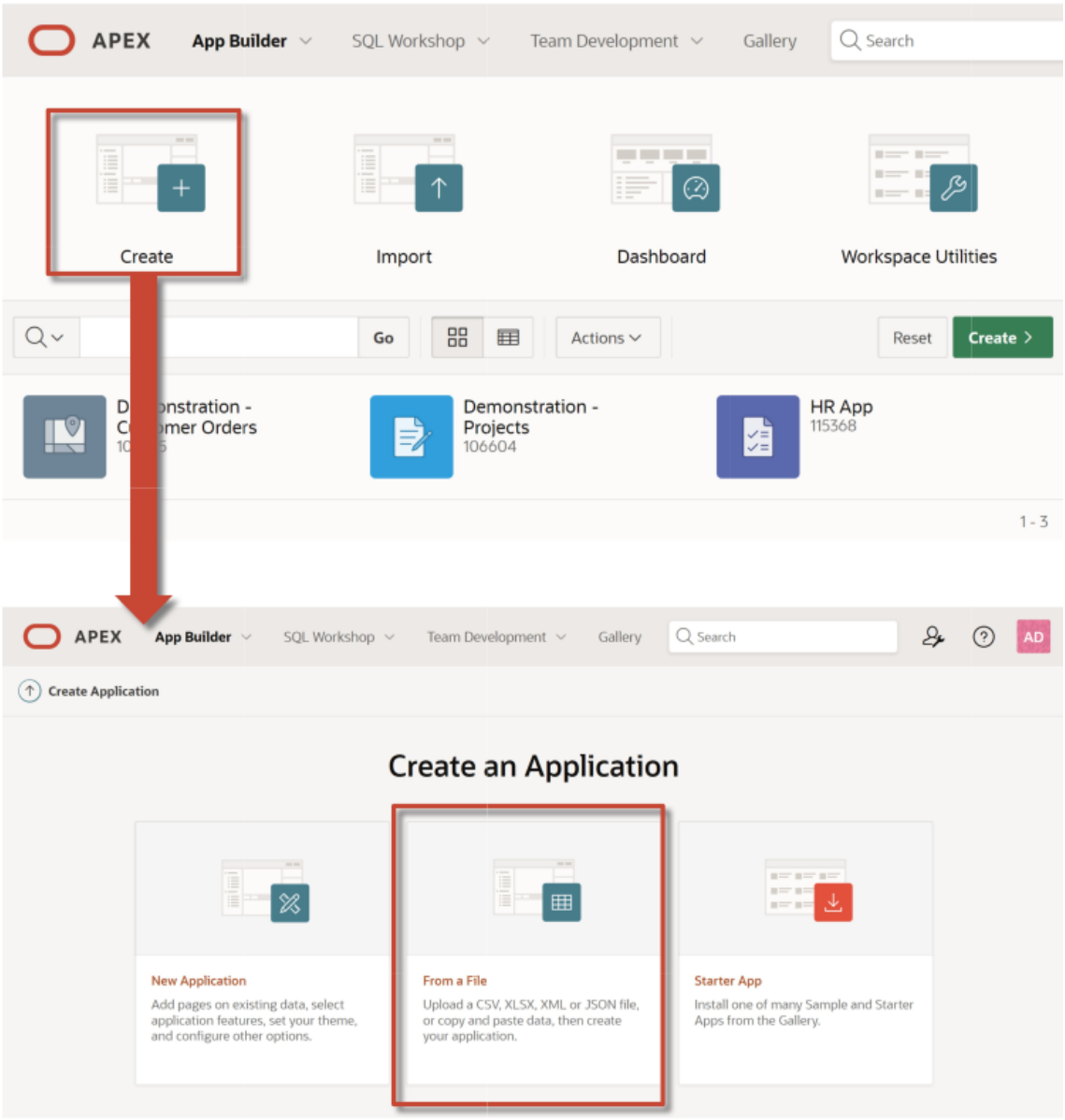


Figure 4.3: Using Data Workshop to Load Data - Access through App Builder.

4.1.2 Importing Steps

In the following, the steps necessary to import data are briefly discussed, together with some important issues useful in order to understand how to use the "Load Wizard":

Provision of the Data Source. The first, quite natural step is to provide the data source which can be done either by uploading (drag/drop or choose file) delimited files, XLSX, JSON or XML files or by a simple copy/paste of delimited data, only (cf. Figure 4.4). Note that this dialog appears when either clicking the *"Load Data-button"* (cf. Figure 4.2) or via the *"From a File-button"* (cf. Figure 4.12).

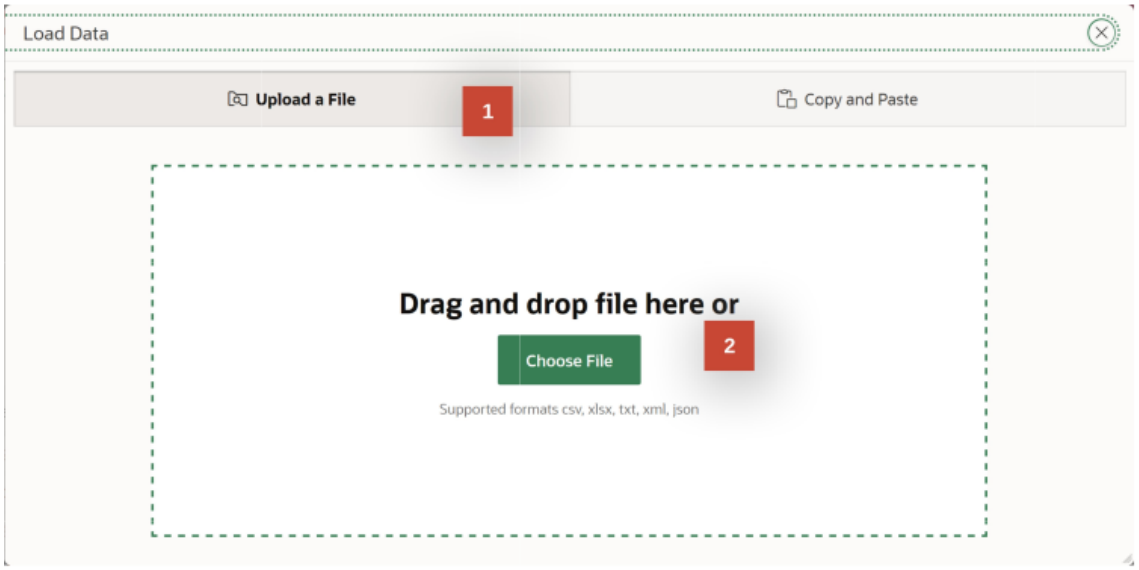


Figure 4.4: Provision of Data Source.

Configuration of the Data-to-Table Mapping. The central step of the Load Wizard is the configuration of the Data-to-Table Mapping, which is visualized in Figure 4.5 and 4.6, where the most important configuration options are annotated by numbers (1)-(4) and (1)-(8), being described in the following.

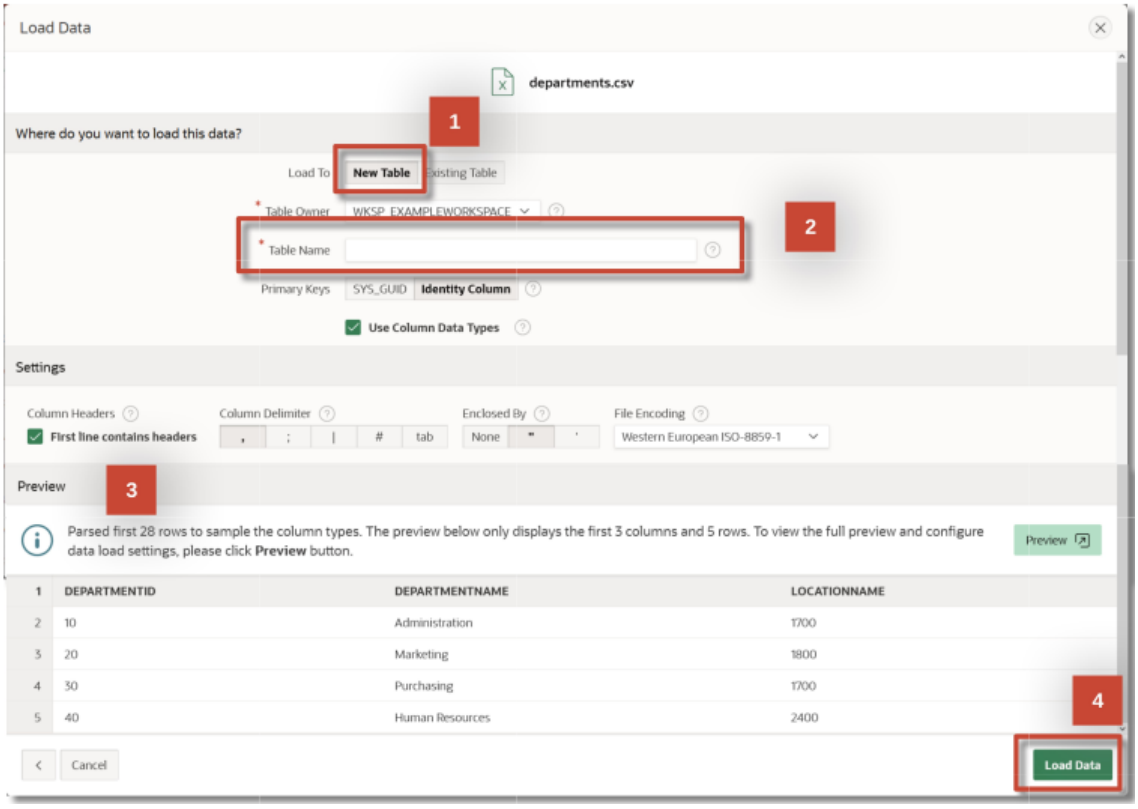


Figure 4.5: Loading Data into New Table.

1. **Load Data into Existing or New Table.** First of all, it has to be decided if the data should be stored in an already existing DB table or if a new DB table should be used (cf. Figure 4.5

(1) and Figure 4.6 (1) respectively).

2. **Specify Table Owner and Name.** For both cases, table owner and table name have to be specified (cf. Figure 4.5 (2) and Figure 4.6 (2) respectively).
3. **Settings for Delimited Files.** The *setting section* (cf. Figure 4.5 (3)) allows for three different important configurations specific to delimited ("CSV") files. The configuration option "*Column Headers*" allows to specify if the first row of the data contains the column names. The "*Column Delimiter*" can be freely chosen as can be seen in the "Settings"-section of Figure 4.5 (3) enabling Oracle to *extract the structure out of the external file* – a process which is called "*parsing*", being the prerequisite to map the data into the table. Through the setting "*Enclosed by*", the starting and ending boundary of a data value can be delineated. If you specify a delimiter character, Data Workshop ignores white-space occurring before the starting and ending boundary of a data value. You can also use this option to enclose a data value with the specified delimiter character.
4. **Sample Preview for Parsed Data.** Oracle automatically parses the data, as already mentioned, in order to extract the structure of the data, thereby also detecting, e.g., appropriate data types. At the bottom of Figure 4.5 (4), a small sample (up to 10 columns and 5 rows) of the result of this parsing process is shown.
5. **Extended Preview and Further Configurations.** When clicking the "*Preview*"-button, a maximum of 100 columns and 100 rows is shown and further configuration possibilities are offered, allowing now also to change the automatically suggested datatypes and to choose, which of the columns should be actually loaded. In case that in step (1) one has decided to load the data into an existing table, the table has to be selected out of the list of available ones (cf. Figure 4.6 (2) and (3). Afterwards, by clicking on the *"Configure-button"* (cf. Figure 4.6 (4), it is possible to *determine the column mapping*, i.e., which of the automatically identified columns of the loaded data should be mapped to the which columns within the DB table. As can be seen in Figure 4.6 (5) and (6), the column `DEPARTMENTNAME` having the datatype `VARCHAR2` which has been automatically extracted out of the external data file during parsing is now mapped to the DB-table column named "DEPARTMENTNAME" having the same data type.

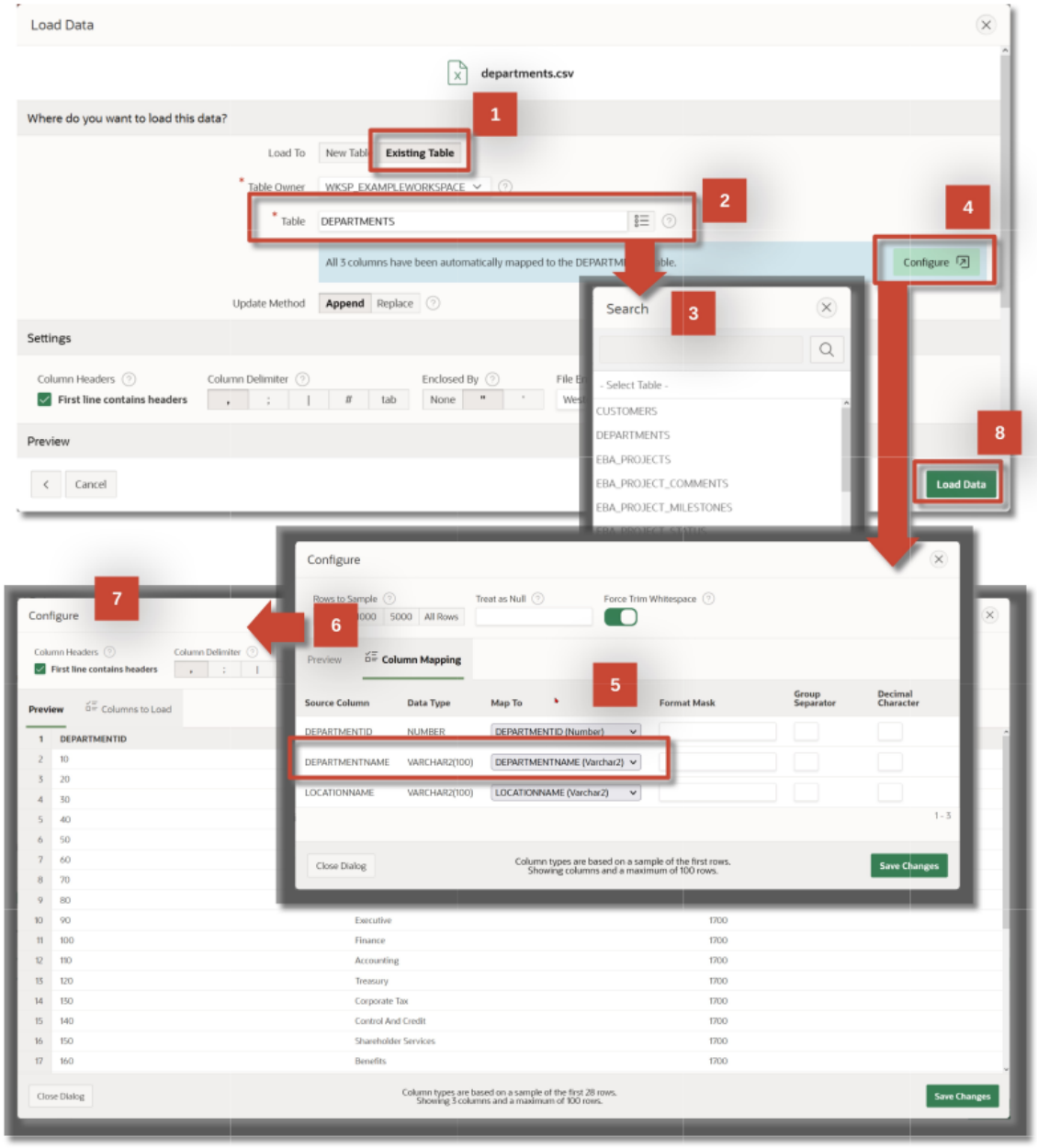


Figure 4.6: Loading Data into Existing Table.

6. **Load Data.** Finally, when clicking the *"Load Data"*-button (cf. Figure 4.6 (8)), a new DB-table is generated (in case this option has been selected in (1)) and the data is loaded into the new or the existing table, whereby the loading dialog informs how many rows have been loaded. The process of data loading runs in the background which is beneficial if a larger file is being uploaded, since the dialog can be dismissed. During this loading process, it could also be the case, that there are some *erroneous rows* encountered, which cannot be inserted into the target table, since, e.g., the one or the other value is conflicting with some data type. In this case, these erroneous rows are automatically saved in an *error table* and can be post-processed manually. One gets also informed by the loading dialog about eventual erroneous rows. The resulting table (and an eventual error table) can now be viewed by navigating to the "Object Browser" or it can be immediately continued with the development of the Web application.

4.1.3 Exporting Data

The “Export Wizard” of Data Workshop can be accessed the same way as described in Section 4.1.1 for the “Load Wizard” (cf. Figure 4.7).

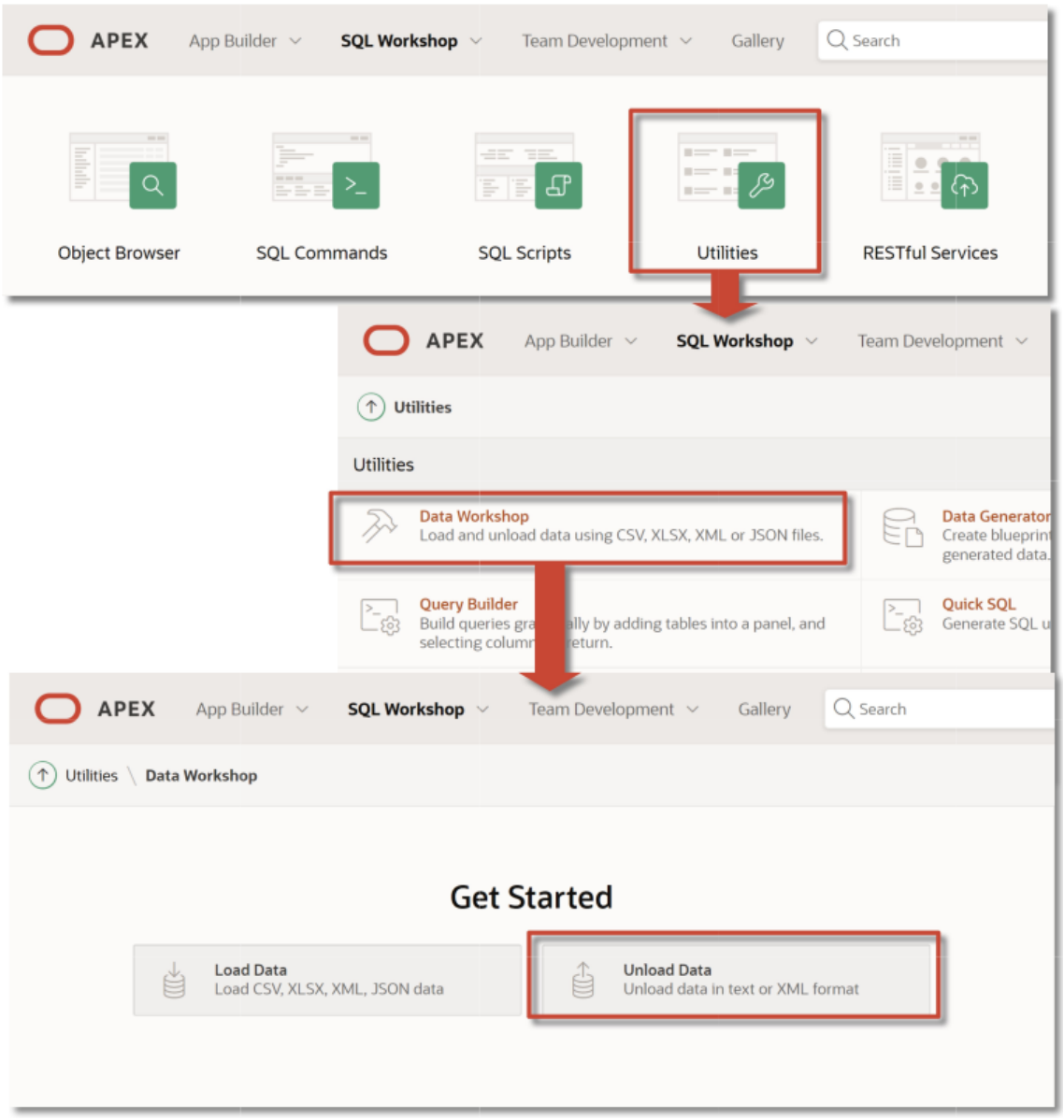


Figure 4.7: Exporting ("Unloading") Data.

In a first step, the file format has to be selected, whereby Data Workshop allows to export data into external files being formatted in terms of CSV or XML (cf. Figure 4.8 (1)). The next step is to select the table as well as the corresponding columns whose data should be exported (cf. Figure 4.8 (2) and (3) respectively). In case that XML has been selected as export format, now the actual export can be started. In case of CSV, some further options can be selected, determining the delimiter between the different rows, if the row names should be included within the output file and if the data format should be DOS or UNIX (cf. Figure 4.8 (4)), before finally activate "Unload Data" to save the data export (cf. Figure 4.8 (5)).

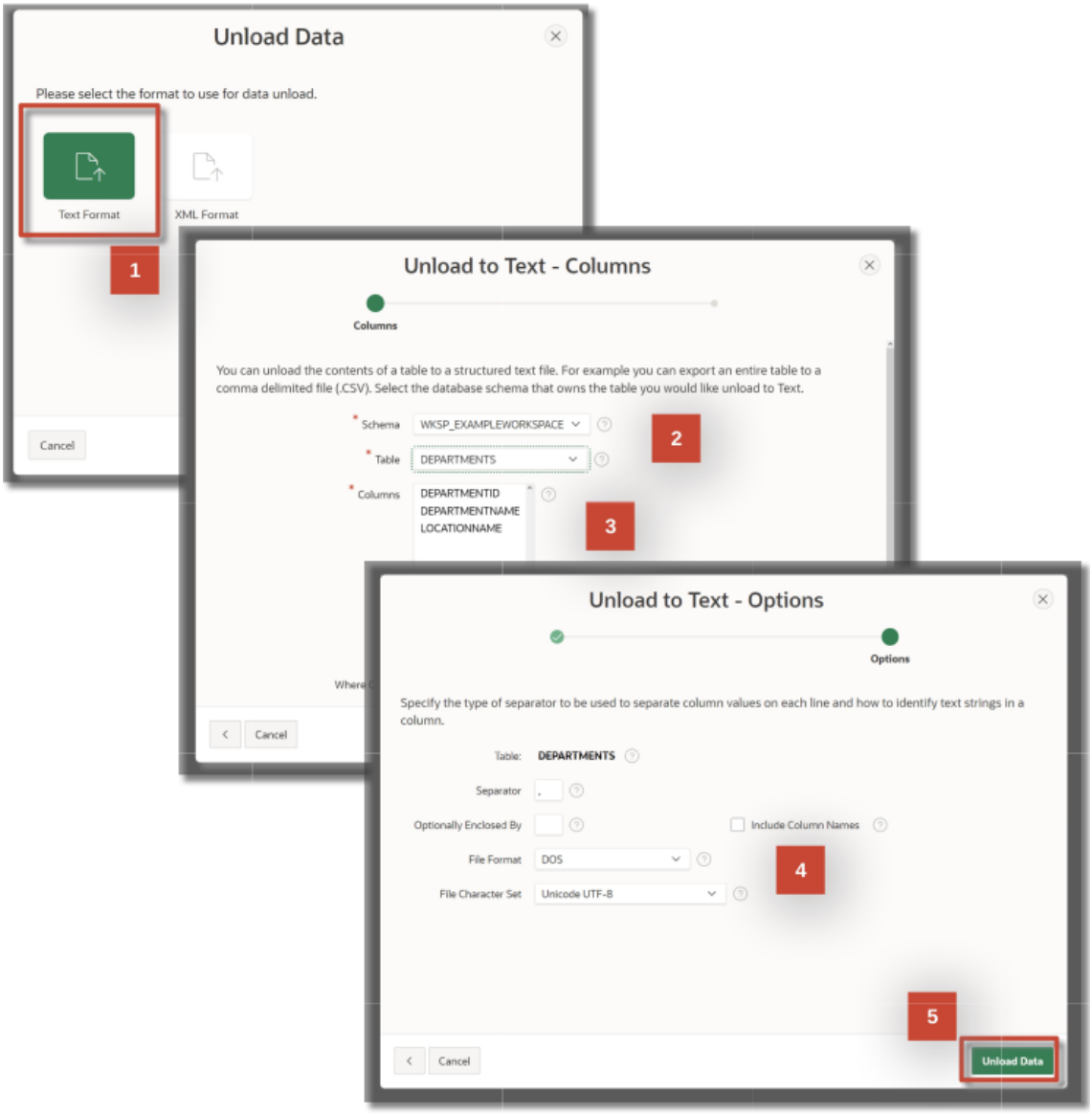


Figure 4.8: Unload Data Wizard.

Regarding DOS vs. UNIX, please note that there is a subtle difference between these two options: DOS files have different line endings than files created on Unix/Linux. DOS uses carriage return and line feed ("\r\n") as a line ending, whereas Unix uses just line feed ("\n"). Thus, you need to be careful when transferring files between Windows machines and Unix machines to make sure the line endings are set properly.

4.2 Importing and Exporting Data Using "Object Browser"

Another alternative is to use SQL Workshop's "Object Browser" for importing data from and exporting data to a file, whereby this can be done in a table-wise manner. Thus, first of all, the respective table has to be selected (cf. Figure 4.9 (1)).

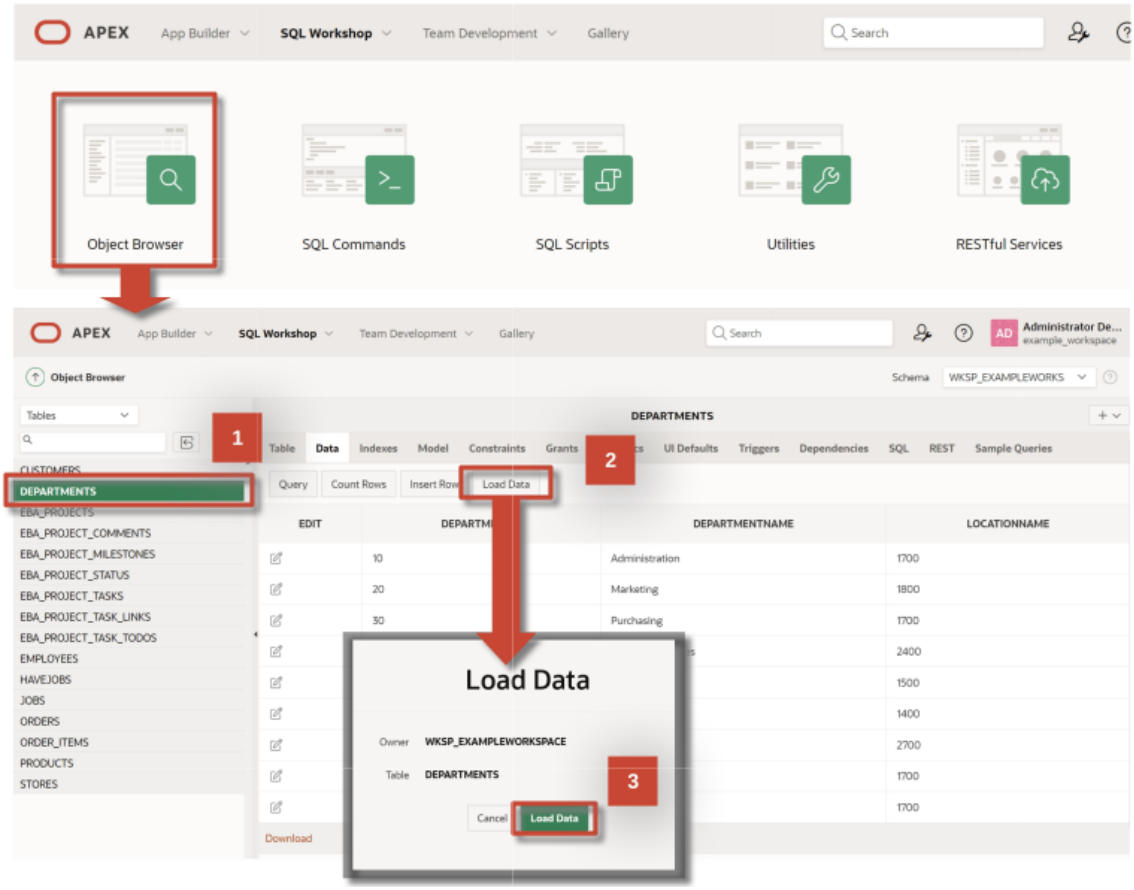


Figure 4.9: Import/Export of Table Data Using the Object Browser.

If now data should be *imported*, after clicking the *"Load Data-button"*, the same dialog appears as when selecting "Load Data" (cf. Figure 4.9 (2), (3)) as already described in Section 4.1.1 - also the same file formats are supported. At the other hand-side, the *export* of data can be simply done by clicking the *"Download-button"* at the bottom of the page, leading to the immediate generation of a CSV file containing the data of the DB-table.

4.3 Exporting the Result of a SQL-Command

Similar to exporting the whole data stored in a table, also a certain subset of this data – or even the data stored within several tables - can be exported at once. Simply stating, the result set of an arbitrary complex SQL-Query can be exported (cf. Figure 4.10 (1))

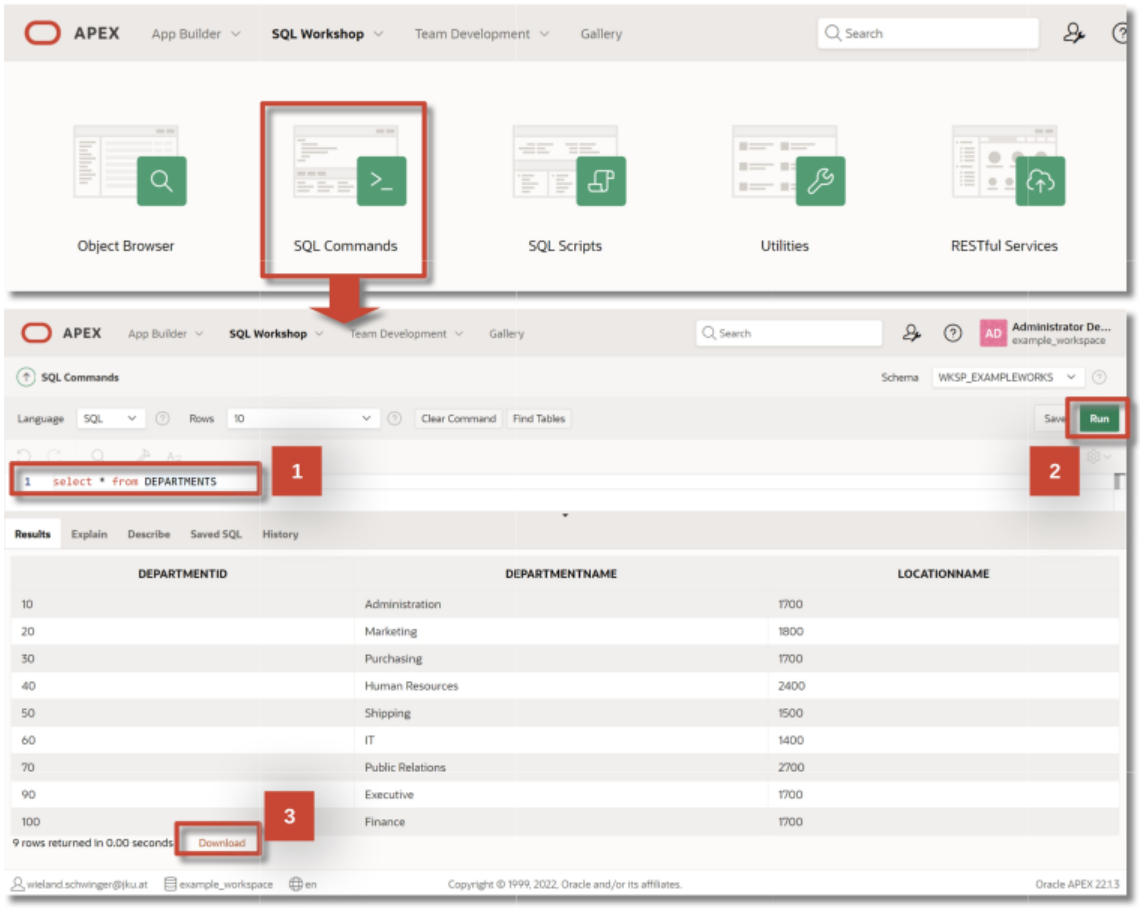


Figure 4.10: Data Export of the Result of a SQL-Query.

For this, after the query has been executed within the “SQL command”-tool of SQL Workshop and the result set is shown (cf. Figure 4.10 (2)), this result set can be downloaded by just clicking on the “Download”-button (cf. Figure 4.10 (3)).

4.4 Exporting Data from an Application Report

As we discussed in Chapter 3, developing a Web application using the APEX’ App Builder requires to select a certain page type for each page which is added to the Web application. In case that one has selected, e.g., the page type “Interactive Report”, the data displayed by this report during runtime can be exported. This is illustrated in Figure 4.11, showing a sample Interactive Report Page “DepartmentsReport”.

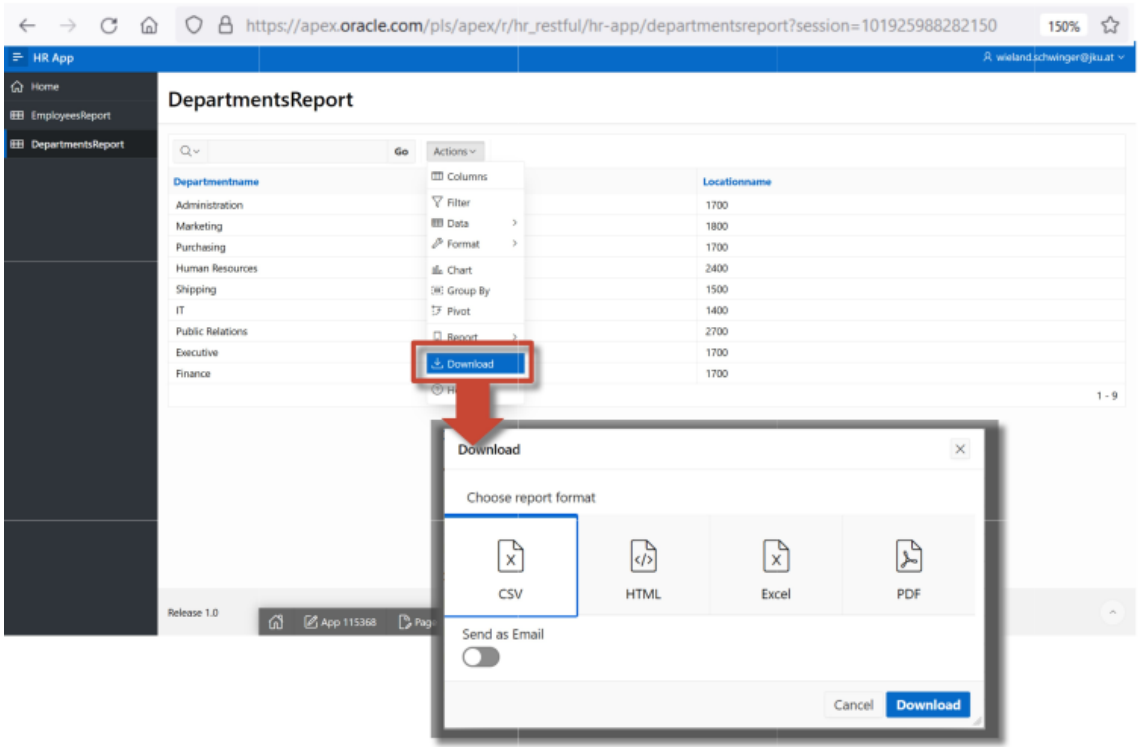


Figure 4.11: Export of Data From an Application Report.

Possible file formats are CSV, HTML, PDF and Excel, also sending the data via eMail is an option.

4.5 Enabling Data Exchange with RESTful Services

Exchanging data of the DB-layer of our Web application by enabling external clients like other Web applications, mobile applications, legacy applications or cloud systems online access to our data can be easily realized using so-called *RESTful services*. *REST (Representational State Transfer)* is in fact an *architectural pattern* proposed by Roy Fielding and Richard Taylor in 2000 [2] for providing interoperability between arbitrary systems over the Internet. It enables the querying and manipulation of data *without the need for direct access* to the underlying DB-tables. Figure 4.12 gives a high-level overview of the basic architectural pattern of REST.

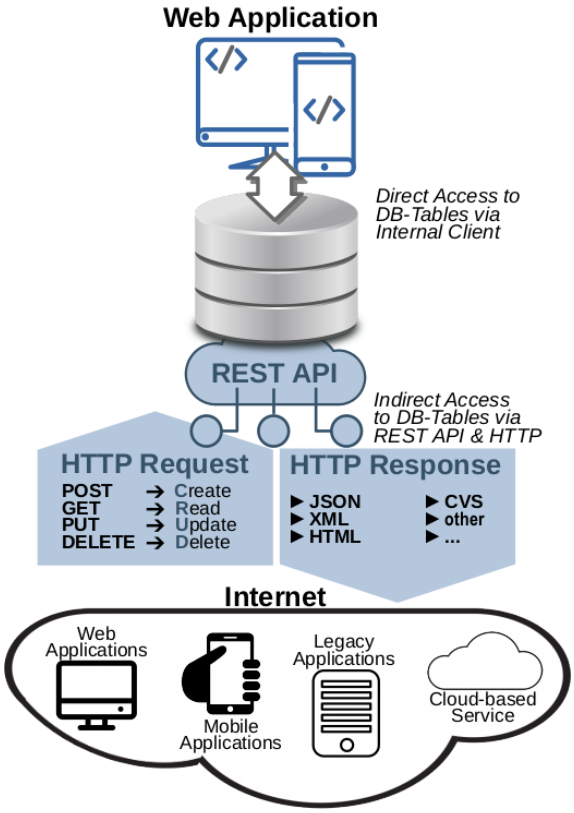


Figure 4.12: RESTful Services for Data Exchange – Basic Architecture.

For realizing such an *indirect* access according to the REST architectural pattern, RESTful services have to be created in terms of a set of *APIs (Application Programming Interfaces)* on top of your DB-tables. Without connecting directly to the underlying DB, these APIs enable external systems to interact securely with the data by querying, inserting, updating, or deleting data. The definitions of RESTful services created within Oracle APEX are stored in the *Oracle REST Data Services (ORDS)* repository , and are referred to as *ORDS-based REST Services*.

4.5.1 REST Architectural Pattern

As already mentioned, a service is described as RESTful when it conforms to the tenets of REST. Although a full discussion of REST is outside the scope of this Chapter, a RESTful service has the following characteristics (Note, that the general characteristics of REST as described in [2] have been slightly adapted to our DB context):

1. **DB-tables provided as Resources for Services**. A RESTful Service is modeled as a set of resources, in our case on top of your DB-tables. These resources are identified by URLs and accessed over the `HTTP` or `HTTPS` Web protocols.
2. **Operations sent via HTTP-Requests**. A small set of operations is used to deal with these resources in terms of a `HTTP`-request comprising `POST`, `GET`, `PUT`, `DELETE`, thereby resembling the CRUD-operations (Create, Read, Update, Delete).
3. **Services are stateless**. RESTful Services are *stateless*, i.e., no client context is stored at the server between requests – session state is kept entirely on the client. This means that each request from client to server must contain all of the information necessary for the server to understand the request and cannot take advantage of any stored context on the server. The main benefit is that reusable and scalable services can be created that can be managed and updated without affecting the system, even during runtime. The disadvantage is that it may decrease network performance by increasing the repetitive data (per-interaction overhead)

sent in a series of requests, since that data cannot be left on the server in a shared context.

4. **HTTP-Responses**. Requests to a RESTful Service always elicit a *response*. This response is in the form of XML, JSON, HTML, or some other defined format. Responses provide details of some sort of alteration to the underlying data, error messages, and hypertext links to other related resources depending upon the operation.

4.5.2 Enabling a DB Schema for RESTful Access

In order to enable a DB schema for RESTful access, it has to be first registered with ORDS . This can be simply done as illustrated in Figure 4.13 by accessing the RESTful services tool within SQL Workshop (1), then clicking the “Register Schema with ORDS”-Button (2), entering an arbitrary alias used in the formation of the URL referencing any RESTful service within the schema (3) and finally, after clicking the *"Save Schema Attributes-button"* (4) the ORDS RESTful Services Dashboard appears, used for managing all RESTful services.

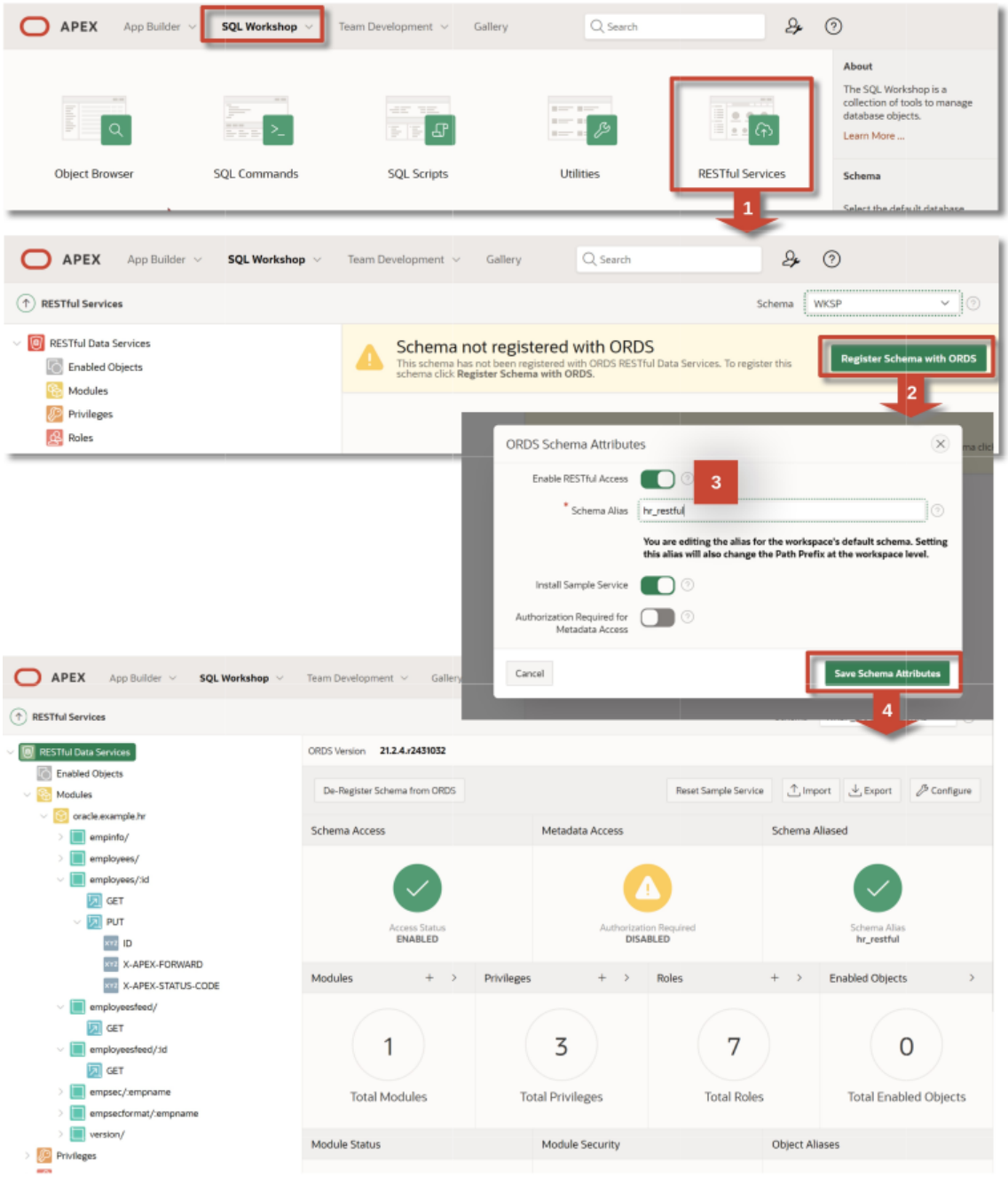


Figure 4.13: Enabling a DB Schema for RESTful Access.

In case that in step (3) the "Install Sample Service" option was enabled, the sample *module* `oracle.example.hr` is installed too, providing example *resource templates* and *resource handlers* (cf. Section 4.5.3) that implement several different operations to retrieve and display employee information from the Employees-table as can be seen in Figure 4.15.

4.5.3 Resource Modules | Templates | Handlers

In ORDS, RESTful services can be created by using three basic components, comprising Resource Modules, Resource Templates, and Resource Handlers (cf. Figure 4.15 for some examples). The overall relationships between these basic building blocks are illustrated by an ER-diagram in Figure 4.14.



Figure 4.14: Relationships Between the Different Components of ORDS RESTful Services.

- **Resource Modules.** A Resource Module is nothing else but a container that groups a set of related RESTful services together, based on a certain *REST-enabled DB Schema*. The DB Schema can contain several resource modules. A Resource Module not only provides a way to identify the group uniquely, but also defines the unique Base Path used on a URI *to access the set of services* defined within the module. For example, a Resource Module that enables you to access information about employees is named `oracle.example.hr` and the base path value for the service is `/hr/`, cf. Figure 4.15.
- **Resource Templates**. A Resource Template defines an individual service that can be called, e.g., `employees/:id`, Figure 4.15. Resource Templates are contained in Resource Modules. Each Resource Template defines a *URI Pattern* where it can be reached, e.g., "empinfo/" (Figure 4.15) and contains at least one Resource Handler (cf. below).
- **Resource Handlers.** Each Resource Handler implements one (and only one) of four different HTTP operations, which resemble traditional CRUD-operations (Create, Read, Update, Delete) .
  - `POST` creates a new resource or adds a resource to a collection, equivalent to a SQL `INSERT` statement.
  - `GET` retrieves a representation of a resource, equivalent to a SQL `SELECT` statement.
  - `PUT` updates the values an existing resource, equivalent to a SQL `UPDATE` statement.
  - `DELETE` deletes an existing resource, equivalent to a SQL `DELETE` statement.

You must define a Resource Handler for each operation associated with the same Resource Template. For example, to provide*an operation to return data* and *another operation to store data*, you must define a Resource Handler for each operation (cf. Fig. 4.15 – for the Resource Template `employees/:id`. two operations `GET` (1) and `PUT` (2) have been defined).

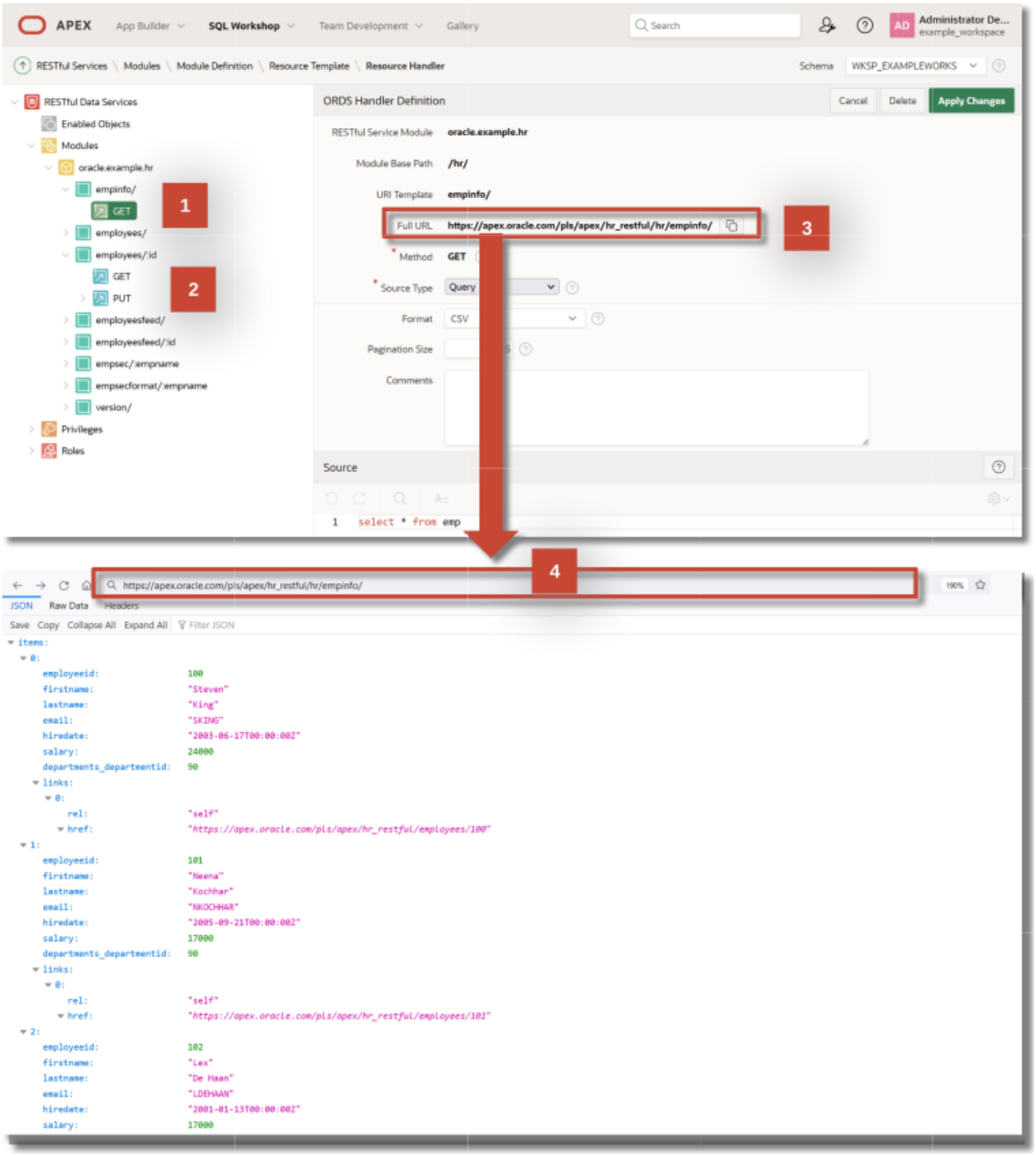


Figure 4.15: Example RESTful Services for Table Employees.

Summarizing, Figure 4.15, gives an exemplary overview of the different ingredients of a RESTful service in terms of Resource Modules, Templates and Handlers, including also the selection of the desired response format (CSV) and the SQL query implementing the operation type of the Resource Handler. After you create a RESTful Service, you can test it by simply entering the generated access URL in your browser (cf. Fig. 4.15 (3) and (4) respectively).

4.5.4 Using "AutoREST" instead of Manually Defining Resources

As could be seen in the previous section, for each kind of DB operation which should be offered to external clients as RESTful service for indirectly accessing our DB, several steps are necessary. For the simplest form of a query, i.e., a full table scan, there is however, also some kind of "shortcut" provided by APEX in terms of the so-called *"AutoREST"-facility*. AutoREST is a quick and easy way to expose DB tables as REST resources. You lose some flexibility and customizability if you

use the AutoREST feature, but it reduces your time and effort to a significant extent. AutoRest lets you quickly expose data but allows only to query the whole table (i.e., a full-table scan), having a fixed output format in terms of JSON only.

AutoREST can be simply used the following way: After having enabled your schema for RESTful access (cf. Section 4.5.2), each DB table can be individually defined as a resource using the Object Browser of SQL Workshop as can be seen in Figure 4.16.

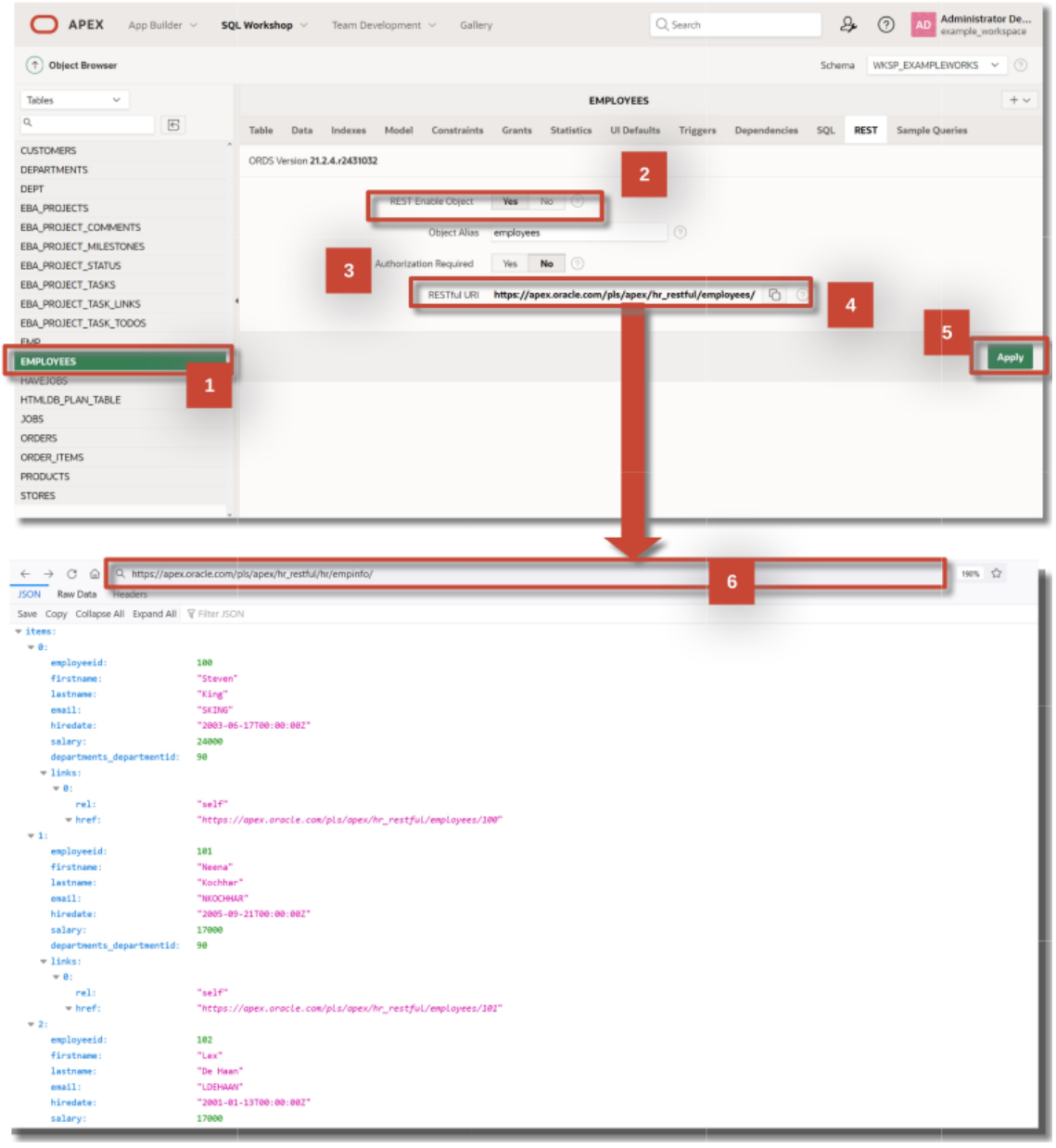


Figure 4.16: Enabling REST Service in the Object Browser.

After selecting the respective DB table (1), clicking on the REST-tab (2), selecting the authorization as required (3) and applying all these settings (5), REST has been activated for the table and the automatically generated access URL appears (4). Again, this REST-service can be simply tested by entering the URL in a Web browser (6).

4.6 Questions

1. What are the use cases for importing data via SQL Workshop and App Builder?
2. What are the different ways to export data?
3. What is REST for ? What is its purpose or benefit?

4.7 Answers

1. Data Workshop allows importing data from external files via a "Wizard" and importing data into a table via the "Object Browser". Data import from Data Workshop can also be initiated via App Builder.
2. Exporting data to a file in a table-wise manner is supported by SQL Workshop's "Object Browser". Data Workshop provides an "Export Wizard" that allows to select the file format and the tables as well as the corresponding columns whose data should be exported.
3. RESTful services provide interoperability between any systems over the Internet by allowing data to be queried and manipulated.

5. How to generate a first draft of the application?

A. ANGEIOPLASTIS, G. MYLLIS, A. TSIMPIRIS, D. VARSAMIS

5.1 Why business need applications?

In today's rapidly evolving business landscape, applications have become a critical tool for companies to stay competitive and meet the ever-changing demands of their customers. Building applications allows businesses to streamline their operations, increase efficiency, and enhance their customer experience, all of which can ultimately lead to greater profitability.

One of the primary reasons businesses build applications is to automate processes that are currently being done manually. For example, an e-commerce company may develop an application that automates their inventory management and order fulfilment processes. By doing so, they can reduce the likelihood of human error and improve the speed and accuracy of their operations.

Another key reason businesses build applications is to enhance their customer experience. Applications can be developed to provide customers with self-service options, such as the ability to track their orders or update their account information. This can help reduce customer wait times and increase customer satisfaction.

In addition, applications can help businesses gather valuable data about their customers, which can be used to improve their marketing strategies and develop targeted advertising campaigns. For example, a retailer may develop an application that tracks customer purchasing patterns, allowing them to send personalized recommendations and promotions to their customers.

Overall, building applications has become a strong business need because it allows companies to operate more efficiently, provide better customer experiences, and gain insights into their customers' behavior. As the business world continues to become more digital, the importance of building applications will only continue to grow.

5.2 Setting up the ORACLE APEX Environment

The aim of this example is to acquaint you with the ORACLE APEX environment, by guiding you through the setup of the APEX environment, demonstrating an SQL workshop example of the employees-departments-projects database, and illustrating how to build an APEX web application based on the data model.

To implement this example, the following steps will be followed:

Step 1: Open a web browser and type in "https://apex.oracle.com/en/". Click on the **Sign In** button, as shown in Figure 5.1.

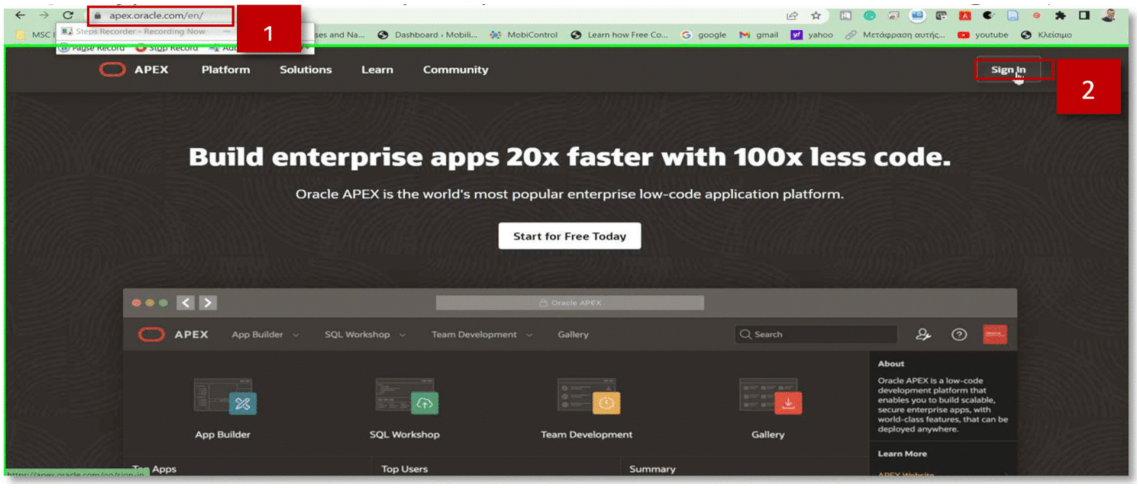


Figure 5.1: Type in APEX link and sign in.

Step 2: Click on the **Request a Workspace** button. In the pop-up window that appears, fill in the required fields for identification and click on **Next**. Then, respond to the survey questions in the next pop-up window and click on **Next**, as shown in Figure 5.2.

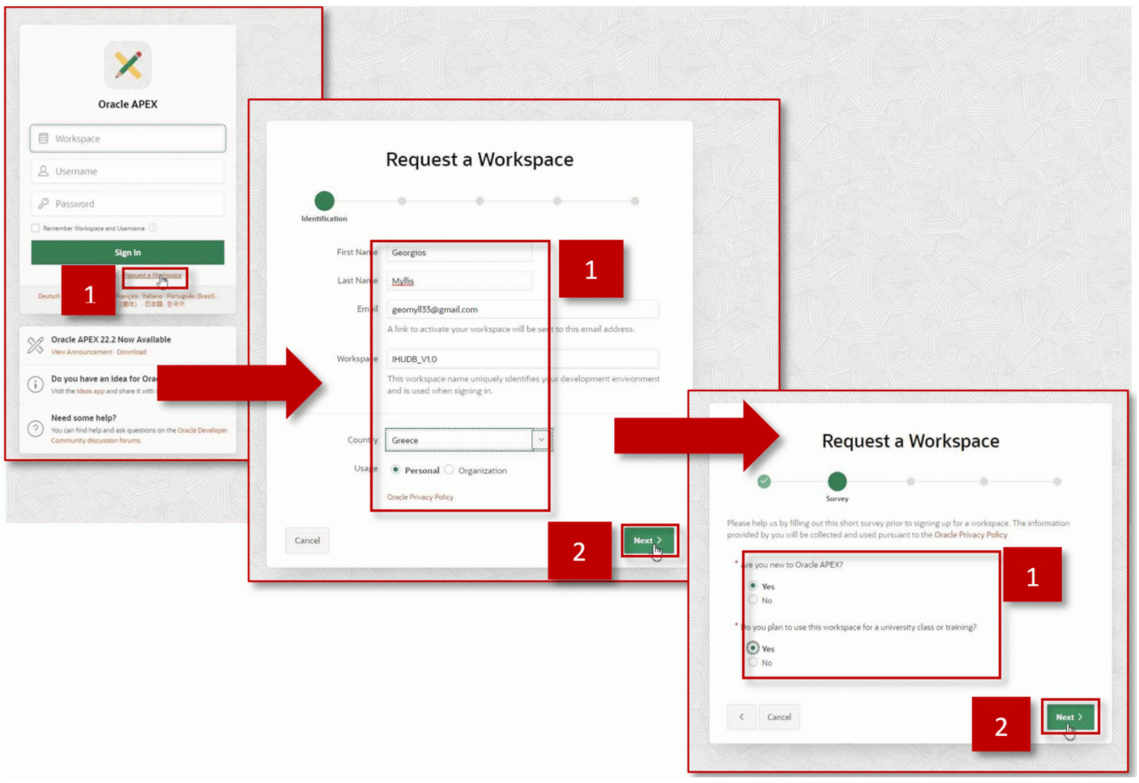


Figure 5.2: Request an Oracle APEX Workspace initial steps.

Step 3: Reply to the information requested for the justification and click on **Next**. In the next pop-up, read the terms of the agreement and then proceed to accept them by clicking on **Next**. In the following pop-up, check the details you have entered for any errors, and then click on **Submit request** as shown in Figure 5.3.

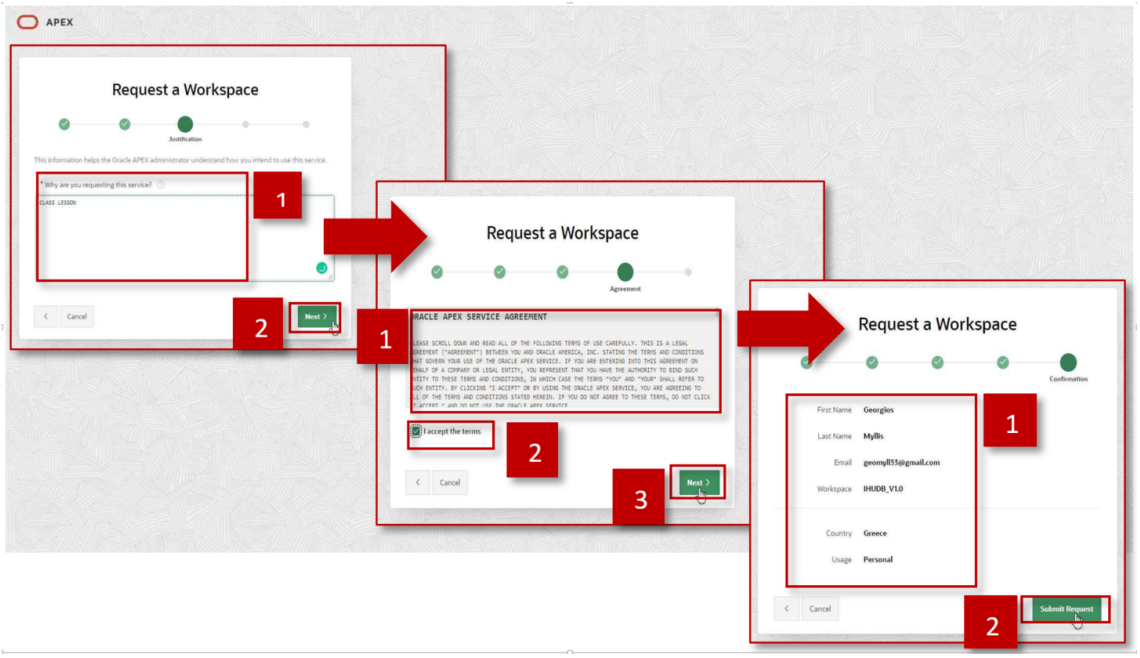


Figure 5.3: Request an Oracle APEX Workspace completion.

Step 4: Once you have submitted the request, a pop-up window will appear, confirming that your workspace has been requested and an activation email will be sent to the email address you have provided, as shown in Figure 5.4.

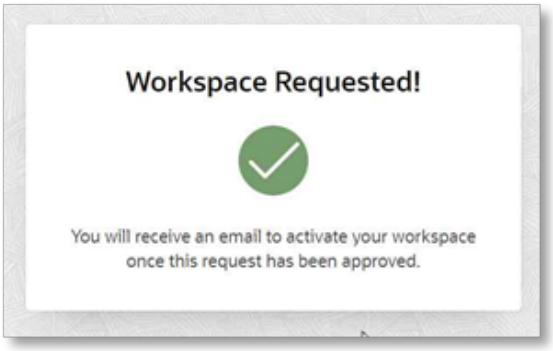


Figure 5.4: Oracle APEX Workspace approved.

Step 5: Check your email for a message from APEX. Your request needs to be approved before you can proceed. Once approved, click on **Create Workspace** as shown in Figure 5.5.

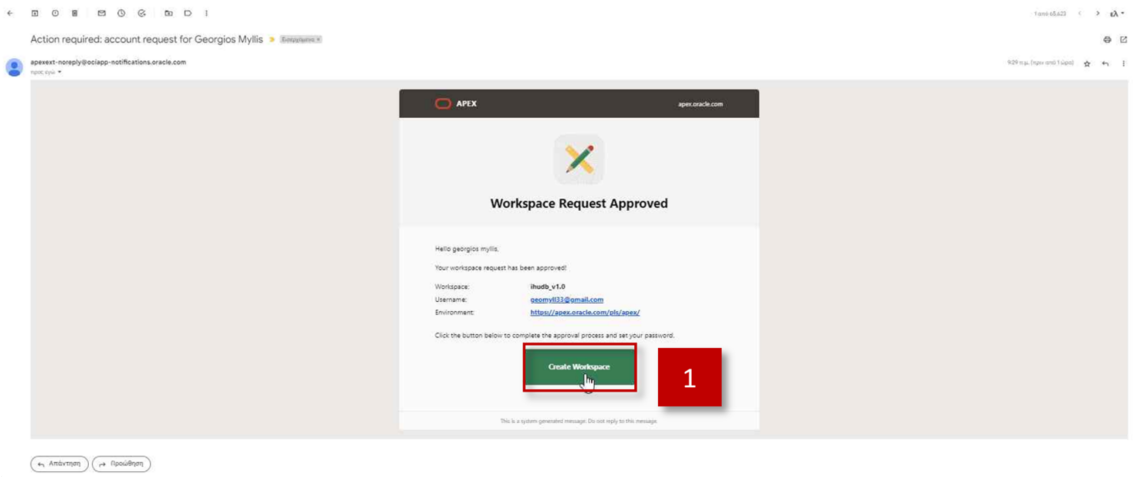


Figure 5.5: Activation email from APEX.

Step 6: Wait until the Workspace creation process is completed. Once completed, a pop-up window will inform you that the Workspace has been successfully created. Click on **Continue to Sign-in Screen**. A new pop-up window will appear, asking you to change your password before logging in. Set a new password, confirm it, and click on **Change Password**, as shown in Figure 5.6.

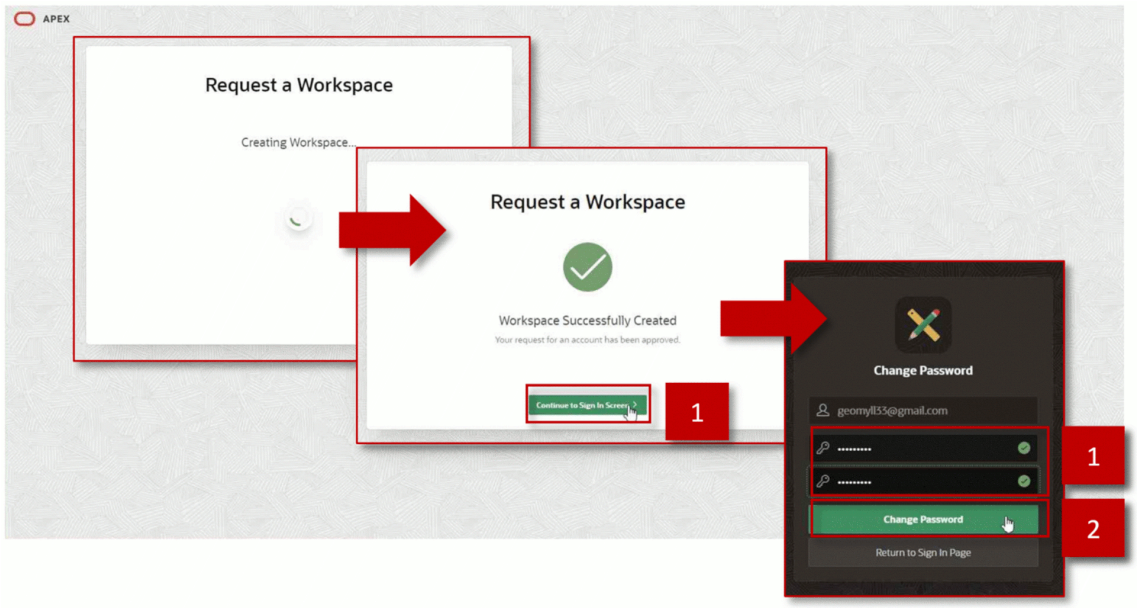


Figure 5.6: Set up a new password for Workspace APEX.

Step 7: The Oracle APEX environment is now ready to be used, as shown in Figure 5.7.

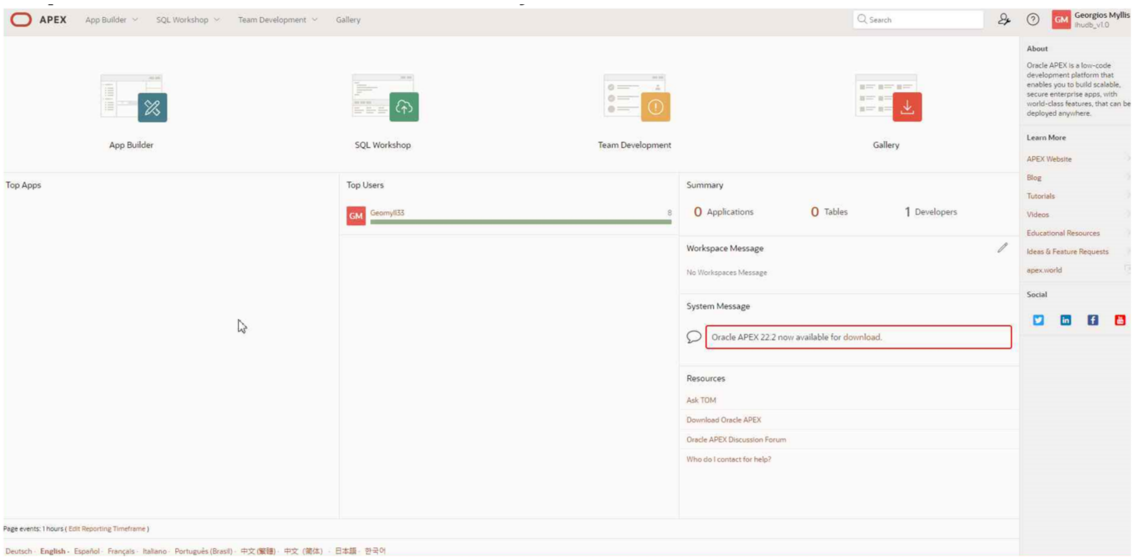


Figure 5.7: Workspace APEX environment.

5.3 SQL workshop example development

To create a database in the APEX environment, there are multiple approaches available. One option is to utilize the Object Browser, a graphical tool that enables users to create tables and their respective fields by completing a pre-designed template.

Another alternative is to employ SQL commands, which empower users to create tables and other database objects by executing SQL statements. This method offers greater flexibility and control over the database structure, although it requires more advanced knowledge.

On the other hand, Quick SQL is a tool provided by Oracle APEX that facilitates the rapid generation of SQL scripts for creating database tables, constraints, and sample data. Quick SQL simplifies the process of creating database objects without the need for manually writing intricate SQL scripts.

In the present example, we will utilize the Quick SQL method. The script has been generated based on the logical model displayed in Figure 5.8.

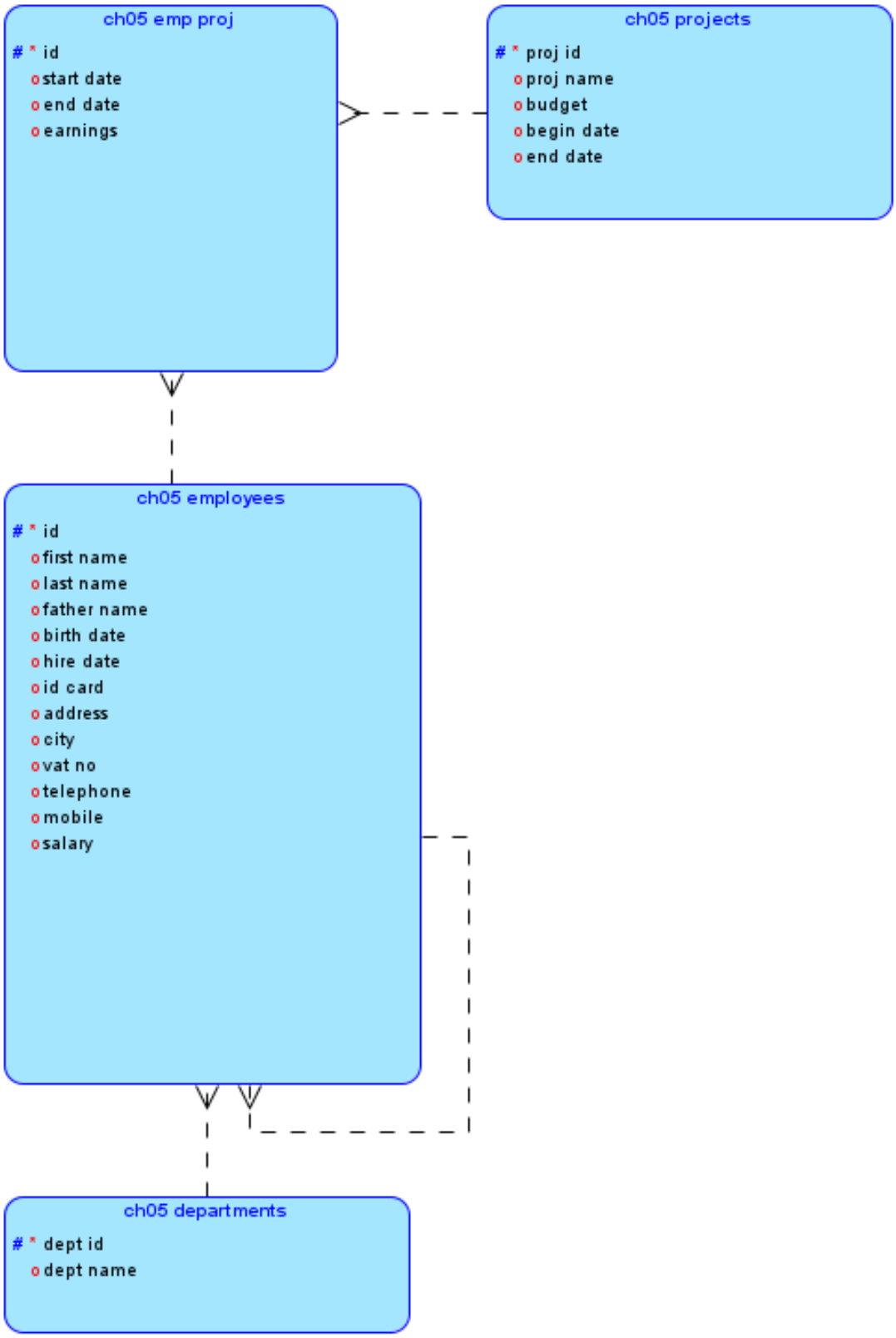


Figure 5.8: Logical Model of our Running HR Example.

According to the relational model derived from the logical model using Oracle Data Modeler

(ODM), as depicted in Figure 5.9, we can proceed with implementing the following structure in the database.

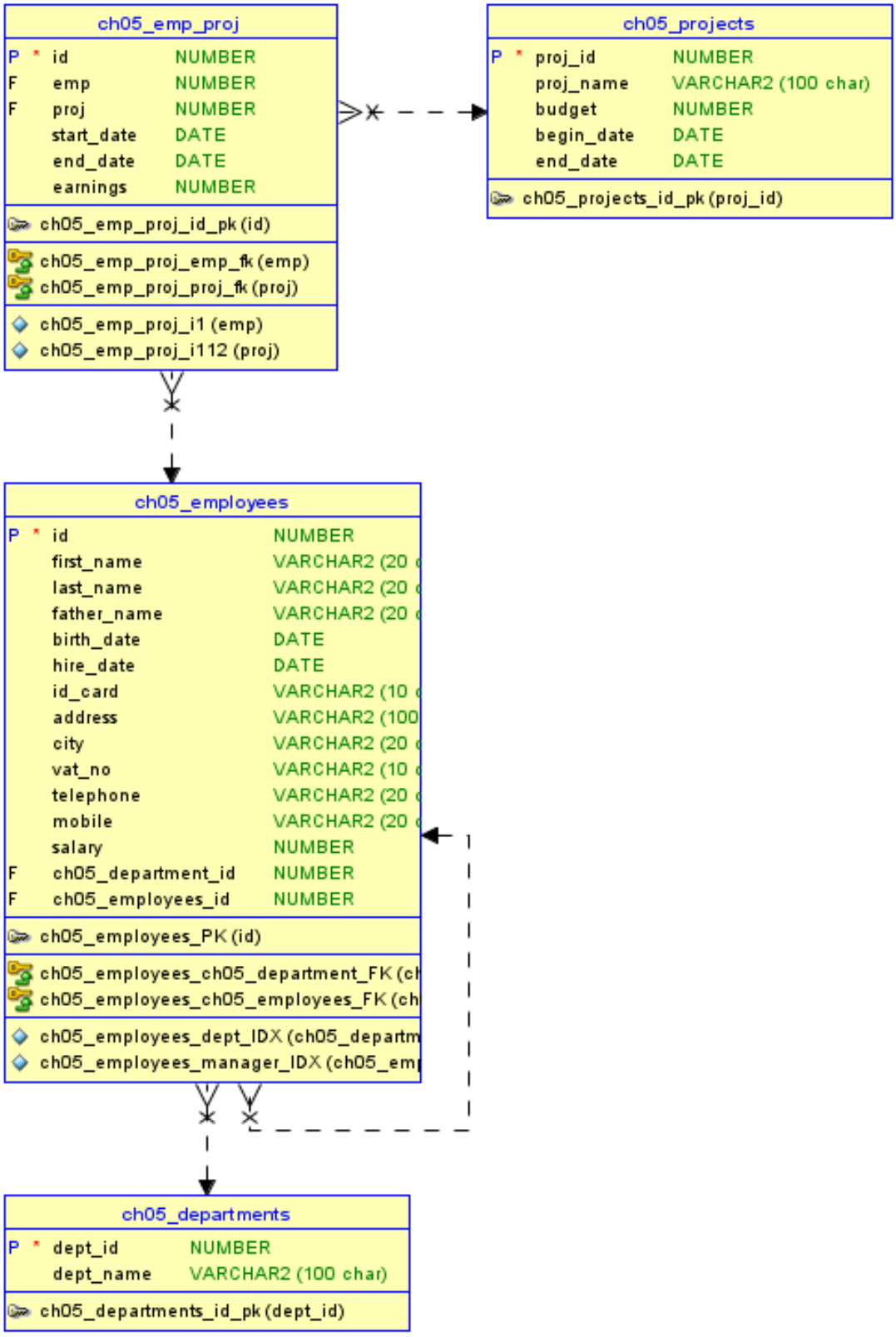


Figure 5.9: Relational Model of our Running HR Example.

5.3.1 Quick SQL

The provided code appears to be a set of instructions for generating sample data using the Quick SQL tool. Each section begins with a table name followed by the "/insert" keyword and a number indicating the desired number of rows to be inserted into the table.

```
ch05_departments /insert 5
    dept_name vc100

ch05_projects /insert 15
    proj_name vc100
    budget num
    begin_date date
    end_date date

ch05_employees  /insert 50
    first_name vc20
    last_name  vc20
    father_name vc20
    birth_date date
    hire_date  date
    id_card    vc10
    address    vc100
    city       vc20
    vat_no     vc10
    telephone  vc20
    mobile     vc20
    salary     num
    dept num /fk ch05_departments
    manager num /fk ch05_employees

ch05_emp_proj /insert 500
    emp num /fk ch05_employees
    proj num /fk ch05_projects
    start_date date
    end_date date
    earnings num
```

These instructions provide a schema and sample data generation plan for a database, related to departments, projects, employees, and their associations.

Step 1:As depicted on Figure 5.10, follow the steps below:

1. Navigate to the **SQL Workshop** section.
2. Select **SQL Scripts** from the options.
3. Click on **Quick SQL**.
4. Insert the provided Quick SQL code into the designated area.
5. Click on **Generate SQL**. The generated SQL will be displayed in the Oracle SQL Output pane.
6. Click on **Save SQL Script**.
7. Enter a script name.
8. Click on **Save script**.
9. Proceed by clicking on **Review and Run**.

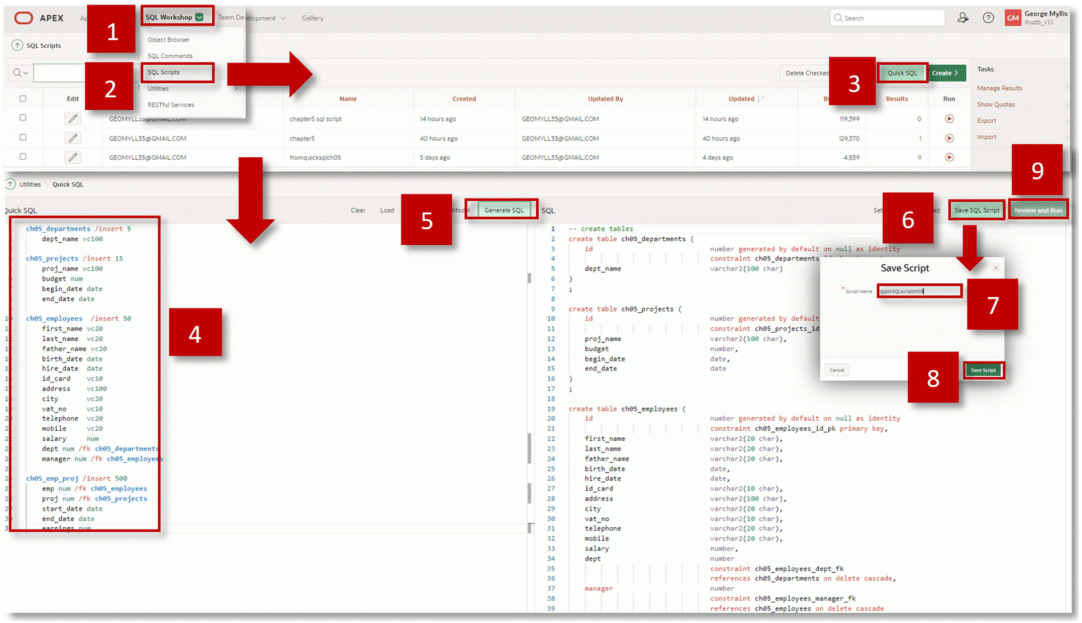


Figure 5.10: Insert Quick SQL code to APEX workspace.

5.3.2 SQL Script

As previously mentioned, the following is the SQL script generated from Quick SQL, which creates a database schema for a sample HR management system. The schema encompasses tables for departments, employees, projects, and employee-project relationships. Additionally, it incorporates indexes and constraints to guarantee data consistency and referential integrity.

```
-- create tables
create table ch05_departments (
    id                          number generated by default on null as identity
                                constraint ch05_departments_id_pk primary key,
    dept_name                   varchar2(100 char)
)
;

create table ch05_projects (
    id                          number generated by default on null as identity
                                constraint ch05_projects_id_pk primary key,
    proj_name                   varchar2(100 char),
    budget                      number,
    begin_date                  date,
    end_date                    date
)
;

create table ch05_employees (
    id                          number generated by default on null as identity
                                constraint ch05_employees_id_pk primary key,
    first_name                  varchar2(20 char),
    last_name                   varchar2(20 char),
    father_name                 varchar2(20 char),
```

```
    birth_date                  date,
    hire_date                   date,
    id_card                     varchar2(10 char),
    address                     varchar2(100 char),
    city                        varchar2(20 char),
    vat_no                      varchar2(10 char),
    telephone                   varchar2(20 char),
    mobile                      varchar2(20 char),
    salary                      number,
    dept                        number
                                constraint ch05_employees_dept_fk
                                references ch05_departments on delete cascade,
    manager                     number
                                constraint ch05_employees_manager_fk
                                references ch05_employees on delete cascade
)
;

-- table index
create index ch05_employees_i1 on ch05_employees (dept);
create index ch05_employees_i62 on ch05_employees (manager);

create table ch05_emp_proj (
    id                          number generated by default on null as identity
                                constraint ch05_emp_proj_id_pk primary key,
    emp                         number
                                constraint ch05_emp_proj_emp_fk
                                references ch05_employees on delete cascade,
    proj                        number
                                constraint ch05_emp_proj_proj_fk
                                references ch05_projects on delete cascade,
    start_date                  date,
    end_date                    date,
    earnings                    number
)
;

-- table index
create index ch05_emp_proj_i1 on ch05_emp_proj (emp);
create index ch05_emp_proj_i112 on ch05_emp_proj (proj);
```

The ch05_departments table stores information about departments, including a unique identifier (id) and a name (dept_name). The ch05_employees table stores information about employees, including a unique identifier (id), first_name, last_name, father_name, birth_date, hire_date, ID_card number, address, city, VAT_number, telephone number, mobile number, salary, and department (dept) and manager (manager) IDs. The ch05_projects table stores information about projects, including a unique identifier (id), name, budget, start_date, and end_ date. The ch05_emp_proj table stores information about employee project relationships, including a unique identifier (id), employee (emp) and project (proj) IDs, start_date, end_date, and earnings.

Indexes are created on the ch05_emp_proj table for the employee and project columns to

improve performance when querying the table based on those columns. Indexes are also created on the ch05_employee table for the department and manager columns for the same reason.

Finally, constraints are added to ensure data consistency and referential integrity between the tables. The ch05_emp_proj_emp_fk and ch05_emp_proj_proj_fk constraints ensure that the employee and project columns in the ch05_emp_proj table reference valid id values in the ch05_employees and ch05_project tables respectively.

The ch05_employees_dept_fk and ch05_employee_manager_fk constraints ensure that the departure and manager columns in the ch05_employee table reference valid id values in the ch05_departments and ch05_employees tables, respectively.

Additionally, the ON DELETE CASCADE clause specifies that if a row in the referenced table is deleted, all related rows in the referencing table will be deleted as well, to maintain referential integrity.

Step 2: To initiate the desired action, please refer to Figure 5.11 and click on the **Run Now** button.

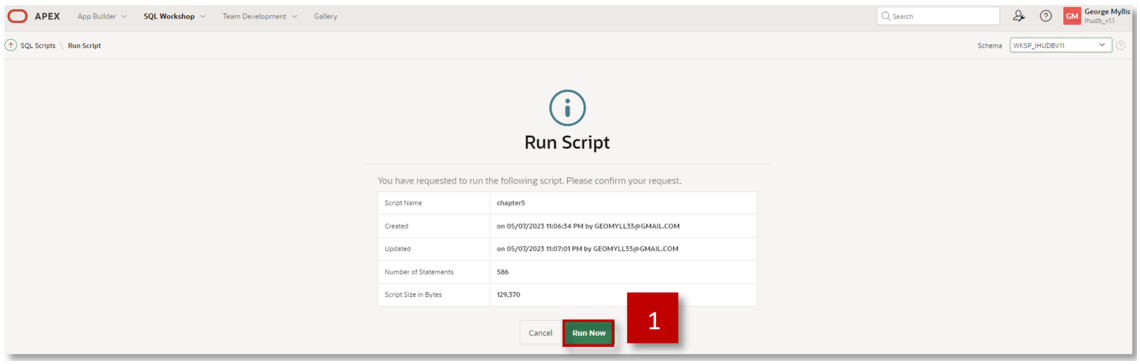


Figure 5.11: Run SQL script.

Step 3: Once all statements have been processed successfully without encountering any errors, please proceed to Figure 5.12 and click on the **Create App** button.

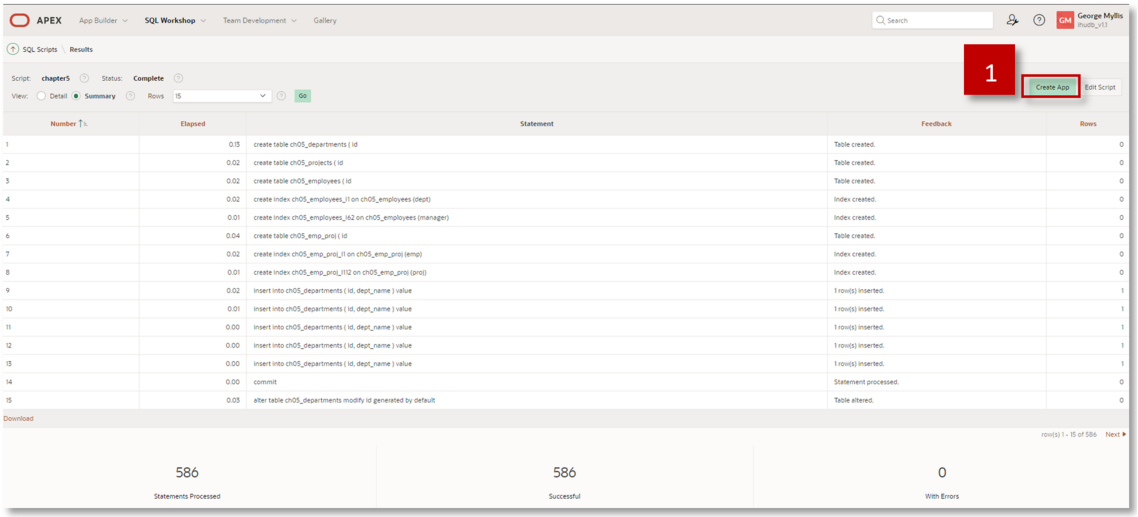


Figure 5.12: Create App starting process.

5.4 Data driven part of application

The data-driven section of the application focuses on harnessing the power of data to drive insights, decision-making, and user interactions. In this section, data takes center stage, enabling users to explore, analyze, and interact with information in a dynamic and intuitive manner.

The primary goal of this section is to provide users with a comprehensive view of the data, empowering them to gain valuable insights and make informed decisions. Through various data visualization techniques, such as charts, graphs, and interactive dashboards, users can easily comprehend complex datasets and identify patterns, trends, and outliers.

Additionally, the data-driven section incorporates robust data management capabilities, allowing users to filter, sort, and search for specific data points or segments. Advanced features such as data aggregation, drill-down, and data export functionality further enhance the user experience, enabling deeper exploration and analysis.

The section also emphasizes the importance of data integrity and data quality. It includes mechanisms for data validation, error handling, and data cleansing to ensure that the information presented is accurate, reliable, and consistent.

With the data-driven section, users can explore data from multiple angles, customize views based on their preferences, and leverage the power of data to drive meaningful insights. It serves as a powerful tool for data analysis, reporting, and decision support within the application, empowering users to make data-informed choices and drive successful outcomes.

Step 1: When the pop-up window labeled **Create App from the Script** appears, in accordance with Figure 5.13,verify that the tables have been created and will be used in your new application, click on the **Create Application** button.

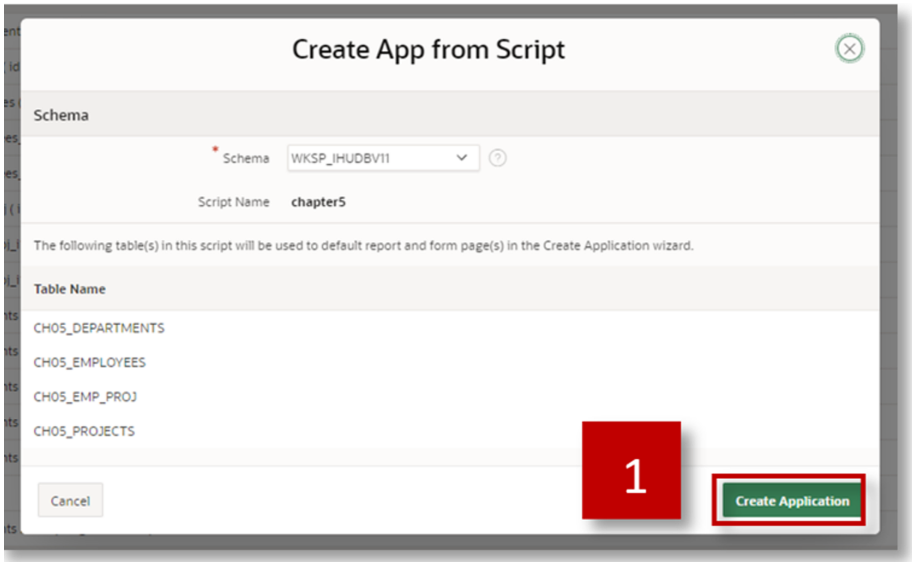


Figure 5.13: Create App from Script.

Step 2: Enter a name in your application, verify that reports and form pages from the tables have been added and will be used in your new application and then click on the **Create Application**, as shown in Figure 5.14

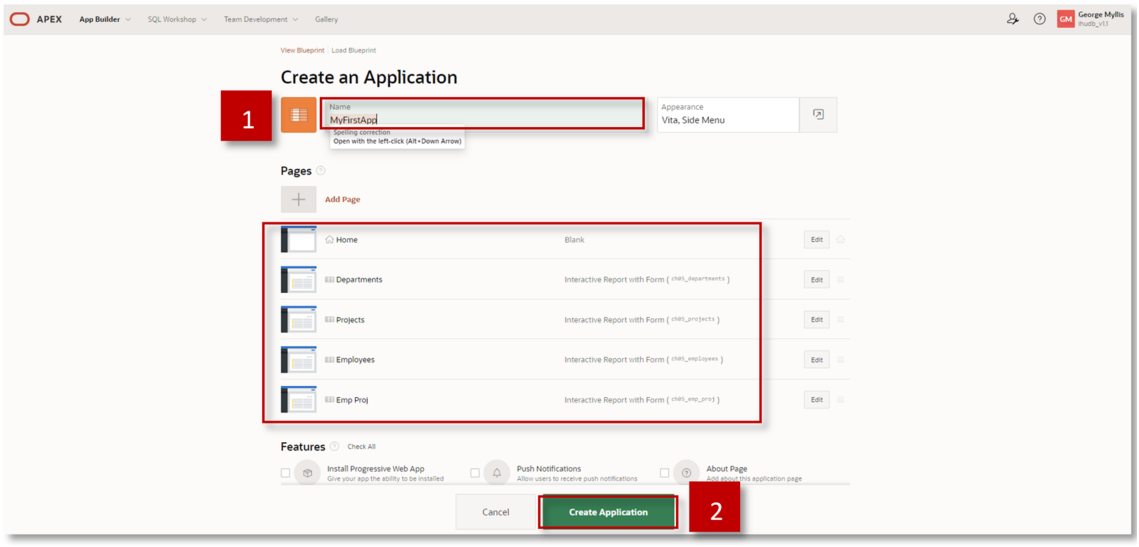


Figure 5.14: Web application created.

Step 3: The app builder environment has been created for your app, as depicted in Figure 5.15. It is important to note that within this environment, you have the ability to customize your app extensively. You can add, remove, or modify form fields, adjust the layout, define validation rules, and perform numerous other actions to tailor the app to your specific needs.

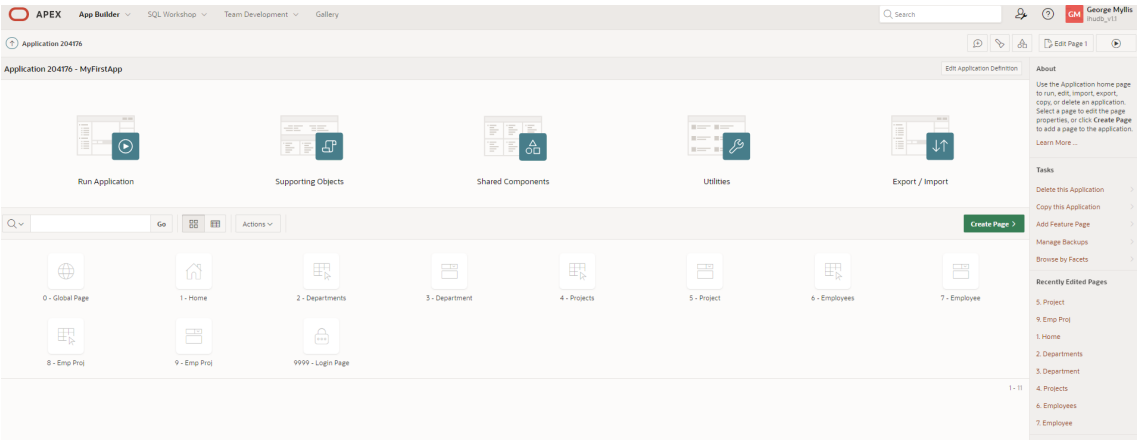


Figure 5.15: App builder environment.

Step 4: Click on the **Run Application** icon in the App builder environment to run the application as shown in Figure 5.16. This will open a new browser page with a dedicated link from APEX for your app. Enter your credentials, which are the same as those in the WORKSPACE, and then click **Sign in**. You will be directed to your home page of your app.

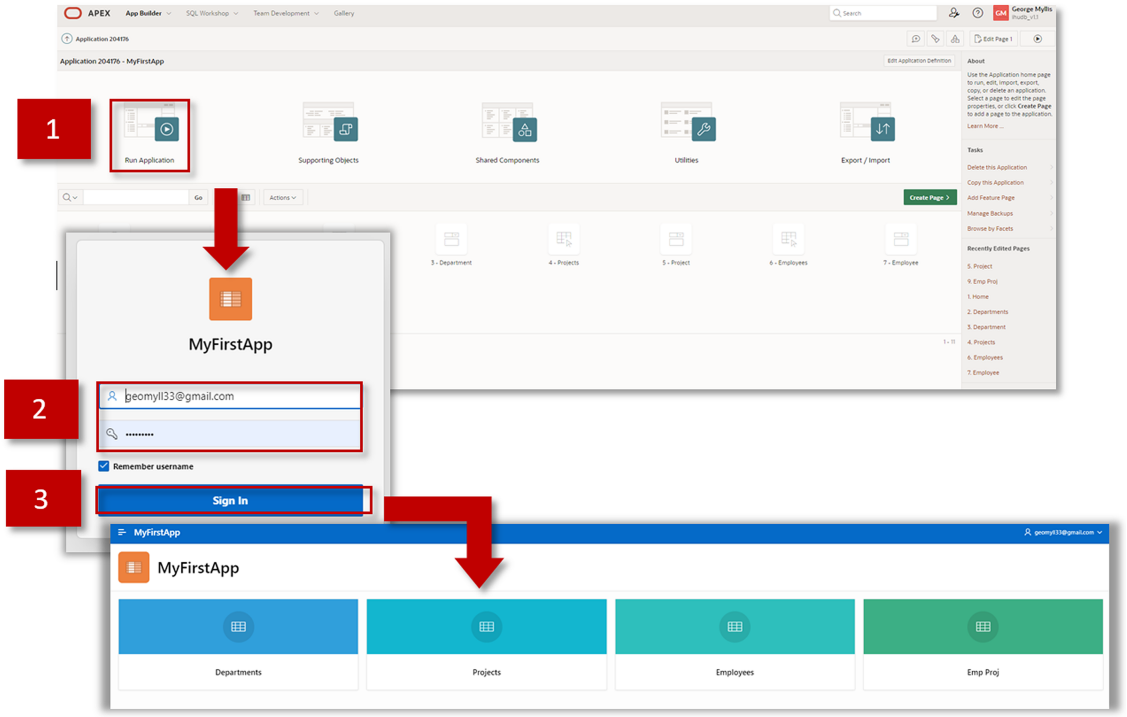


Figure 5.16: App Login Page.

Step 5: You can easily navigate through your app using the side menu, as illustrated in Figure 5.17. The app provides a user-friendly environment that enables you to seamlessly add, modify, and present your data according to your specific needs and preferences.

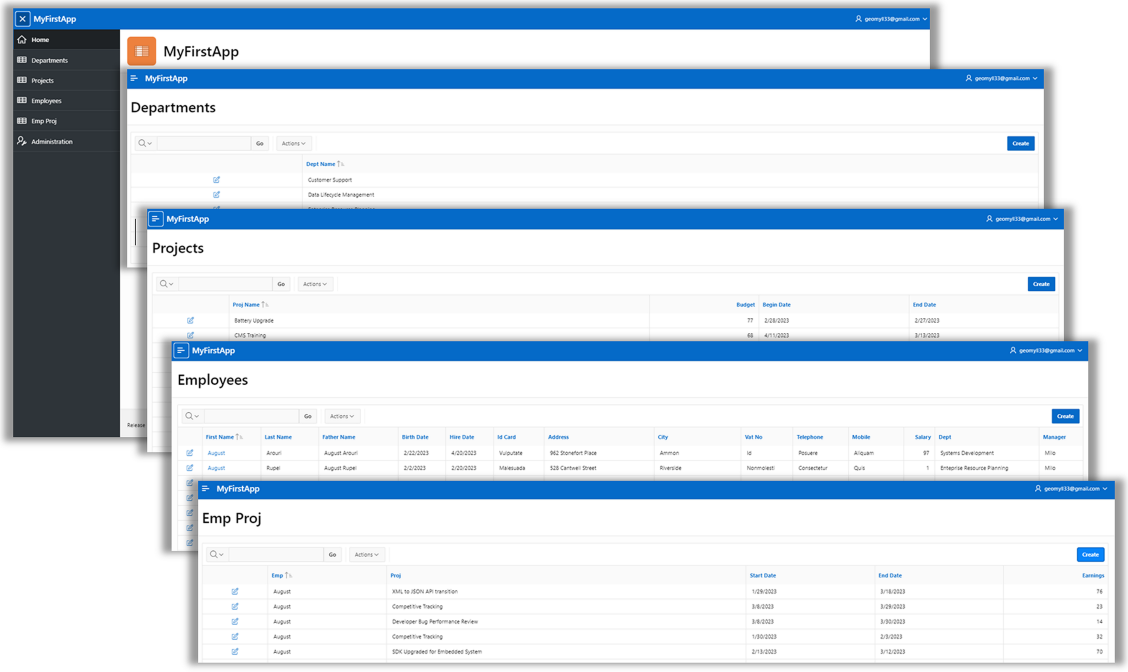


Figure 5.17: Your new App environment.

5.5 Administration of application

Regarding Application Accounts Authentication, it is a built-in authentication method in Oracle Application Express that allows for the creation and management of user accounts in the Application Express user repository. This method is especially useful if your application uses Application Express Accounts authentication.

In the application Workspace environment, click on **Administration** and select **Manage Users and Groups**. The initial account created can be used as an administrator for the built-in application. Oracle APEX User maintains accounts for authentication schemes that authenticate against the username and password stored in these accounts.

Workspace administrators have the authority to create and modify applications and database objects, as well as manage user accounts, groups, and development services. Developers can create and modify applications and database objects, while end-users have no development privileges and can only access applications that do not use an external authentication scheme, as shown in Figure 5.18.

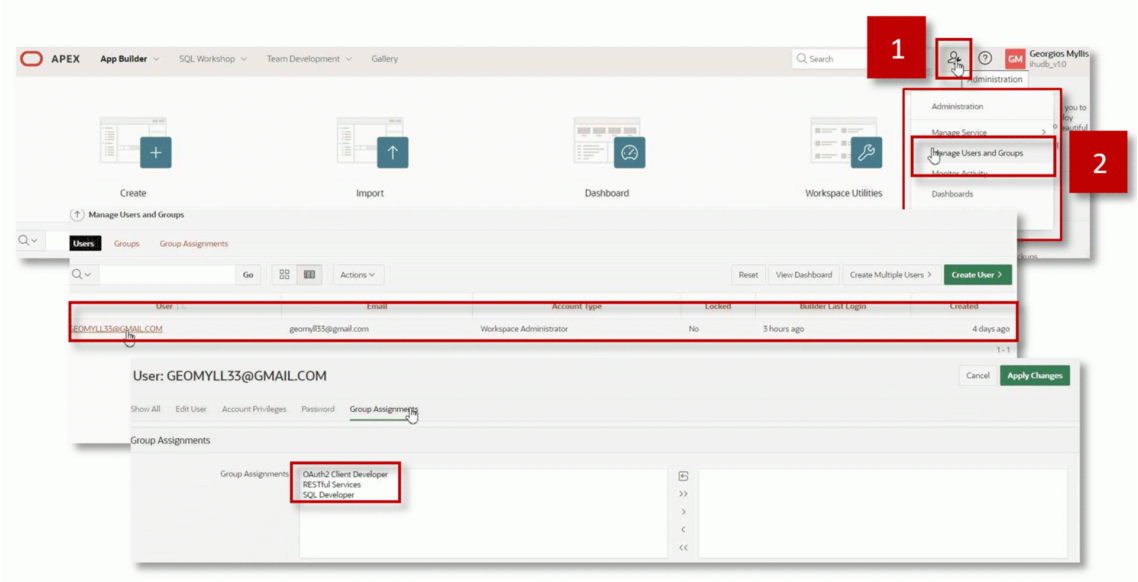


Figure 5.18: Manage users and groups environment.

5.6 Access control

Next, as shown in Figure 5.19, we will provide an example of access control for users who are using the application, following the logic depicted in the figure below.

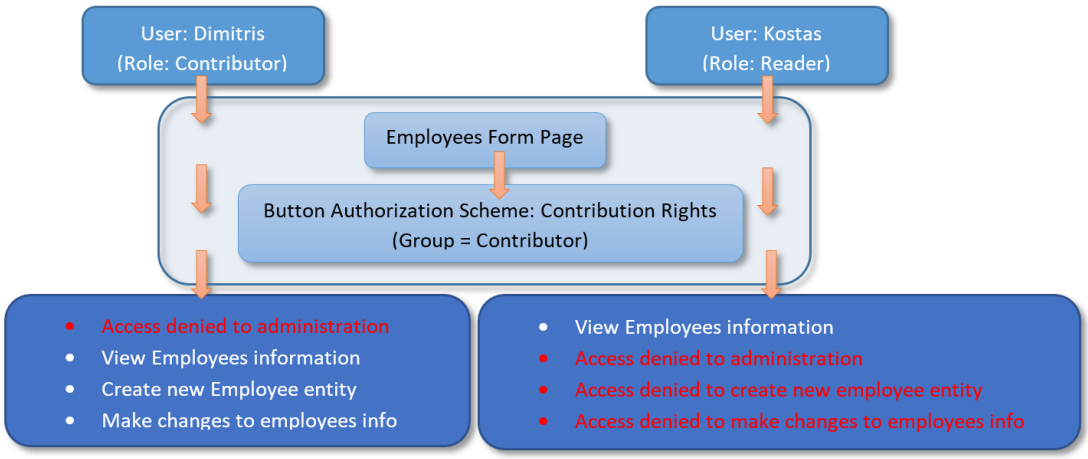


Figure 5.19: Access control example.

Oracle APEX provides three built-in privileges that allow for access control to an application or its components. These privileges are administration, edit, and view, and each corresponds to an access role. Administration corresponds to the administrator role, edit corresponds to the Contributor role, and View corresponds to the Reader role.

Initially in order to add new users and define their roles is to navigate to **Manage Users and Groups** from the APEX display environment. Click on **Administration**, and select **Manage Users and Groups**. Then, choose **Create User**, enter the user identification and password, and click on **Create User**, as illustrated in Figure 5.20.

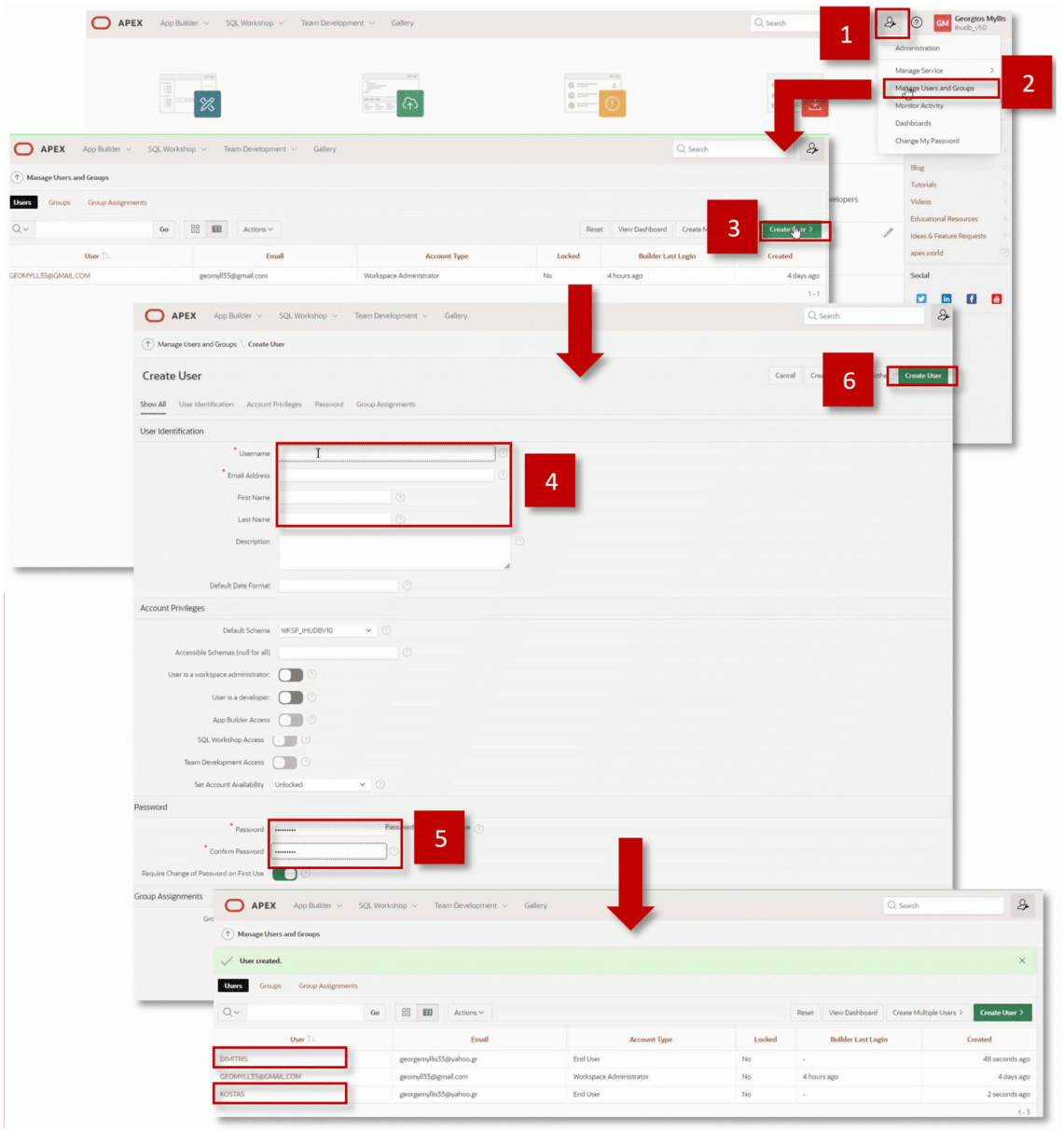


Figure 5.20: Add new users.

Next, you will need to define the roles for your users. In our example, we demonstrate how this can be done within the app itself.

Step 1: Log in to the app with administrator credentials, go to side menu and click on **Administration**, click on **Add User**, enter user identification and role and click on **Add User** on Figure 5.21.

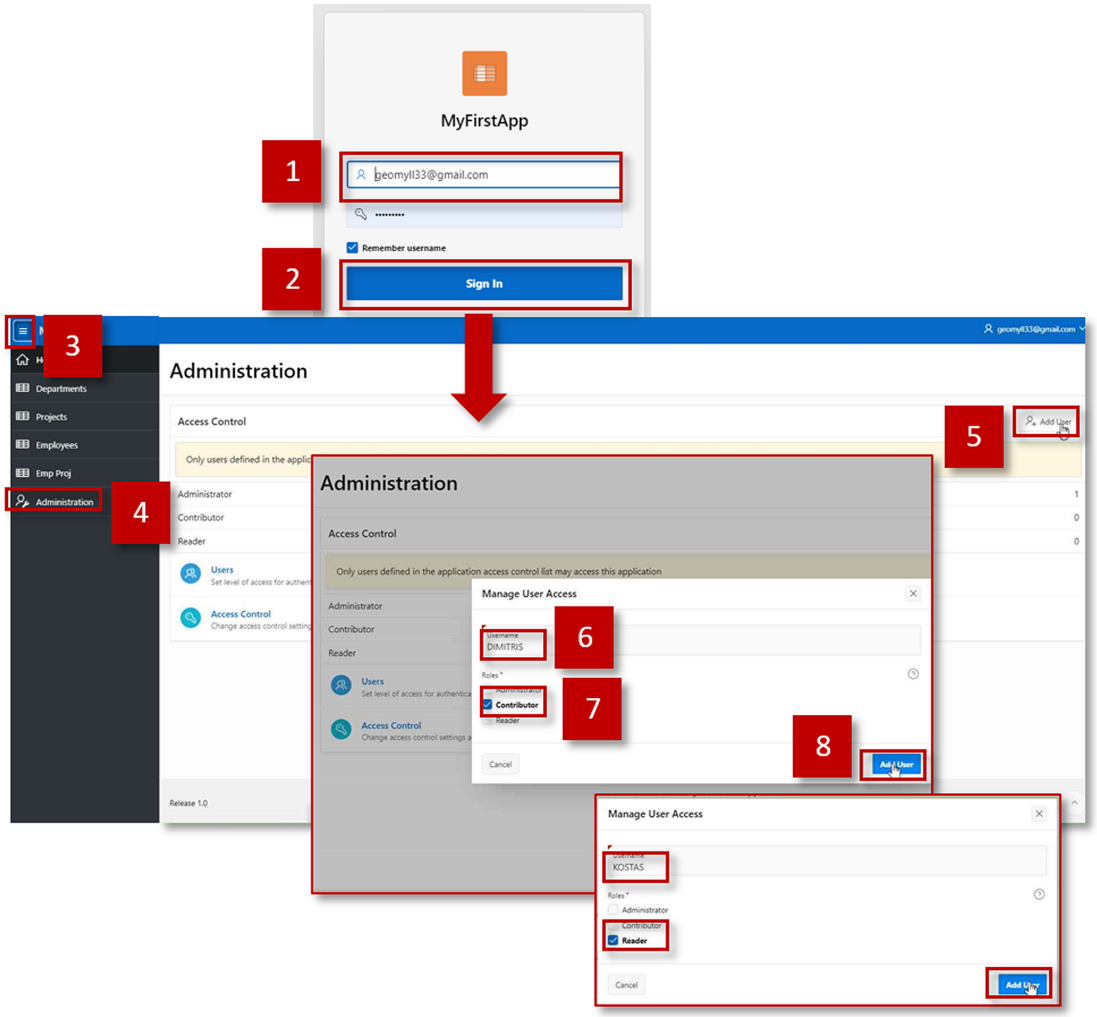


Figure 5.21: Add roles to users.

Step 2: Go to the application workspace environment and click on the **employees-Form**. In the page designer of the Form, select the **Create** button, and in the Security option, choose **Contribution Rights** as the authorization scheme. Click on **Save**. Now, only users with a contribution role will be able to create and add new Employees, as shown in Figure 5.22.

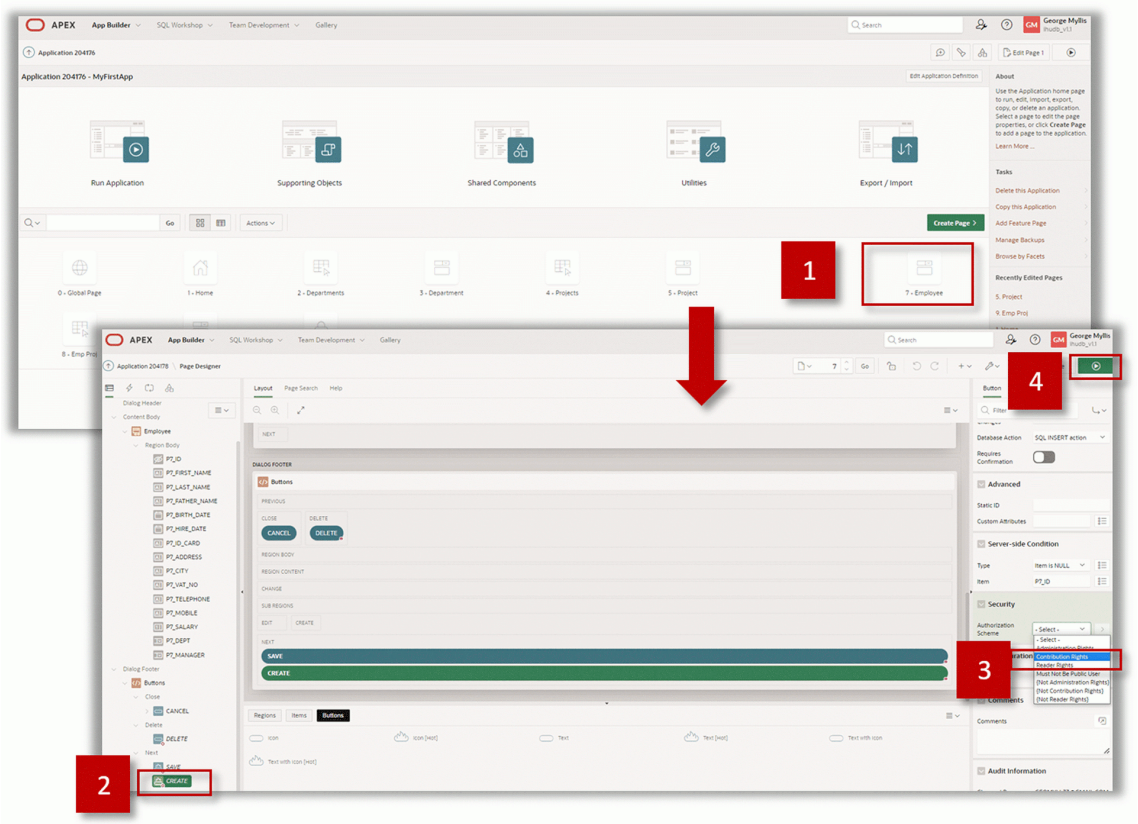


Figure 5.22: Create button configuration.

Step 3: This step is for verifying the access control configurations made for Contributor role.

- Log in to the app with the credentials of user Dimitris, who has the Contributor role assigned.
- Open the menu and verify that the administration option is missing, as expected.
- Try to create and add a new employee as a contributor, as shown in Figure 5.23.
- Verify that you can view the existing Employees' information, and you can make changes.
- Verify that you can create a new employee entity.

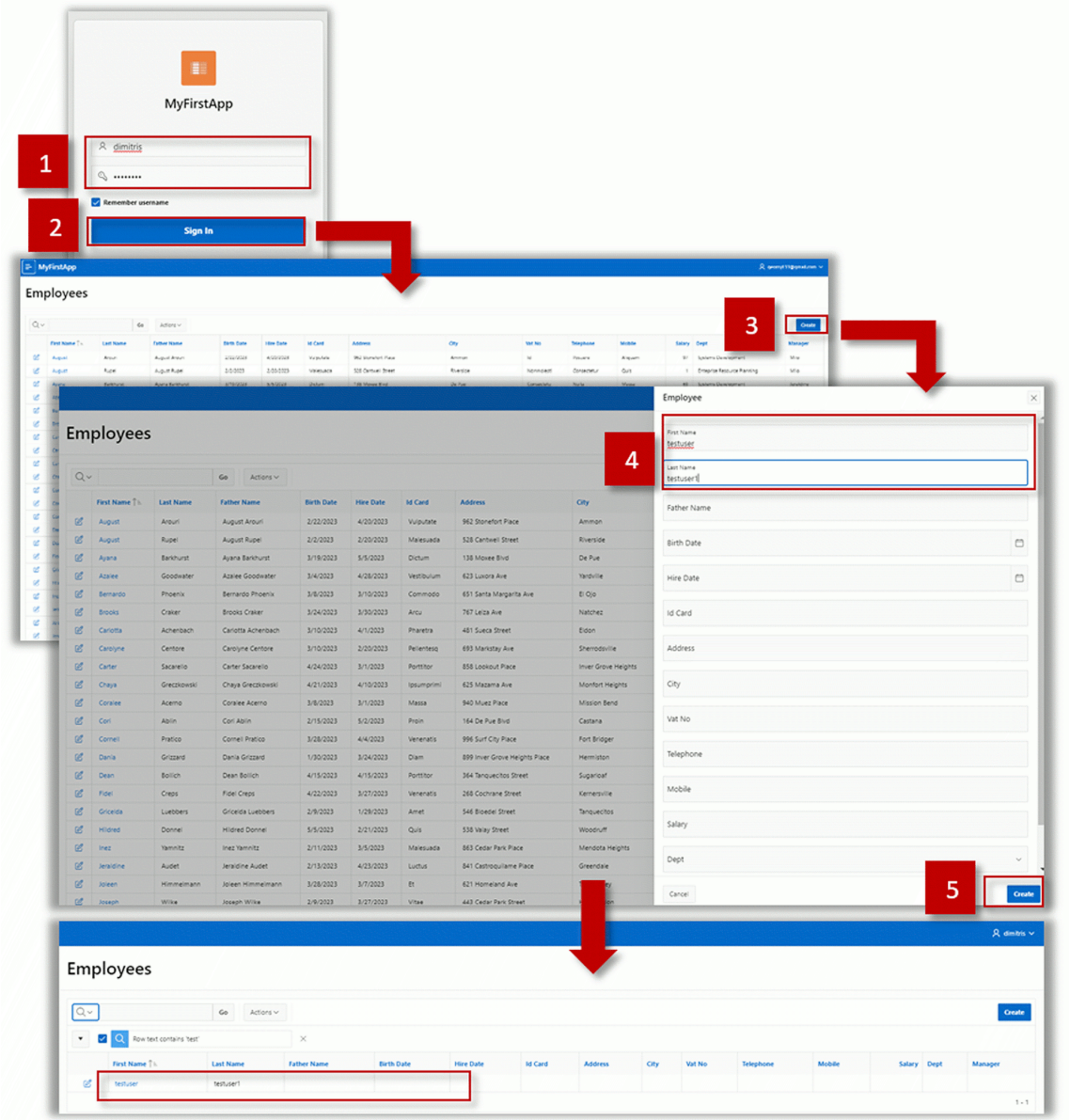


Figure 5.23: Contributor access and rights.

Step 4: This step is for verifying the access control configurations made for Reader role.

- Log in to the application with the credentials of user Kostas, who has a Reader role assigned.
- Open the side menu and verify that the administration option is missing, as expected.
- Click on the Employees-Form in the application workplace environment.
- Try to create or add a new worker. You will receive an "access denied" message, as shown in Figure 5.24.
- Verify that you can view the existing Employees' information, but you cannot make any changes.

These steps confirm that the access control configurations have been successfully applied to the users.

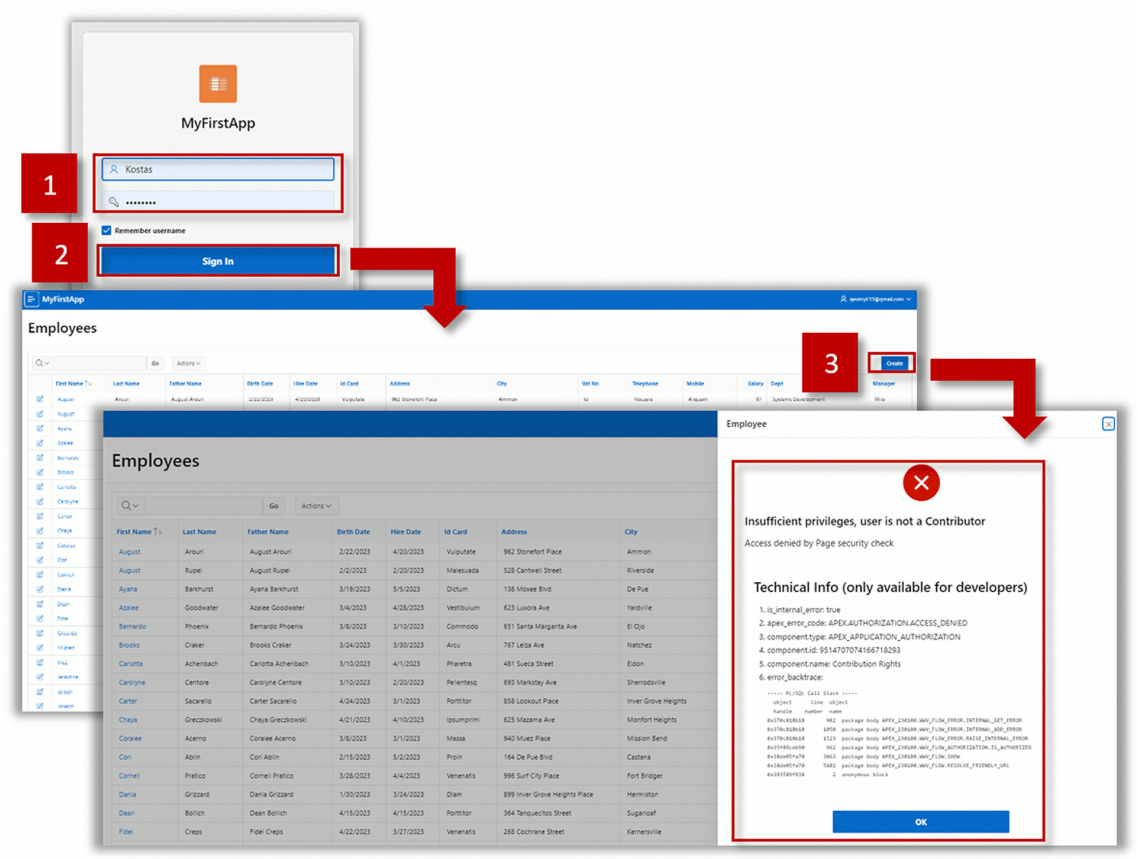


Figure 5.24: Reader access and rights.

5.7 Supplementary learning material

You can find the following supplementary learning material:

- scripts
- application
- video guides

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter 5 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

5.7.1 Exported applications

Exported application is packaged. Installation create tables, index, function, procedure and trigger as well it populate data. De-installation removes all data base objects used in this application.

5.7.2 Video guides

Video guide show every step in application development.

5.8 Questions

1. How many ways are there to create a new application in ORACLE APEX application Builder?
2. How can you add users to the application
3. How many built-in user privileges are there in ORACLE APEX for access control to an application or its components?

5.9 Answers

1. There are 3 ways
    - New Application
    - From a File
    - Starter app
2. Navigate to application workspace environment and choose Manage Users and Group
3. There are three
    - Administrator
    - Contributor
    - Reader



6. How to manage reports?

ZUZANA ŽILLOVÁ, ERIK MALINA, MATEJ GROCHAL, ANDREJ STANÍK, ANDREA MELEKOVÁ, MICHAL KVET, MIROSLAV POTOČÁR

The data in the databases are often meaningless if we do not get outputs from them that told us more about their real information that they contain. To represent the results, observe and find out, we just use reports, which we will deal with in the following chapter. We will learn how to easily create such a report in the APEX environment and how to edit it. We will deal with different types of reports such as interactive reports, classic reports, Column Toggle Report or reports with graphs visualizations.

6.1 Report

Report is a *region* which simplifies data browsing and displays data in rows. It can figure as one unit or it can be used to divide pages into logical units. It can be static, with only title and formatting, but without any additional functionality. Or it can be a dynamic region, having specific functionality like *calendar, report* or *list*. These regions are pre-programmed with its basic functionality that can be also edited easily by users.

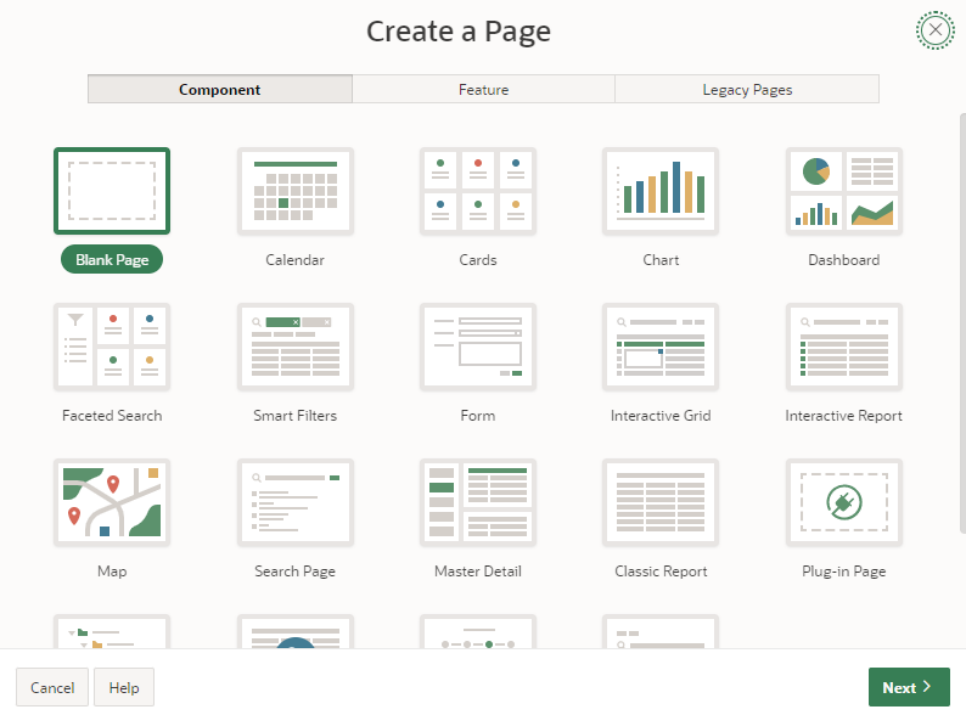


Figure 6.1: Selection of page type.

As you can see in Figure 6.1 when creating a **page**, users can select from pre-built page types which already contain reports, so it is not necessary to insert a region by individual user. An example of the form for report creation is shown in Figure 6.2. When creating a page containing a report, it is necessary to fill in name of the page and data source of the page. Data can be loaded from *local database, REST enabled SQL* or *REST* source. When selecting a local database option, it is necessary to select what exactly would be used as a reports source - the user can not only use the database *table* itself but also existing *views* or own *SQL query* statements.

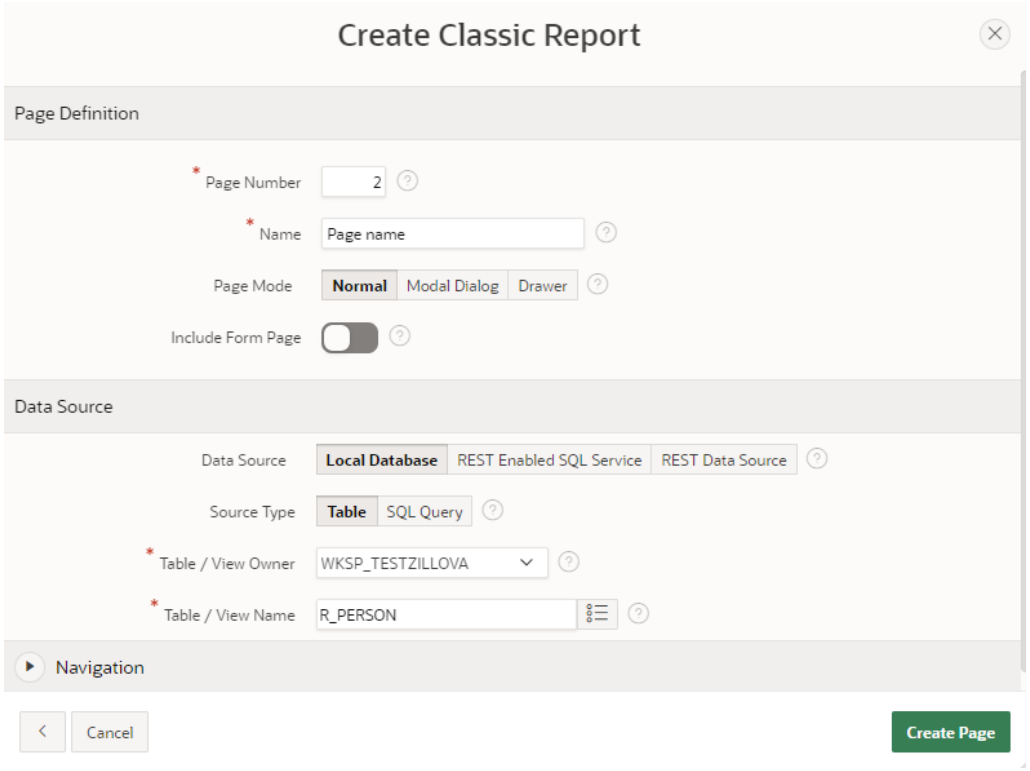


Figure 6.2: Creation of page with report.

When the page is created, a section similar to the one in Figure 6.3 appears on the left side. In the side navbar you can see page, its name and all its components. After clicking on one of the components it gets highlighted. After that you can see columns it contains, and also its sub-regions or items. With the right click you can create a new empty region.

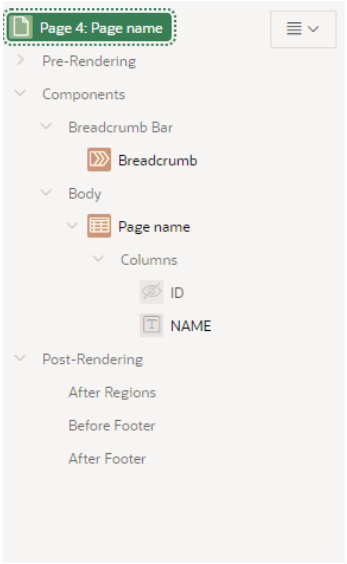


Figure 6.3: Rendering.

In the Figure 6.4 you can see the region and its attribute details. **Identification** defines page the title and type of report (e.g., classic report). Figure 6.5 depicts several examples of source section in which the source of data for our report as well as information about data origin (as mentioned above:

table, view, function or *SQL query*) are defined. **Where Clause** speaks about which condition will be applied for data filtering.

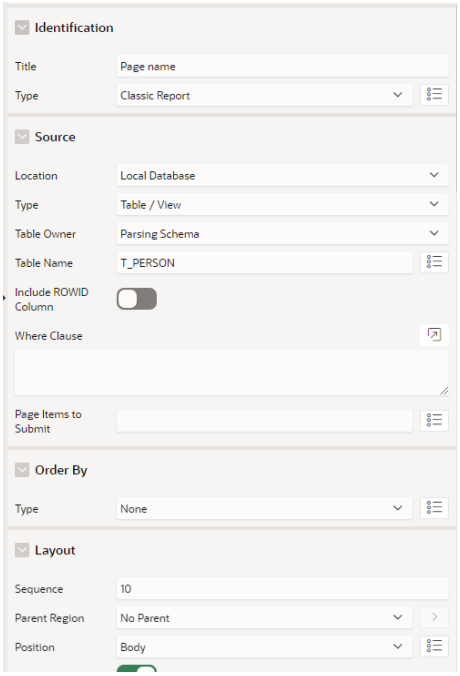


Figure 6.4: Properties of region.

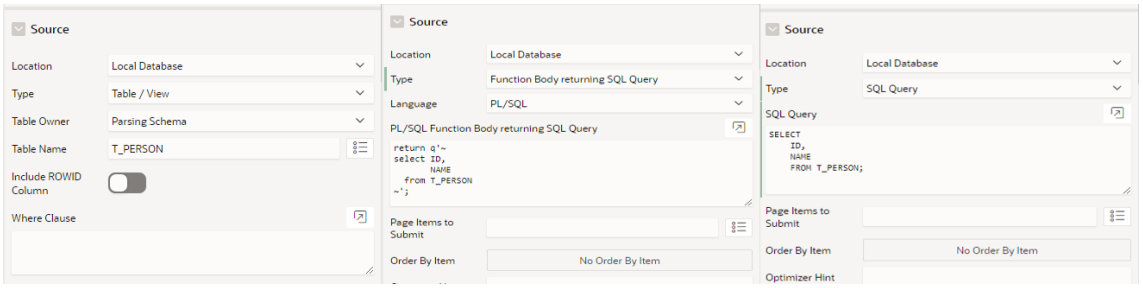


Figure 6.5: Various data sources.

In Figure 6.6 you can see **Layout** where you can specify region location within the page. **Parent Region** is to specify which region would contain our report. You can also adjust this via **Position**. Positions are defined by the specific *page template*. After creating the page, it is built with specific theme and layout depending on chosen template. Default template for theme **Universal** is called **Standard**.

Regions are organized in a *grid* - so, every region has its own location defined by row and column. These regions are organized by its **sequence number**. First region is in first row and first column of the grid. Every next region is then organized by one of these patterns: in same row and column as previous region, and it is displayed under the previous region. Attributes **Start New Row** and **New Column** are used for specifying the way of organization for these regions.

Size and *location* are calculated after assignment of row and column. This is accomplished by calculating occupation and width for every part of the row by values of parameters **Column** and **Column Span**.

Indentation of area in its row is defined by attribute **Column**. Value of this column is in range **1 - 12** which refers to **1** of **12 points** of the grid which split the width. Value **1** stands for the most *left*

part and **7** is for *middle* of the page.

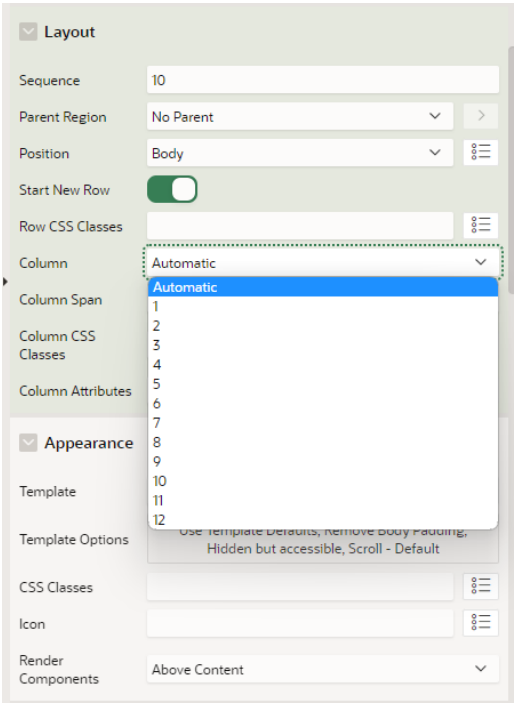


Figure 6.6: Layout.

In Figure 6.7 you can see the **Appearance** section which is used to edit the template for specified region. Figure 6.8 shows pre-defined templates which the user can select from. Every template can be customized by clicking on template options or by writing your own **CSS** statements. In template options you can define multiple rules like *body height*, *header visibility* or even *distance between items*.

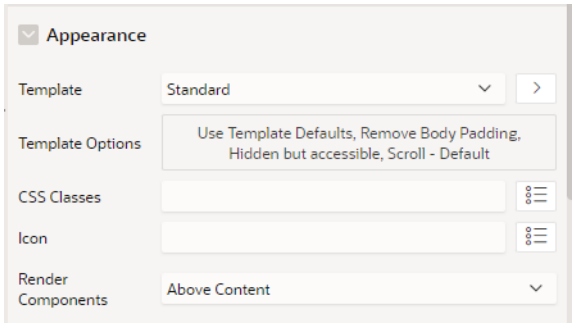


Figure 6.7: Appearance.

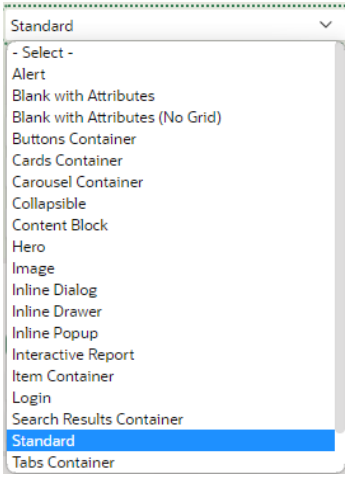


Figure 6.8: Templates.

Figure 6.9 shows a section named **Advanced**, where you can choose from four options. **Static ID** is dedicated for setting up identifier of report. This **Static ID** is used as **ID** for **HTML** element within the page for situations when user wants to configure the page appearance using **CSS**. **Custom Attributes** are for additional information for **HTML** element. Custom attributes are stored as name/value, for example: *name* = "*MY_REPORT*"

Region Display Selector is a component of region which provides navigation controls for other regions within the page. **Exclude Title from Translation** avoids the title from translation in case that page has defined translations.

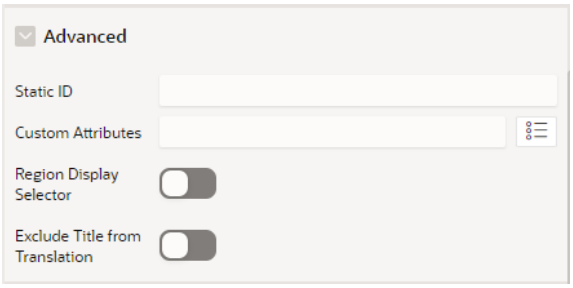


Figure 6.9: Advanced.

Header and Footer shown in Figure 6.10 are used for editing content of these parts of a report. It is important to mention that the footer and header are related to the report and not to the entire page.

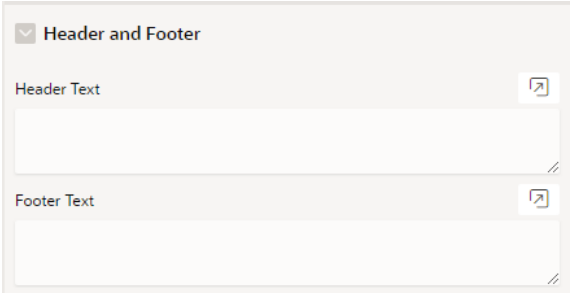


Figure 6.10: Header and Footer.

Types of **Server-side Conditions** are listed in Figure 6.11. In an application developed by Oracle APEX, **Server-side Conditions** are resolved in initialization phase of website loading. The report is not displayed if server-side conditions are not met. There is a variety of types of conditions. To give an example you can compare the values of items or there are conditions for checking requests.

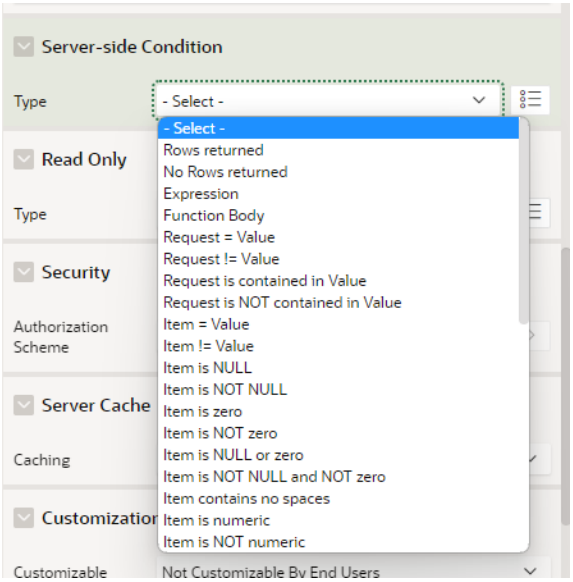


Figure 6.11: Server-side Condition.

The **Read Only** item depicted in Figure 6.12 has the same options as **Server-side Condition** item displayed in Figure 6.11. Naturally, it is not possible to change view of selected region, while it is read-only. In case of basic report, it is not possible to sort column values.

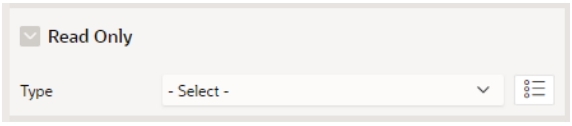


Figure 6.12: Read Only.

In Figure 6.13, you can see last three sections that will be explained. These are **Security, Server Cache**, and **Customization**. The **Security** section contains options for restrictions in terms of displaying the website - e.g. *public website, only for authorized users*. **Server Cache** section includes settings about cache management. In **Customization** window you can adjust access to modify report.

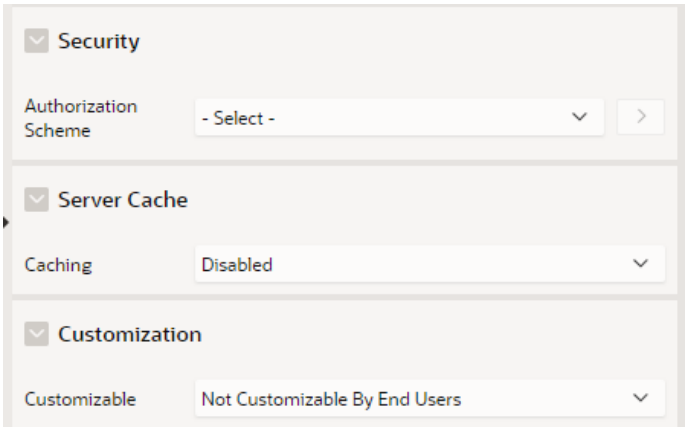


Figure 6.13: Security, Server Cache and Customization.

Another group of sections is **Attributes**. Here, you can set up things *like number of displayed rows*, if it is *possible to list over report results* and in *which way you will list* throughout the report.

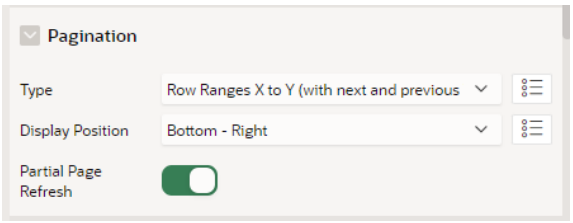


Figure 6.14: Pagination.

Another interesting feature is to set maximal number of loaded rows. You can also define what to display when there is *no data found.* It is possible to download your reports as **.csv** file or print them. Everything mentioned in this paragraph can be found in **Attributes**.

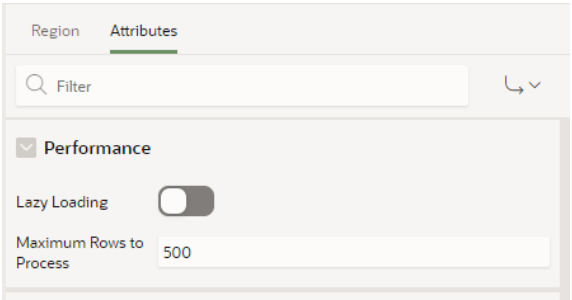


Figure 6.15: Number of rows to load.

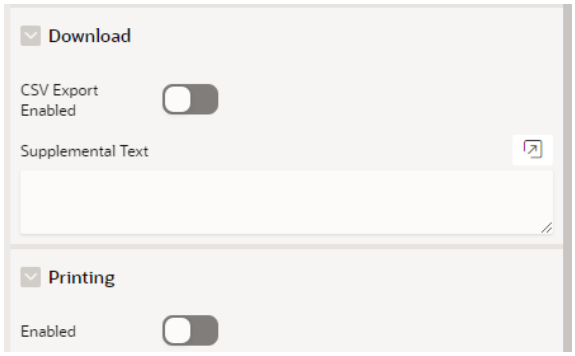


Figure 6.16: Download and Printing.

Very important part is the *center* of the application. In Figure 6.17 you can see the **Layout** which shows us how all components will be organized on our page. Entire page is structured into the grid, so it is more intuitive to imagine where components will be placed. To be user friendly as much as possible, you can select region/item and *drag & drop* it to the desired place. For each region exists its default template, which is provided when you put it in. All necessary settings are already set up.

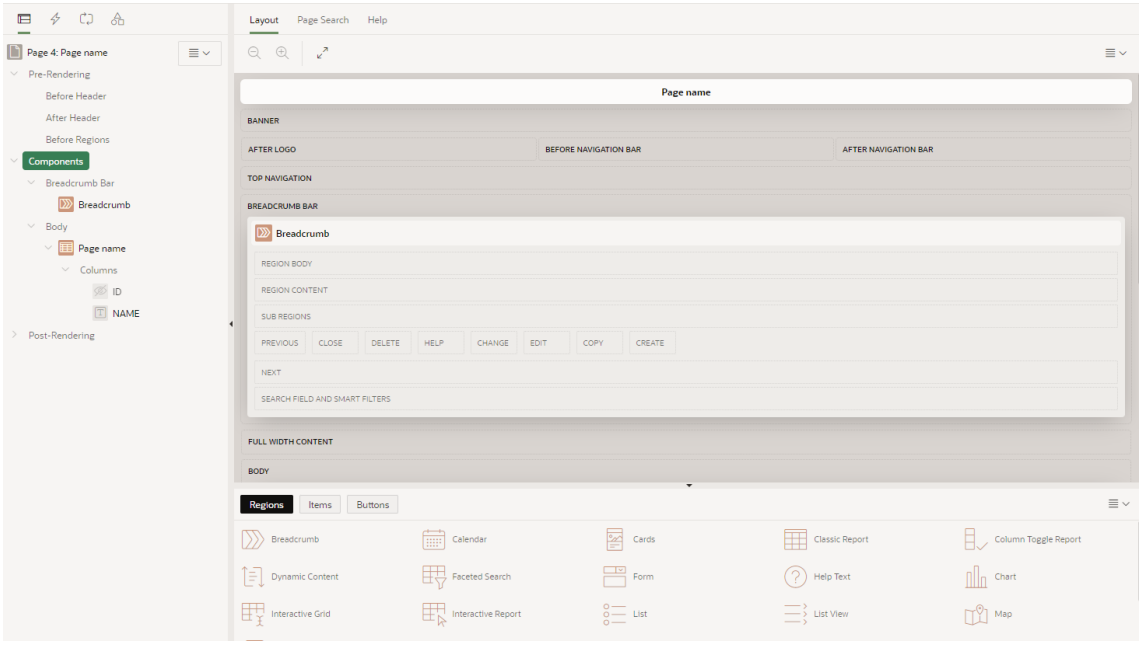


Figure 6.17: Center of page designer.

As you can see on the left side of the page in Figure 6.17, APEX creates some components for report by default. Each component represents one column, and it is possible to alter it in the same way as you do with report.

Column settings are shown in Figure 6.19. In the **Identification** you can see two features - **Type** and **Name of Column**. It is unable to change **Name of Column**, because its value is obtained from the *SQL query*. **Type of Column** shown in Figure 6.18 determines how the column will be displayed - to point out some of them: *Hidden* (nothing is displayed), *Plain Text*, *Display Image, Percent Graph* and so on. By default, all columns are marked as *Plain Text*.

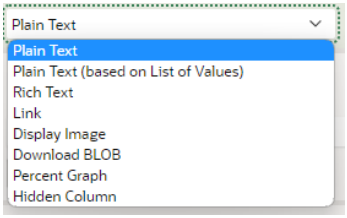


Figure 6.18: Type of column.

Heading consists of two options about header. Contrary to column name, the **Heading** (name) can be edited. Header can also contain **HTML** code. The **Alignment** option speaks about itself. **Layout** section offers **Sequence**, which determines in which order columns will be provided and also **Alignment** option, same as in the previous section.

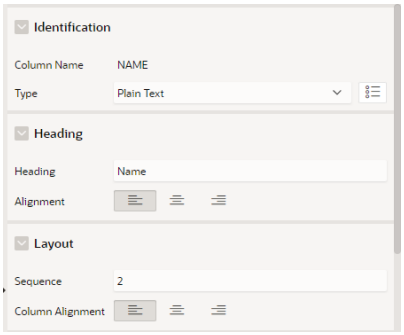


Figure 6.19: Column settings.

Last crucial part is **Appearance**, which should be edited, when you want to treat a number as a string in some *special format*. As you can see in Figure 6.20, you can choose from plenty of basic formats when you click on **Format Mask**. These formats are offered by APEX itself, but you can express your own format if you want.



Figure 6.20: Format Mask.

6.2 Classic Report

A **Classic Report** is a type of report that allows only a basic display of data, without possibility for users to modify displaying options. The only thing that is available for the user is sorting data by individual columns, this also needs to be set.

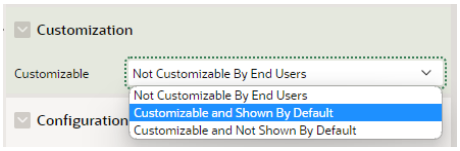


Figure 6.21: Customization.

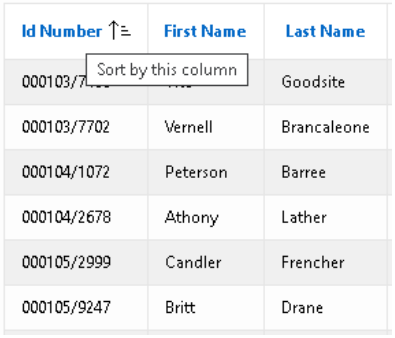


Figure 6.22: Option for sorting by column.

If you want to see how the page will look like, you can click on the green arrow in the circle. As you can see in Figure 6.23, the arrow is located in the upper right corner. In Figure 6.24 we can see how the classic report is displayed. It has columns with names and data. No interaction, except sorting by columns is allowed for user.

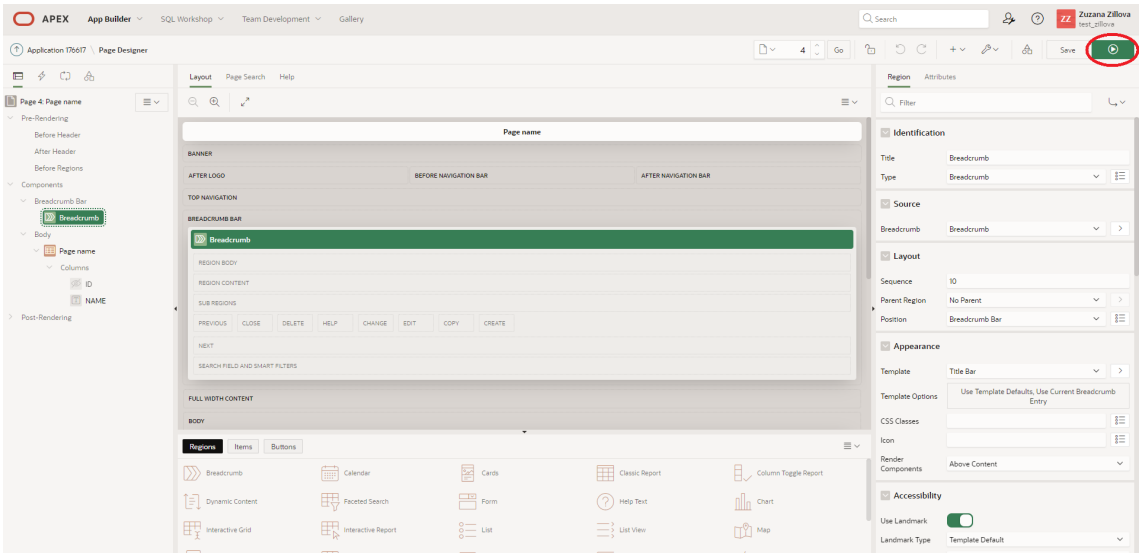
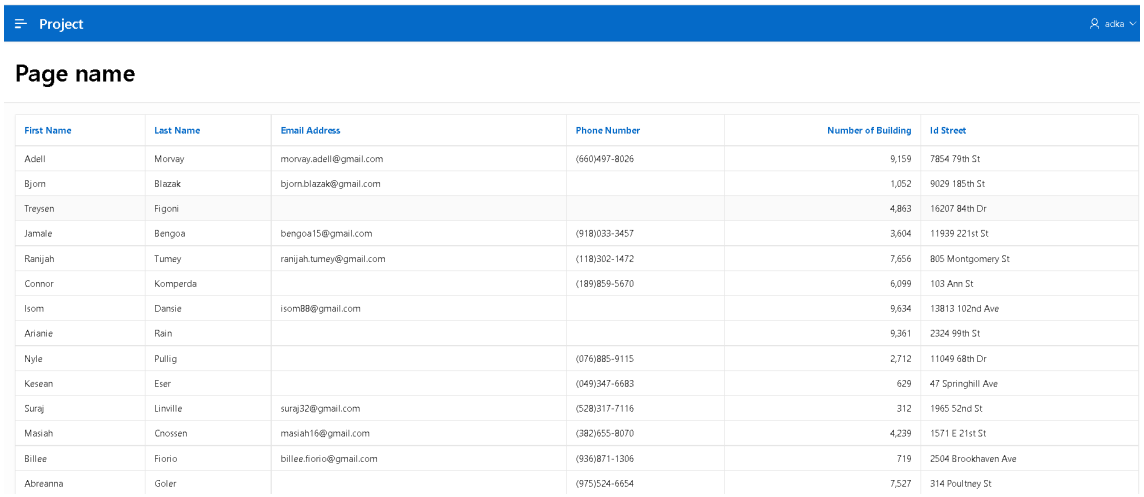


Figure 6.23: Location of green arrow for page showcase.

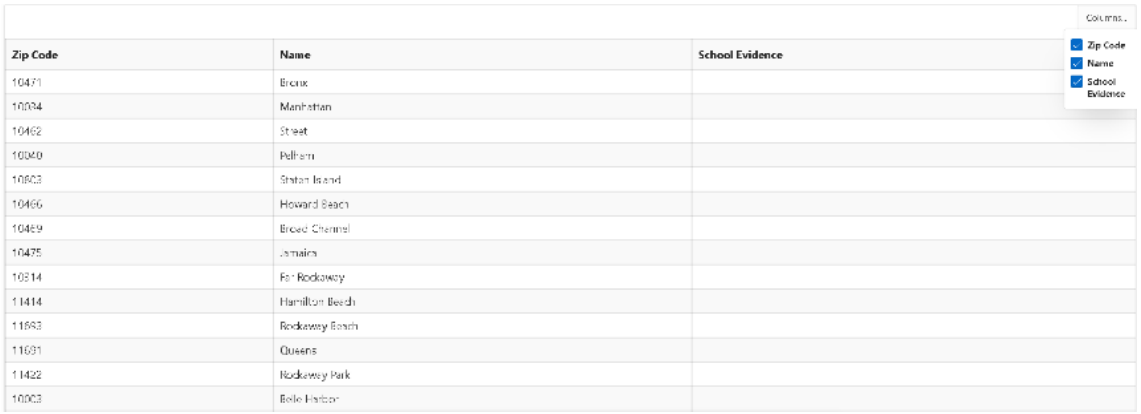


First Name	Last Name	Email Address	Phone Number	Number of Building	Id Street
Adell	Morvay	morvay.adell@gmail.com	(660)497-8026	9,159	7854 79th St
Bjorn	Blazak	bjorn.blazak@gmail.com		1,052	9029 185th St
Treysen	Figoni			4,863	16207 84th Dr
Jamale	Bengoa	bengoa15@gmail.com	(918)033-3457	3,604	11939 221st St
Ranijah	Tumey	ranijah.tumey@gmail.com	(118)302-1472	7,656	805 Montgomery St
Connor	Komperda		(189)859-5670	6,099	103 Ann St
Isom	Dansie	isom88@gmail.com		9,634	13813 102nd Ave
Arianie	Rain			9,361	2324 99th St
Nyle	Pullig		(076)885-9115	2,712	11049 68th Dr
Kesean	Eser		(049)347-6683	629	47 Springhill Ave
Suraj	Linville	suraj32@gmail.com	(528)317-7116	312	1965 52nd St
Masiah	Cnossen	masiah16@gmail.com	(382)655-8070	4,239	1571 E 21st St
Billee	Fiorio	billee.fiorio@gmail.com	(936)871-1306	719	2504 Brookhaven Ave
Abreanna	Goler		(975)524-6654	7,527	314 Poultney St

Figure 6.24: Classic report example.

6.3 Column Toggle Report

Column Toggle Report is almost the same as a classic report, with only one extra feature to specify which column should be visible and which should not. This is displayed in the right upper corner of *Figure 6.25*.



Zip Code	Name	School Evidence
10471	Bronx	
10004	Manhattan	
10462	Street	
10040	Pelham	
10603	Staten Island	
10466	Howard Beach	
10469	Broad Channel	
10475	Jamaica	
10314	Far Rockaway	
11414	Hamilton Beach	
11693	Rockaway Beach	
11691	Queens	
11422	Rockaway Park	
10003	Belle Harbor	

Figure 6.25: Possibility to change visibility of column in Column Toggle Report.

6.4 Interactive Report

an example of an **Interactive Report** is depicted in Figure 6.26. This allows users to make some customizations of the report whilst viewing the page without having access to the core of the application. An interactive report is created just like any other and does not differ much in appearance. Now we will take a closer look at the individual functions and customizations that the user can adjust in the interactive report.

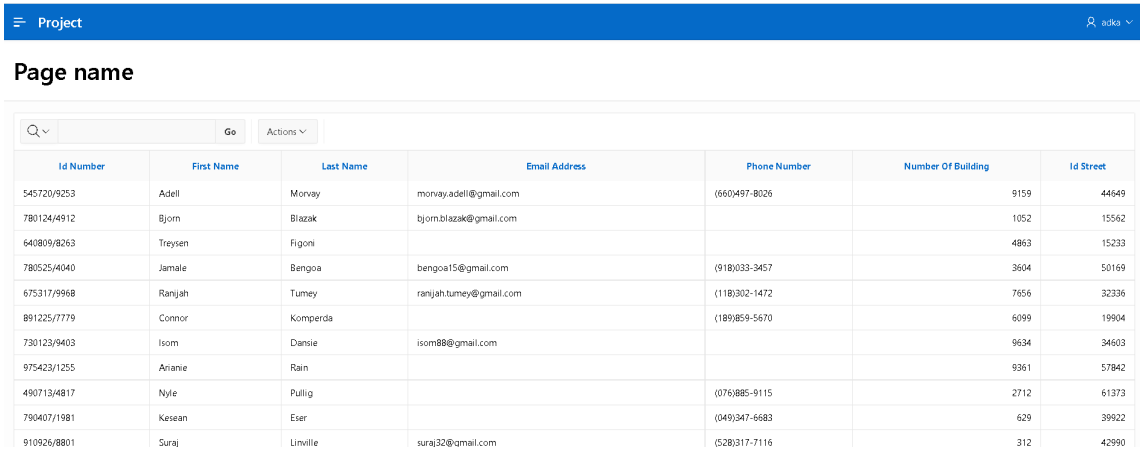


Figure 6.26: Interactive Report.

The search panel shown in Figure 6.27 consists of four items:

- **Magnifying Glass,**
- **Text Box,**
- **Go Button,**
- **Actions button**.

The first three items allow the user to filter report lines. We enter a search string in the field, click on the **Magnifying Glass**, which is where we select which columns should be displayed, and then click on the **Go Button**. We will see those records that contain the search string.

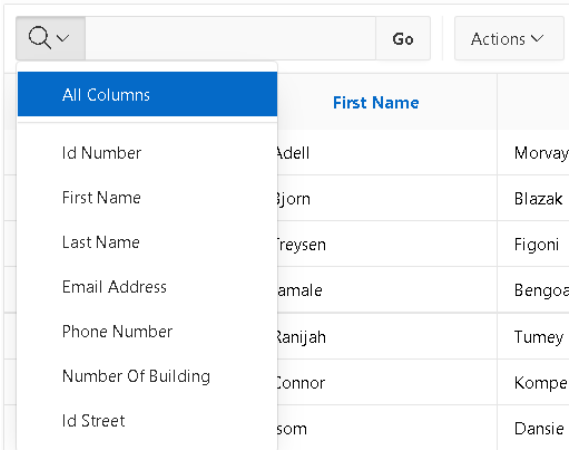


Figure 6.27: Search panel.

For example, we want to search for records that contain the number **"730123/9403"** in any column. This filter will be displayed, and it can be turned off by clicking on the **X Button**. The situation is shown in Figure 6.28. We can have multiple filters active at the same time on different columns.

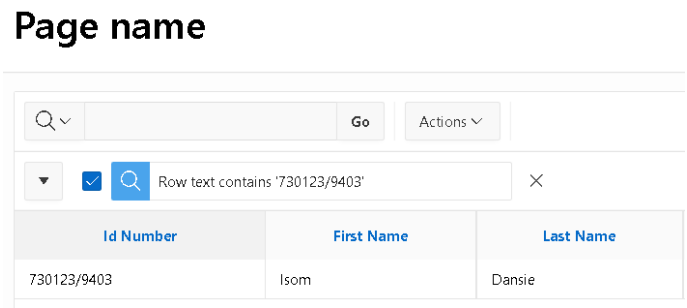


Figure 6.28: Searching.

The **Actions Button** offers more possibilities to adjust display of the report. After clicking, a sub-menu containing several items will appear. The listed items are shown in Figure 6.29.

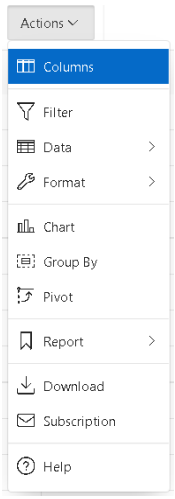


Figure 6.29: Actions Button.

Button **Columns** will show us a box that you can see in Figure 6.30. Here, the user can choose which columns will be displayed in the report and also the order in which they should be displayed. Users can achieve this by moving columns between two sections **Do not Display** and **Display in Report**. There are helpful arrows, that easify work with moving of columns.

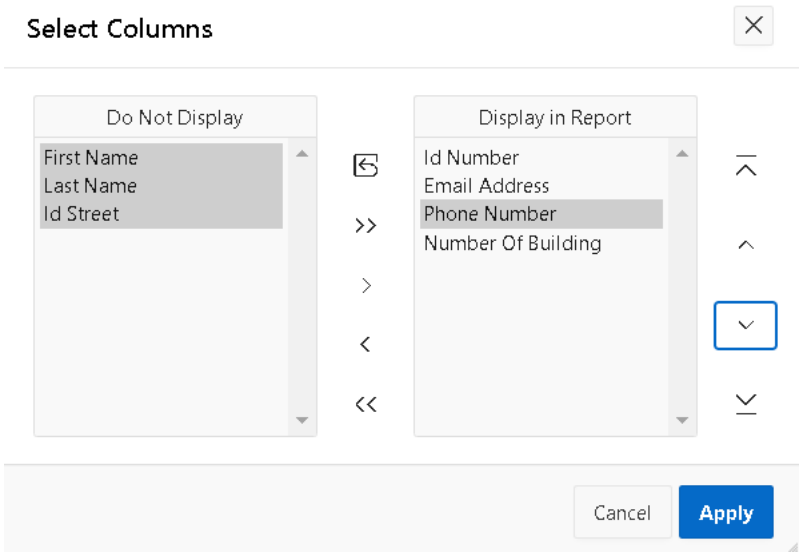


Figure 6.30: Selection of columns to display.

As another option, we have a **Filter** shown in Figure 6.31. We can filter values in column by expressions like *to be equal to, not equal, is null*, etc. You can define expression by choosing from predefined, or by writing your own.

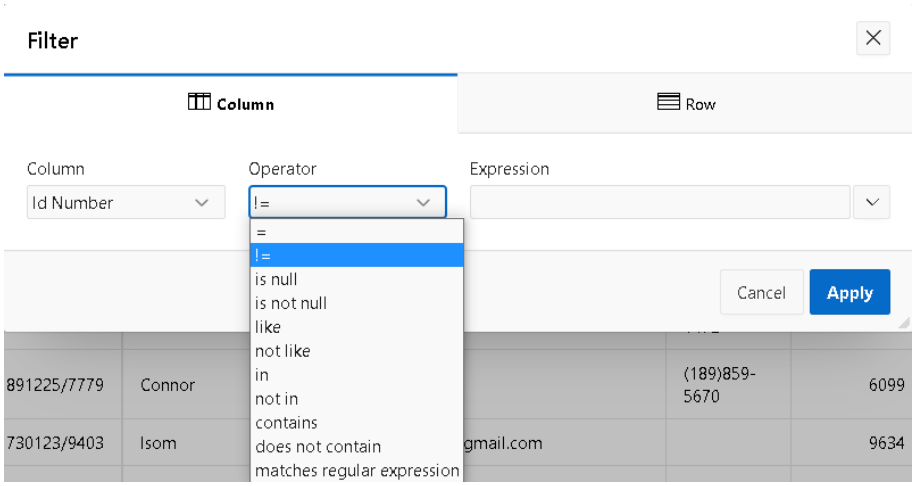


Figure 6.31: Column Filter.

In case of filtering rows, we have a choice of countless operations and expressions that we can compose. This filter appears in the **Filter Expression** window depicted in Figure 6.32.

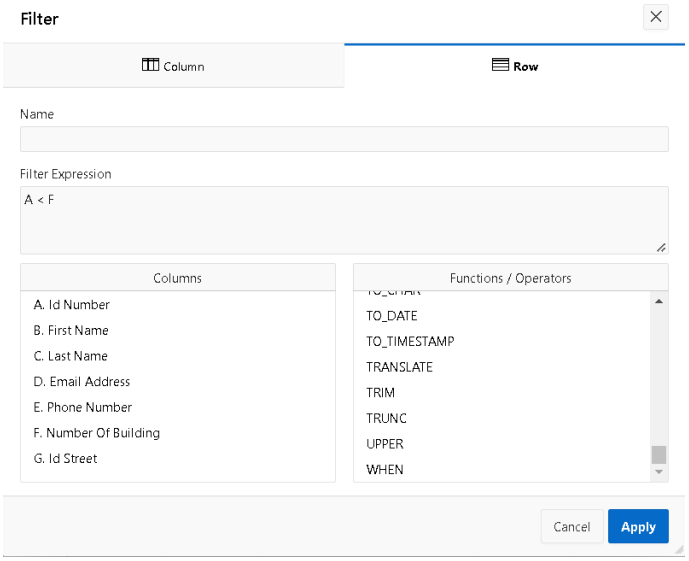


Figure 6.32: Row Filter.

After clicking on the **Data** button, the menu shown in Figure 6.33 will appear. The menu contains another four possibilities:

- sort,
- aggregate,
- compute,
- flashback.

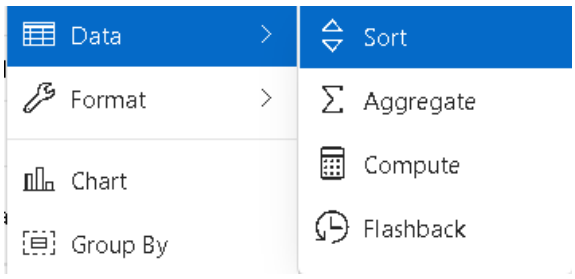


Figure 6.33: After clicking on Data button.

In Figure 6.34 you can see **Sort** option which can be applied on each column and user can determine how to work with *null values*.

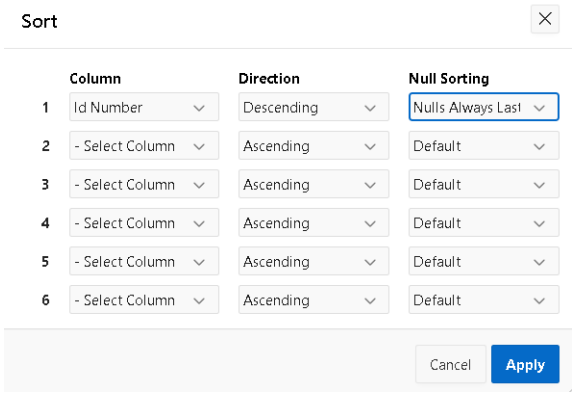


Figure 6.34: Sort.

With **Aggregate** we can perform an aggregation function above one of the columns of our report. The aggregation functions that we can choose from are shown in Figure 6.35.

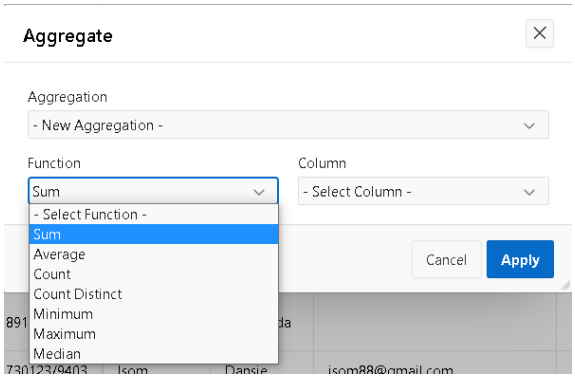


Figure 6.35: Aggregate.

Flashback function depicted in Figure 6.36 allows us to return to state of displayed report before changes a few minutes ago.

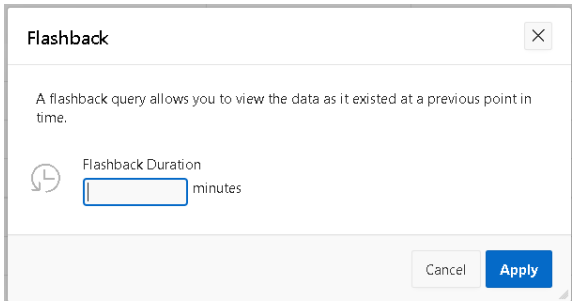


Figure 6.36: Flashback.

Control Break shown in Figure 6.37 allows us to divide the report into several smaller units, as if they were separate reports. These small units are divided depending on column values. Each unit shares the same data in the selected column.

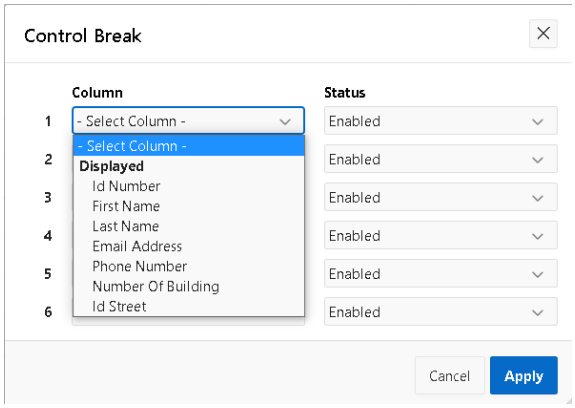


Figure 6.37: Control Break.

Figure 6.38 shows us control break applied on the *First Name* column.

First Name: Aaban

Id Number	Last Name	Email Address	Phone Number	Number Of Building	Id Street
650724/0703	Holmstedt	aaban4@gmail.com	(435)936-9323	4881	61479
640517/1114	Sartor	aaban.sartor@gmail.com		8368	6494
921024/6657	Kovari	aaban.kovari@gmail.com	(268)217-2739	1935	7728
740410/0666	Dantico	aaban.dantico@gmail.com		3115	40604
870703/0560	Labreck	aaban.labreck@gmail.com	(888)203-2292	2639	6159

Figure 6.38: Control Break result.

In Figure 6.39 you can see the **Highlight** function that easily makes the records in the report transparent, without filtering out the others. We can highlight either one cell or the entire row with the colors we choose. Comparing the column value works the same as with the **Filter**, except that unfiltered records do not disappear from the report.

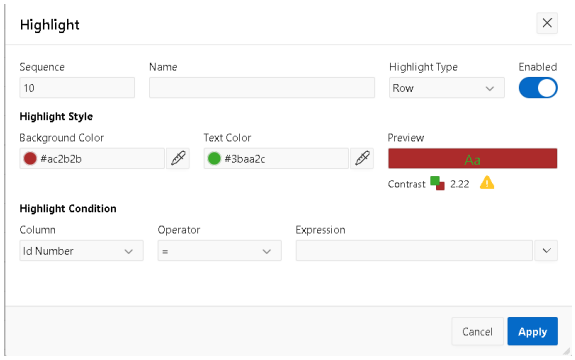


Figure 6.39: Highlight.

In Figure 6.40 we can see the report with highlighted rows, with first name equal to **"Aaban"**.

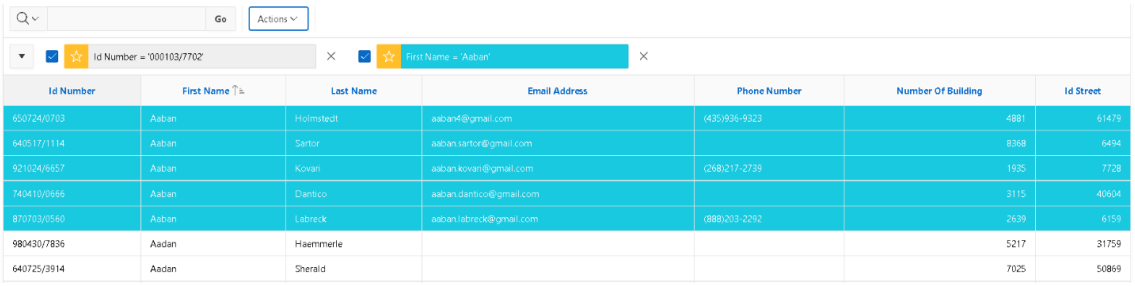


Figure 6.40: Highlight result.

Figure 6.41 show **Rows Per Page** which is an attribute, where the user can choose how many records will be displayed on the page. It's a similar setting to **Pagination**.

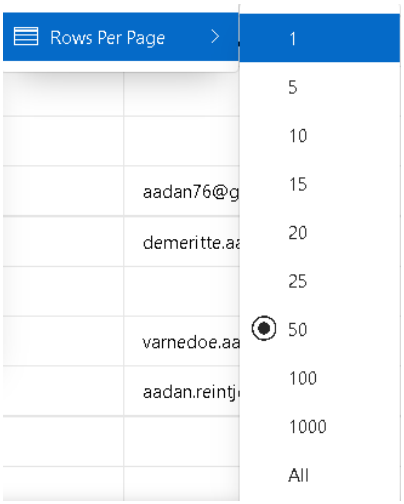


Figure 6.41: Row Per Page.

If the display of data in the report is insufficient, there is also the option of projecting this data into a graph via the **Chart** item depicted in Figure 6.42. There is a choice of **four** basic graphs. We can also choose an aggregation function that is applied to the given column.

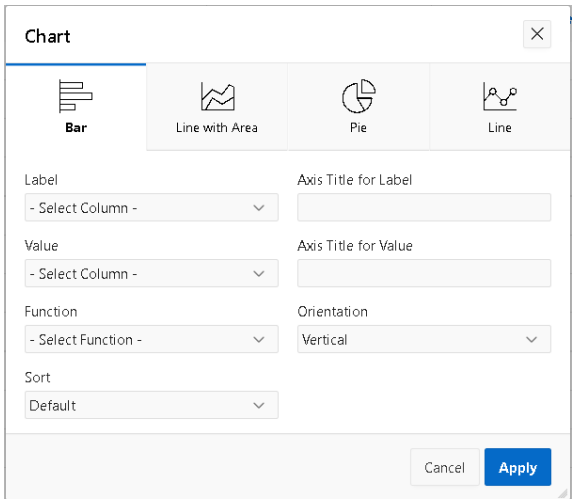


Figure 6.42: Chart in the interactive report.

Also, very important functionality of the interactive report is **Group By** shown in Figure 6.43. It works in the same way as in SQL query.

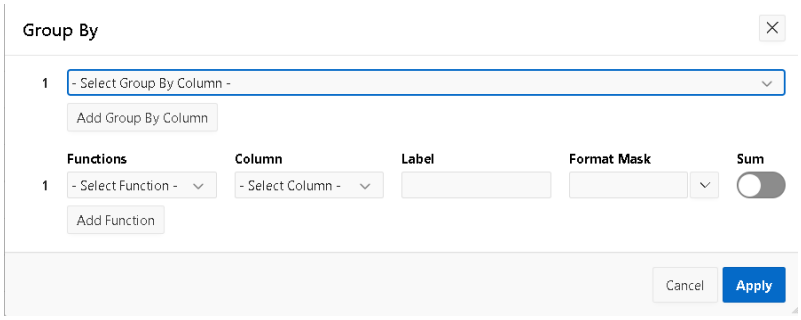


Figure 6.43: Group By.

Save Report depicted in Figure 6.44 saves the report in its current state. However, this report can only be seen by the developer after logging into the application and not by the user.

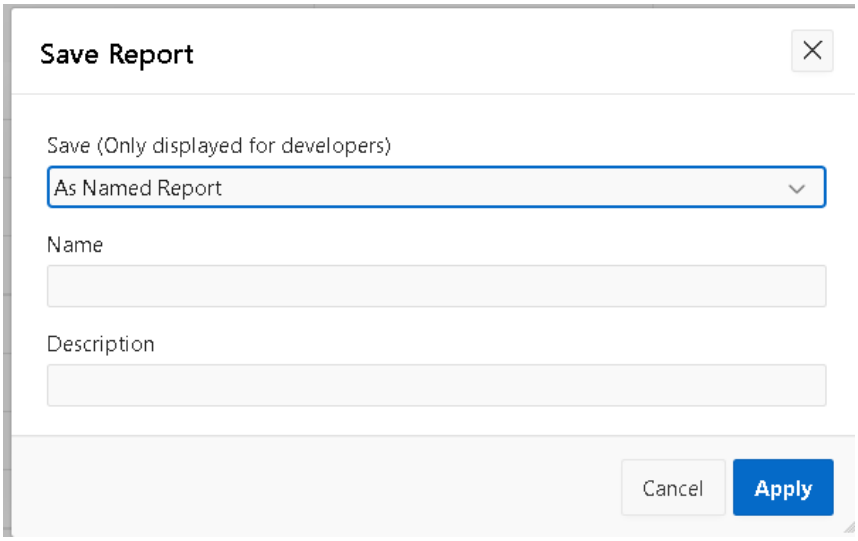


Figure 6.44: Save Report.

We can also reset the report to its original state or download it in the selected format. You can see this option in Figure 6.45.

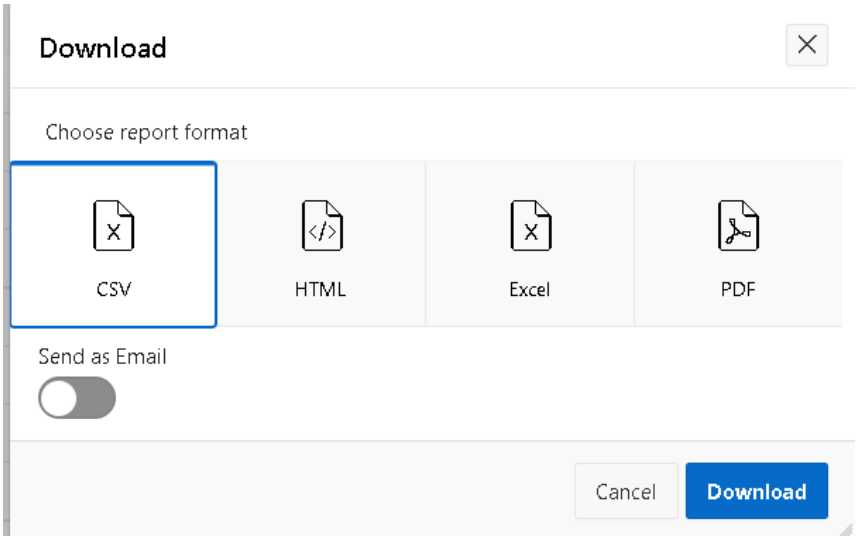


Figure 6.45: Download.

6.5 Questions

1. What category does the report type belong to? Regions, items or buttons?
2. What can be the data source for a Report?
3. What source types can be used for specifying Classic Report?
4. What is the main feature of the Classic Report?
5. Which technology allows user getting the data image as it existed in the past using Interactive Report?

6.6 Answers

1. Report is a region simplifying data browsing and searching.
2. Data can be loaded from local database, REST enabled SQL or REST source.
3. Available source types are Table or SQL query.
4. The main feature of the Classic Report is the read-only mode, disallowing user to modify the displaying options or data.
5. Flashback functions allow user to construct data image as it existed at defined timepoint.

7. How to manage forms?

VERONIKA ŠALGOVÁ, MICHAL KVET, MIROSLAV POTOČÁR

7.1 Types of forms

In Oracle APEX, we can use three basic form types:

- an editable interactive grid,
- a form on a table,
- a master detail form.

7.2 Editable Interactive Grid

In the editable interactive report, can add, modify, delete and search data. It is a feature-rich component with data editing capabilities available directly on the page. In addition, this report can be customized and rearranged interactively using the mouse, directly updating a grid´s structure and contents. It is possible to create an editable interactive grid as a conversion of a read-only interactive grid, which was created by selecting *Report* and then *Interactive Grid*. Another possibility is to create *Report and Form* and select the Report Type to be *Interactive Grid*, which will also be created as a read-only version but can be converted to editable form.

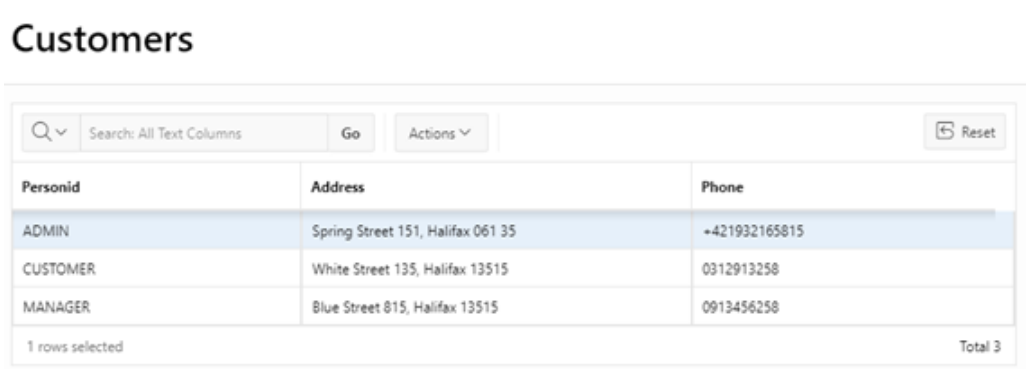


Figure 7.1: Interactive Grid with Customers

Data can be searched using the text in the search bar at the top of the page. Other actions available with this interactive grid are shown in Figure 7.2.

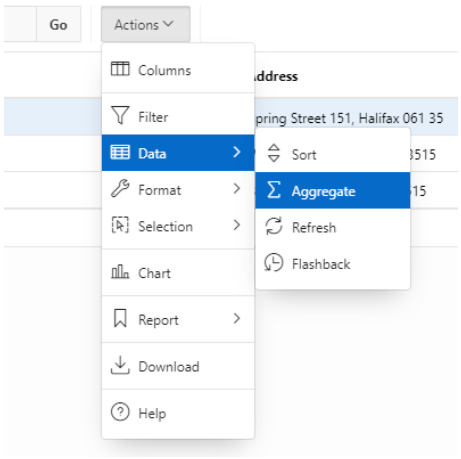


Figure 7.2: Actions of Interactive Grid

It is possible to resize a grid column by clicking and holding the edge of a column heading and adjusting it with the mouse. You can also hide a column by selecting the header and clicking the *Hide* icon, as shown in Figure 7.3.

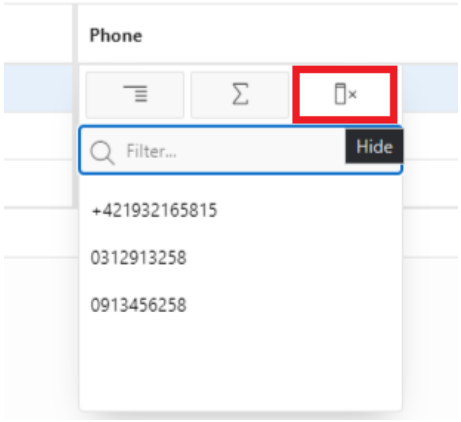


Figure 7.3: Hide icon of the Phone column

After hiding the column, it is no longer displayed in the grid. To view the column again, you have to click on the *Actions*, choose the *Columns* option, and click on the columns to be displayed.

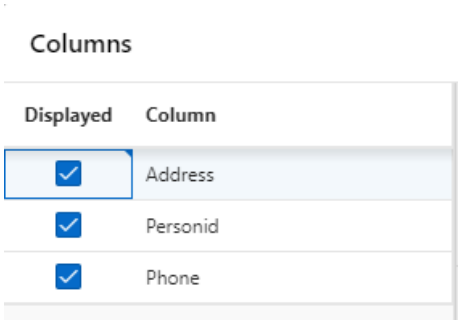


Figure 7.4: Displayed columns

A column can be frozen after clicking the *Freeze* icon, as shown in Figure 7.5. Freezing a column excludes it from the scrollable area.

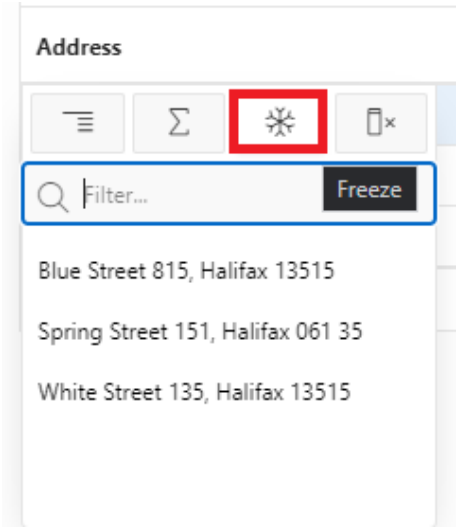


Figure 7.5: Freeze icon of the Address column

Data can be sorted according to a column by selecting the *Sort Ascending* or *Sort Descending* icon, as shown in Figure 7.6.

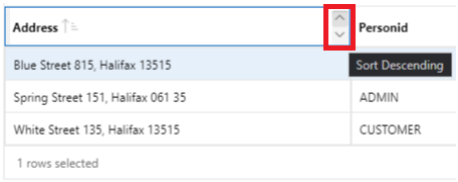


Figure 7.6: Sorting the Address column

Using the *Aggregate* icon of the column, as shown in Figure 7.7, it is possible to choose from Count, Count Distinct, Minimum and Maximum functions on the selected column.

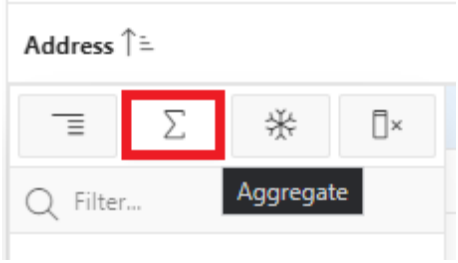


Figure 7.7: Aggregate icon of the Address column

Data from the grid can be used to create charts, choosing *Actions* and then *Charts* options. There are various types of charts, as shown in Figure 7.8. You can choose from a variety of aggregation functions provided with the charts.

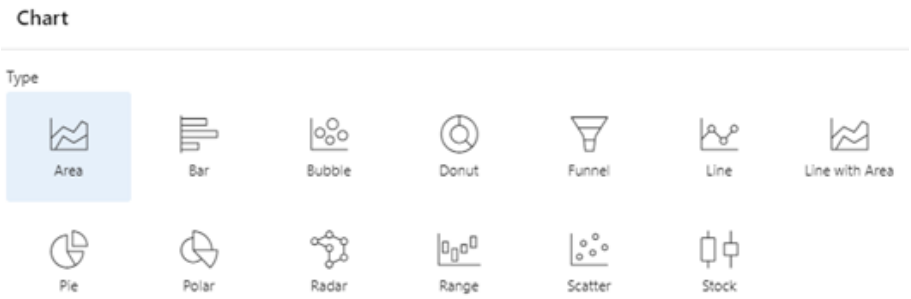


Figure 7.8: Types of charts

7.3 Form on a Table

Oracle APEX provides us with basic forms to insert a single row in a table, as shown in Figure 7.9. After clicking the *Create a Page* button, you need to choose a *Form* option. You need to select the table connected to the created form, or you can write your SQL Query and also a primary key column of the table.



Figure 7.9: Form for inserting customers

7.4 Master Detail Form

It is possible to query, insert, update, and delete values from two related tables or views. After clicking the *Create a Page* button, you need to choose a *Master Detail* option. In the following detail, you choose a master-detail style, such as *Stacked*, *Side by Side*, or *Drill Down*. A master detail form usually displays a master row and multiple detail rows. It means you need to choose a table or a view to being a master data source and another one to be a detail data source. After selecting the primary key of each table, it is necessary to choose the master detail foreign key.

Figure 7.10 shows a *Stacked* master detail form. It contains editable interactive grids. Users can select a row in the master grid to update the detail grids.

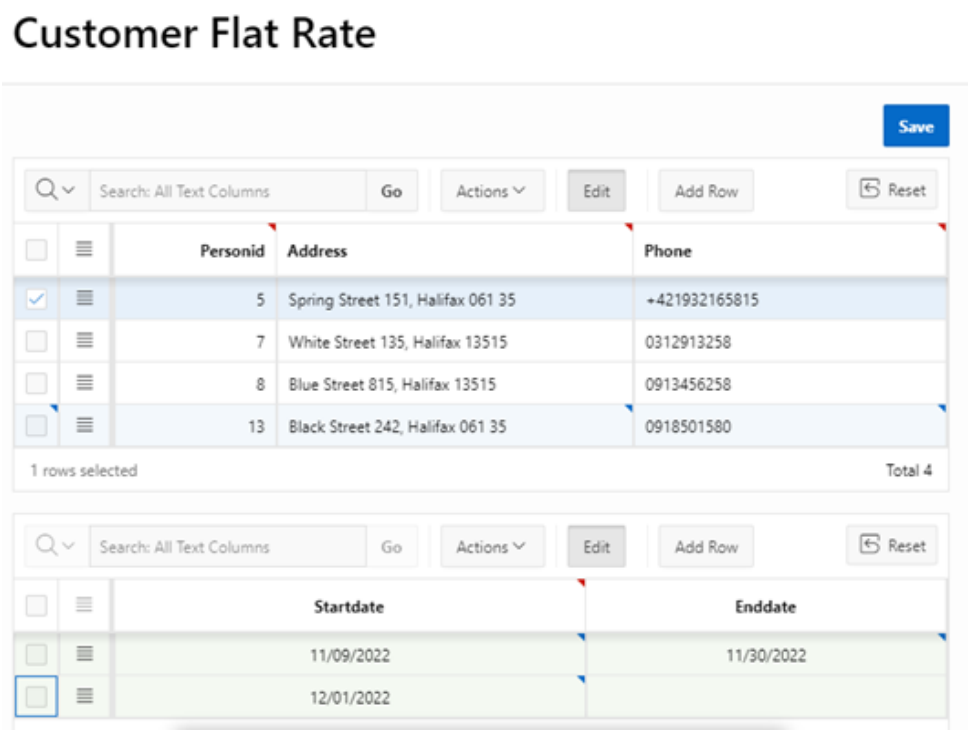


Figure 7.10: Stacked master detail form of Customer Flat Rates

In Figure 7.11, there is a *Side by Side* master detail form containing a single page master detail utilizing a side-by-side layout and report regions with modal edit windows. On the left side, you can choose from a master list to navigate to the master record. The right side contains the selected master record and the associated detail reports.

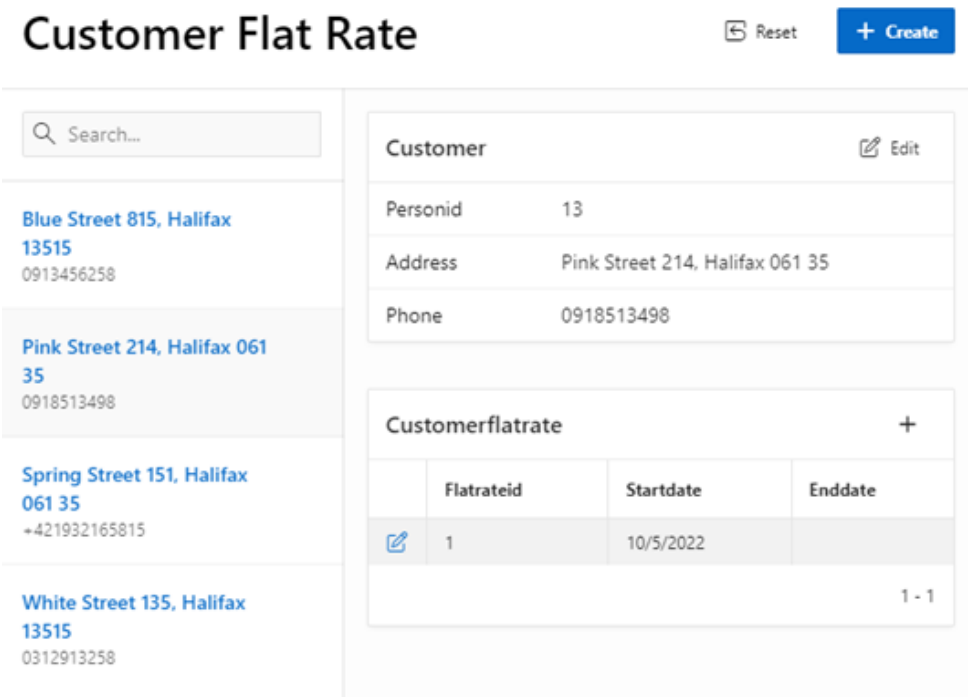


Figure 7.11: Side by Side master detail form of Customer Flat Rates

A *Drill Down* master detail contains two pages. The first page includes an interactive report for the master table, as shown in Figure 7.12. On the second page, as shown in Figure 7.13, there is a standard form for the master and editable interactive grids for the detail.

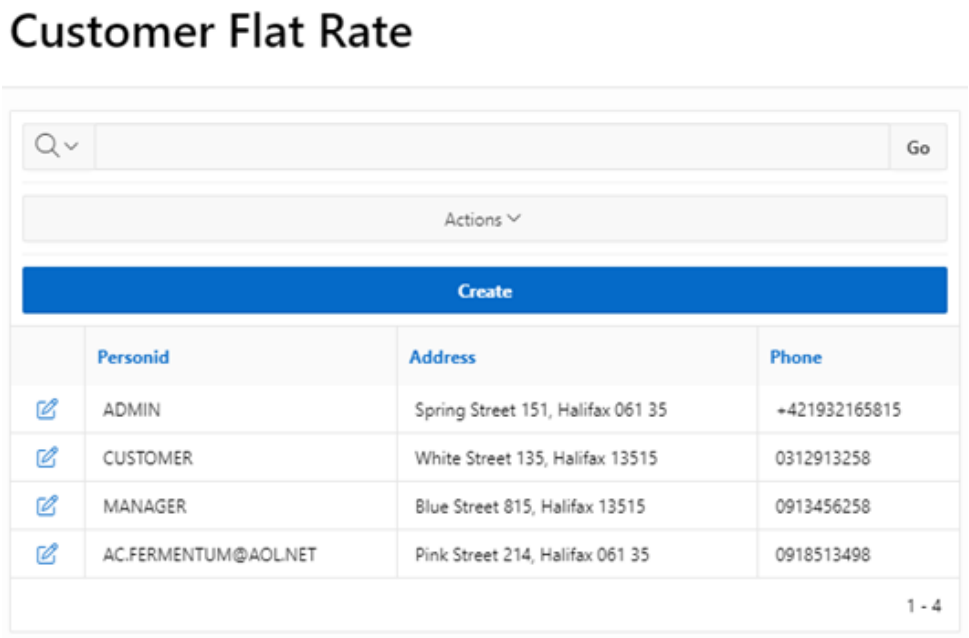


Figure 7.12: Drill Down master detail form – first page

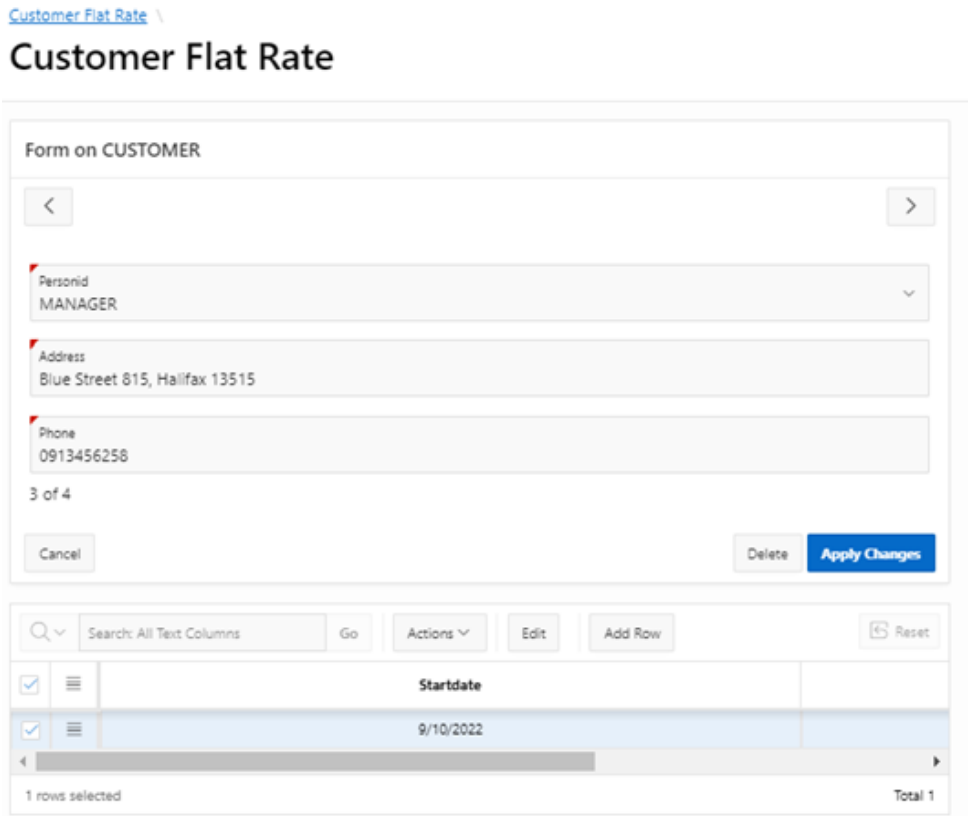


Figure 7.13: Drill Down master detail form – second page

7.5 Questions

1. Is it possible to arrange the data using Editable Interactive Grid?
2. Which table component definition must be selected, when creating a Form?
3. Which option excludes a column from a scrollable area?

7.6 Answers

1. Yes, the reports can be customized and rearranged interactively using the mouse, directly updating a grid´s structure and contents.
2. Primary key attributes must be specified.
3. A column can be frozen after clicking the Freeze icon. Freezing a column excludes it from the scrollable area.

8. How to transform text reports into charts?

IVAN PASTIERIK, MICHAL KVET, MIROSLAV POTOČÁR

8.1 Chart

Chart regions in Oracle APEX are versatile components that empower developers and users alike to gain deeper insights from data through visual representation. These dynamic charts, ranging from bar charts and line charts to pie charts and more, serve as important assets in enhancing the user experience and facilitating data-driven decision-making.

In this chapter, you will learn how to create a simple chart with **filtering** and **sorting** capabilities. **Filtering** and **sorting** can dramatically improve interactivity and the readability of charts. For this purpose, we have created a simple data model as shown in Figure 8.1. The data model consists of two entities: **"product"** and **"sale_transaction"**. The entity **"product"** represents a single product, with its primary key attribute as **"product_id"**, and the product's name stored in the **"product_name"** attribute. The entity **"sale_transaction"** stores information about all our sold products with the primary key **"sale_id"**. The **"product_id"** attribute is a foreign key that references the **"product_id"** attribute in the **"product"** entity. The **"sell_date"** attribute contains the date the product was sold.

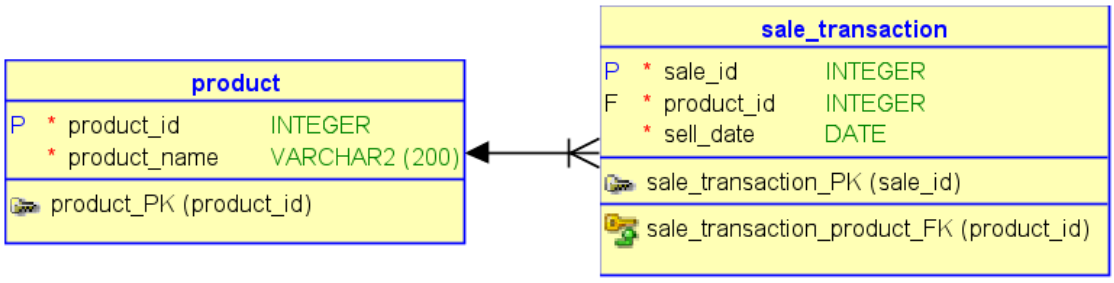


Figure 8.1: Data model used in example application.

To begin, it is necessary to create a new application, which we will call **"Chart Showcase"**. This application will contain the **"Global Page"**, **"Home"** and **"Login Page"**. We will be creating charts on the home page. Detailed instructions for creating applications were described in Chapter 5. After creating the application, enter the **Page Designer** for the **"Home"** page within our newly created application.

8.2 Creating Bar Chart

To place graph region, highlighted in Figure 8.2, into our application, we need to drag and drop it into the layout section of **Page Designer**.

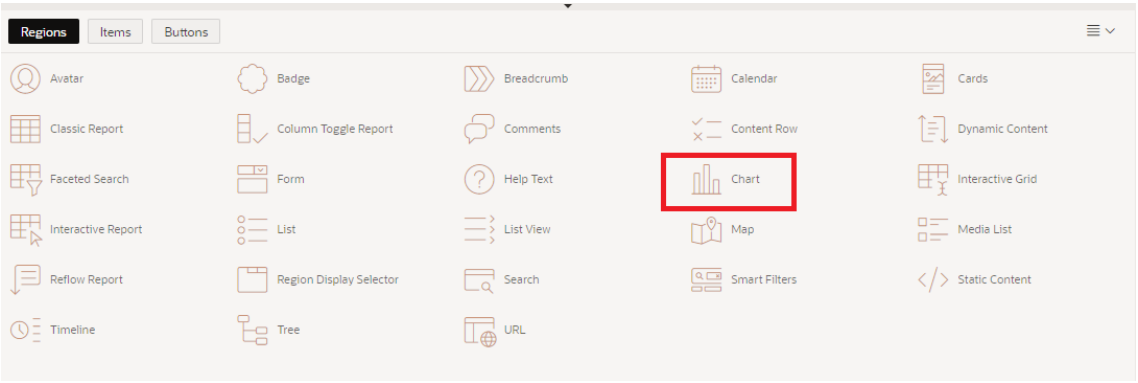


Figure 8.2: Selecting and placing Chart region.

We will place the **Chart** region in the **body** section of our page because we want our graph to be the main content of the page. We will set the **title** of our newly placed Chart region to **"Sales Chart"**. In Figure 8.3, you can see how our layout body section looks after placing and changing the title of our **Chart** region.

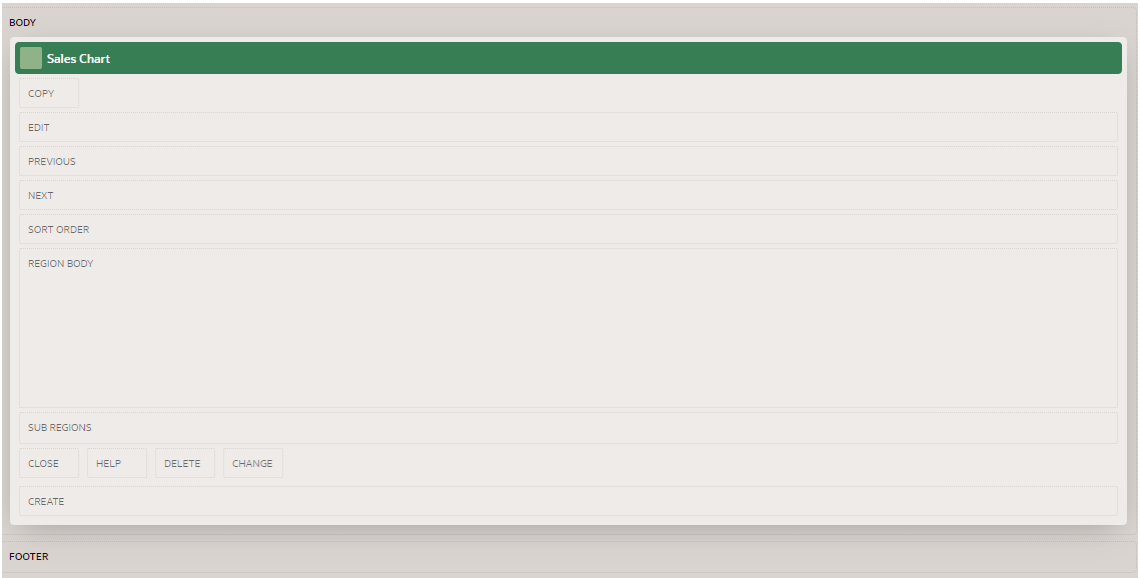


Figure 8.3: Layout body section after placing Chart region.

In the left part of **Page Designer**, you can see the **tree overview** of all of our placed components. If you look closely at our **Sales Chart** component, inside **"Series"**, it shows an error. This error, as seen in Figure 8.4, is caused because we haven't defined the source for our data that we want to display in our graph.

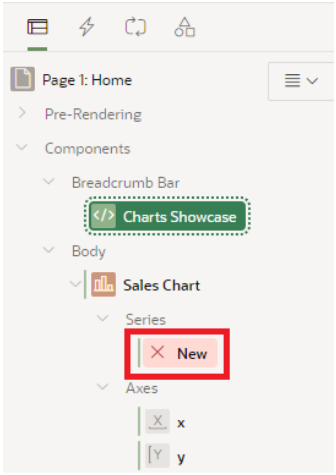


Figure 8.4: Tree overview showing error in chart series.

When you click on the series labelled as **"New"** from the tree overview, the settings menu for the series will open on the right side of **Page Designer**. Inside this settings menu, we will change the name of the series to **"sales"** because we want to show the sales count of our individual products in our **Sales Chart**. Next, we need to define the series **source**. To do that, we will change the source type to **"SQL Query"** and inside the SQL Query, we will place a **SELECT** statement as shown in Figure 8.5. This **SELECT** statement will return the total count of sales grouped by product names. Now that we have defined the source for our data that we want to show in the chart, we need to map the columns selected by the **SELECT** statement to axes **Label (X)** and **Value (Y)**. The final configuration of our series is shown in Figure 8.5 with the important changes highlighted.

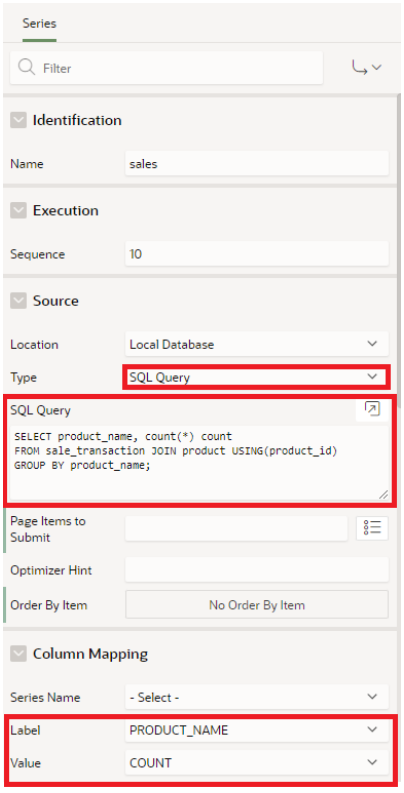


Figure 8.5: Final configuration of sales series

If we take SQL Query from Figure 8.5 and execute it, it returns data as shown in Figure 8.6.

PRODUCT_NAME	COUNT
Smartwatch	68
Monitor	67
Keyboard	66
Drone	67
Virtual Reality Headset	62
Printer	64
Camera	74
Laptop	59
Tablet	65
PC	72

Figure 8.6: Data returned by executing SQL query.

Now, we can save our page and run it. Don't forget to set the page's **authentication** to **"Page Is Public"** so that it doesn't require logging in. When you run the page, you should get results like those shown in Figure 8.7. However, you may notice that our chart doesn't have labelled **X** and **Y** axes, making it harder to read.

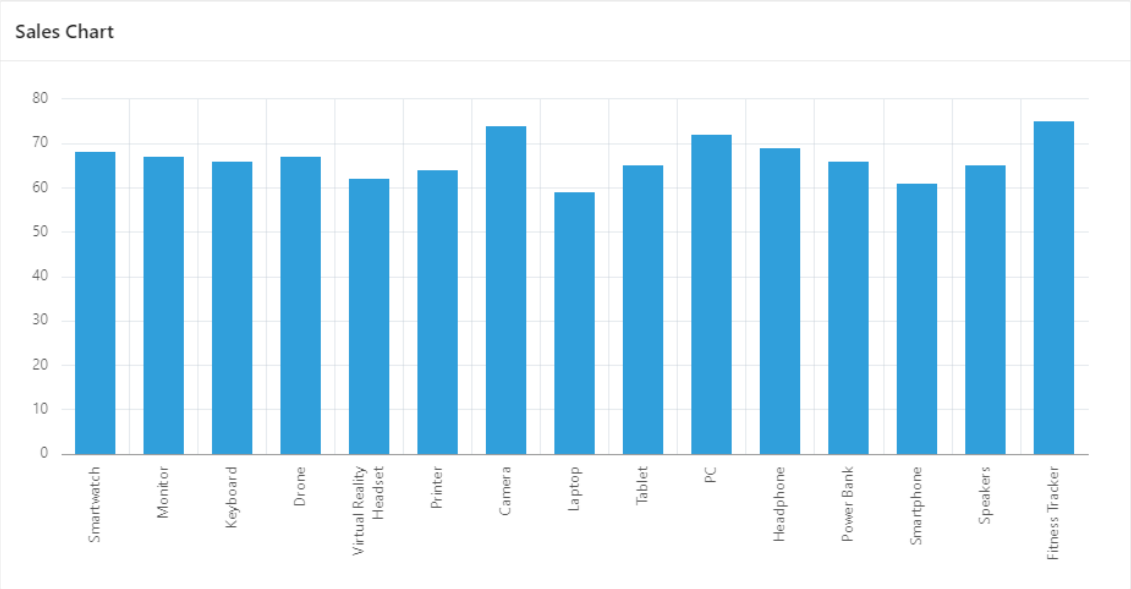


Figure 8.7: Chart shown when launching the application.

To address this, we can label our axes similarly to how we labelled our series. By clicking on individual axes from the left tree overview and changing their **Title** attribute, as shown in Figure 8.8, we can name the **X** axis **"Products"** and the **Y** axis **"Sales Count"**.

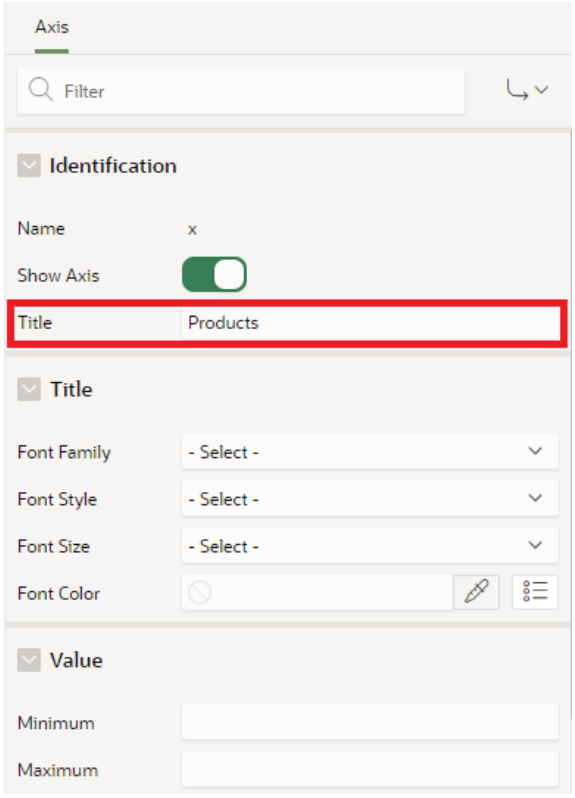


Figure 8.8: Changing title attribute for x axis.

Once we label both axes of our chart, the result should resemble the chart shown in Figure 8.9.

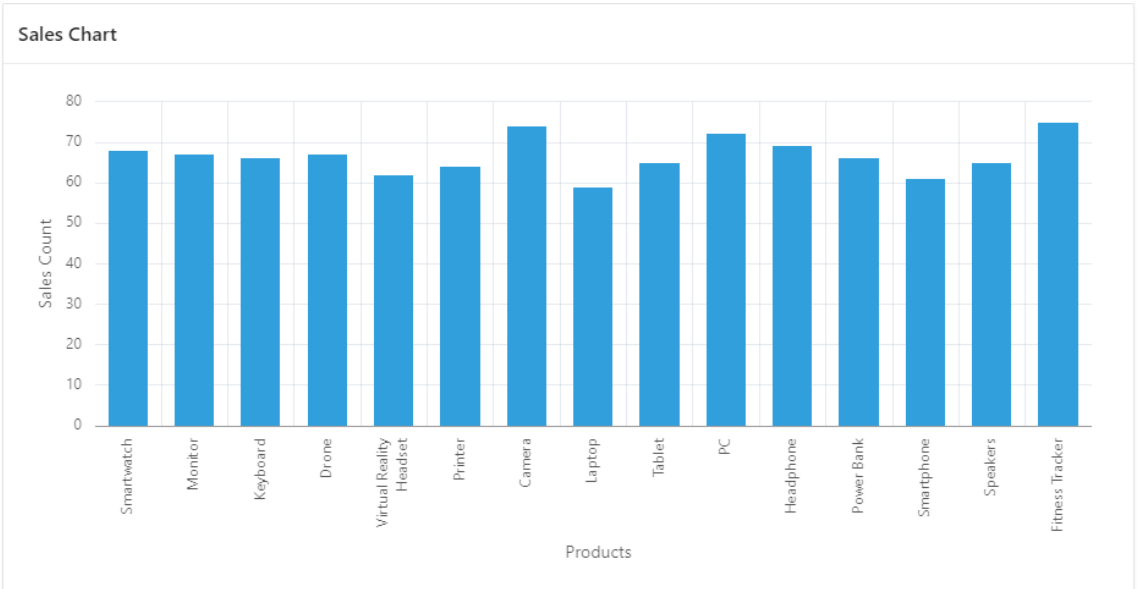


Figure 8.9: Sales Chart after labelling axes.

8.3 Adding Filtering to Bar Chart

Now, let's add **filtering** for products to our chart, allowing us to choose which products to display. To achieve this, we will add a **Checkbox Group** item from the bottom toolbar of **Page Designer**, placing it just above our **Sales Chart** region within the body.

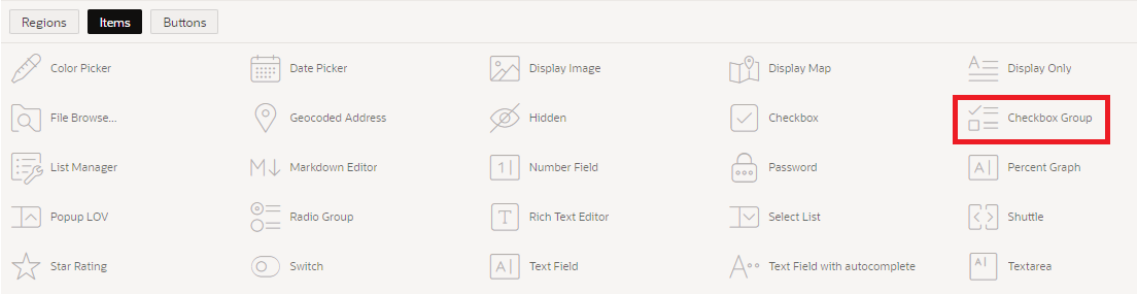


Figure 8.10: Selecting and placing Checkbox Group item.

We will name the **Checkbox Group** item **"P1_PRODUCTS_CHOICE"** and set it's label to **"Products Choice"**. Additionally, we can adjust the layout to display it in three columns, which is shown in Figure 8.11.

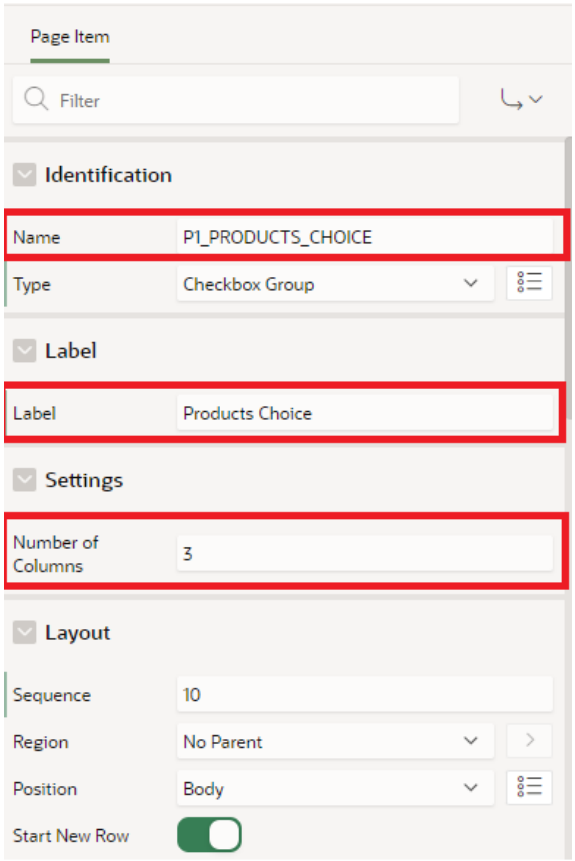


Figure 8.11: Setting basic attributes of Checkbox Group item.

To populate our **Checkbox Group** with products, we need to define a **list of values**. In the **"List of Values"** section of the **Page Item** settings, we will change the **"Type"** attribute to **"SQL Query"** and write a **SELECT** statement as shown in Figure 8.12. This statement should select exactly two columns: the first column containing display values (with an alias **"d"**), and the second column containing return values (with an alias **"r"**).

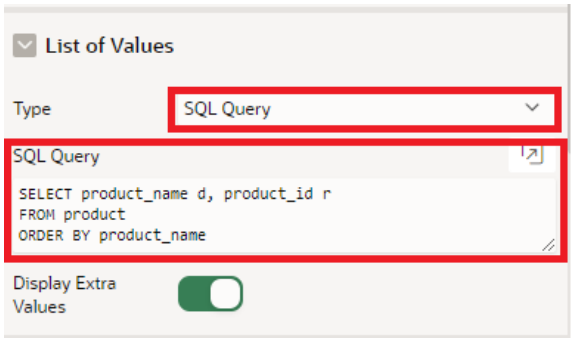


Figure 8.12: Setting list of values for Checkbox Group item.

By default, **Checkbox Group** item typically have all checkboxes unchecked when you run the application. However, for better user-friendliness when used with charts, it's preferable for users to see all products displayed in the chart by default. To achieve this, we will go to the **"Default"** section of the **Checkbox Group** item settings and change the **"Type"** attribute to **"SQL Query returning Colon Delimited List"**. This type of SQL Query allows us to define which values are selected by default. Regular SQL Query cannot be used with **Checkbox Group** item because it can define only one default value, not multiple default values. Inside the **"SQL Query Colon"**, we will write a **SELECT** statement as shown in Figure 8.13. This **SELECT** statement should return exactly one column, which will be internally converted to a **Colon Delimited List**, which might look like this: **"1:5:7:2:3"**.

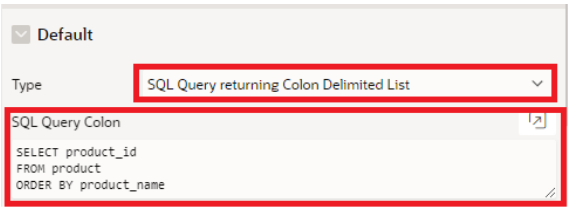


Figure 8.13: Setting default values for Checkbox Group item.

Now, with the **Checkbox Group** item configured to our liking, we can link it to our **Sales Chart** region. To do that, we will add the name of our **Checkbox Group** item to the **"Page Items to Submit"** field in the **"Source"** section of our **sales** series inside the **Sales Chart** region. It is **required** to add every item used in the SQL Query into **"Page Items to Submit"**. The return value of **"P1_PRODUCTS_CHOICE"** is a **Colon Delimited List**, which consists of products that are checked. Therefore, we need to use the **INSTR** function, which returns the index of the first occurrence of a substring defined in the second parameter of the function within the string defined in the first parameter of the function. **INSTR** function will return **0**, if there is no occurrence in string. We will use the **INSTR** function to check if each product's **"product_id"** is present within the selected list of product IDs in the **"P1_PRODUCTS_CHOICE"**. The **INSTR** function searches for the occurrence of **':'||product_id||':'** within the concatenated string **':'||:P1_PRODUCTS_CHOICE||':'**. If it finds a match, it means that the product is among the selected products. We need to concatenate **':'** to both **"product_id"** and **":P1_PRODUCTS_CHOICE"** because we want to perform a precise match between individual product IDs. By doing so, we ensure that, for example, **":1: "** is matched with **":1:2:3:8:9:11: "**. This ensures that we're looking for exact matches, ensuring that each selected product ID is considered independently.

If we were to match **"1"** with **"1:2:3:8:9:11"**, we could potentially display a product with the ID "1" when checkboxes for products with IDs **"1"**, **"11"** or any other IDs containing the value **"1"** are checked. In other words, if we check product with ID **"11"**, not only the selected product would

be displayed, but also product with ID **“1”** would be displayed. This leads to undesired results and inaccurate data representation.

By adding **’:’** to both sides of the IDs, we create a consistent delimiter pattern, making sure that the selected product IDs are compared precisely and eliminating any ambiguity in the matching process. This ensures that the chart displays the data accurately based on the user’s selections.

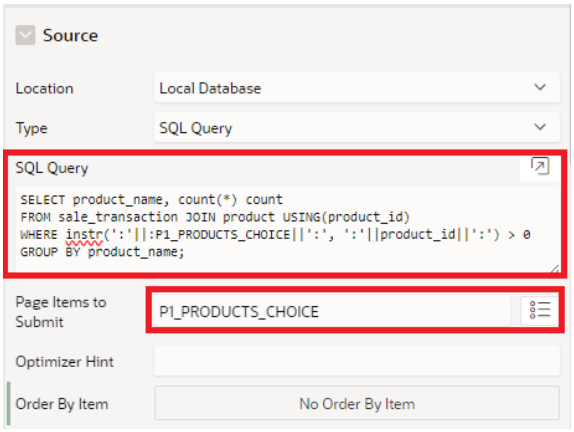


Figure 8.14: Linking Sales Chart region with Checkbox Group item.

The last configuration step is to set up the refresh of our **Sales Chart** region when there’s a change in the selection of the **Checkbox Group** item. To achieve this, we need to create a **Dynamic Action** on our **“P1_PRODUCTS_CHOICE”** Checkbox Group item, as shown in Figure 8.15.

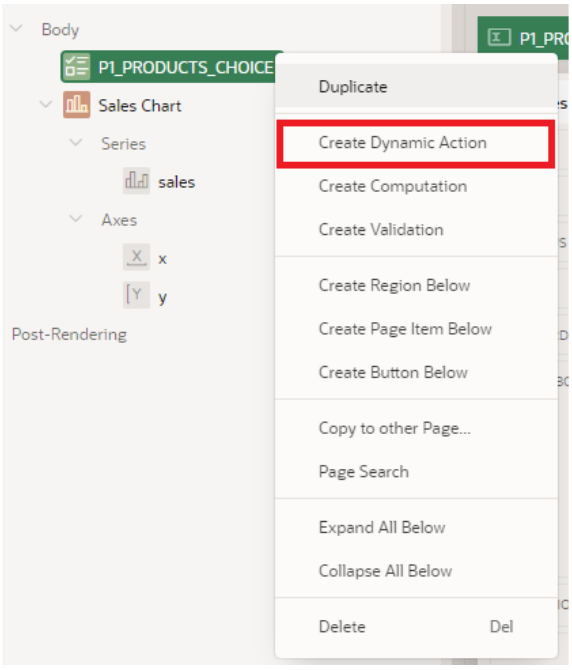


Figure 8.15: Creating Dynamic Action for P1_PRODUCTS_CHOICE item.

We can set the **“Name”** attribute of this **Dynamic Action** to **“PRODUCTS_CHOICE_REFRESH”** as seen in Figure 8.16. We also need to make sure that **“Event”** attribute inside **“When”** section is set to **“Change”** and that **“Selection Type”** attribute is set to **“Item(s)”** and **“Item(s)”** attribute is set to our current item name, so this **Dynamic Action** will be fired when the value of **“P1_PRODUCTS_CHOICE”** changes.

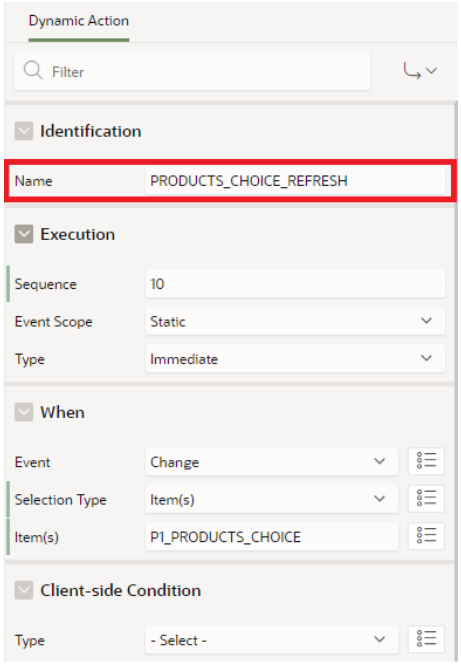


Figure 8.16: Setting basic attributes of on change Dynamic Action.

Within the tree overview on the left side of the **Page Designer**, click on the action that executes when the **Dynamic Action** condition is **true**, which happens when our **Checkbox Group** item changes. When you create a new **Dynamic Action**, you will see the **"Show"** action with an error, so click on it. In the settings of this action, set the **"Action"** attribute inside the **"Identification"** section to **"Refresh"**. In the **"Affected Elements"** section, change the **"Selection Type"** to **"Region"** and then set the **"Region"** to **"Sales Chart"**, as seen in Figure 8.17.

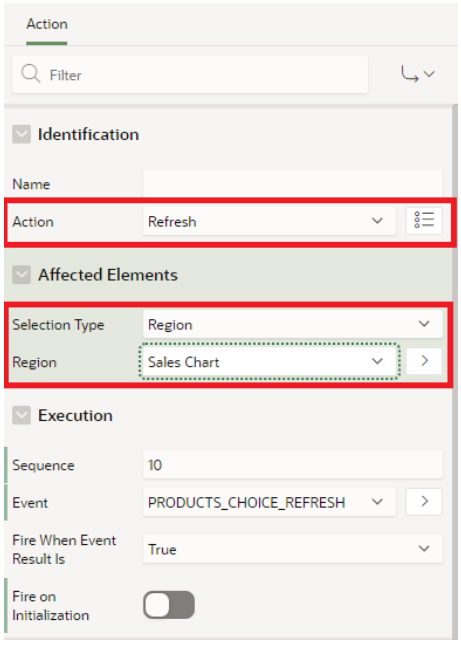


Figure 8.17: Setting attributes of Refresh action.

Figure 8.18 shows how your tree overview in the left part of **Page Designer** should look if you've followed all the steps correctly.

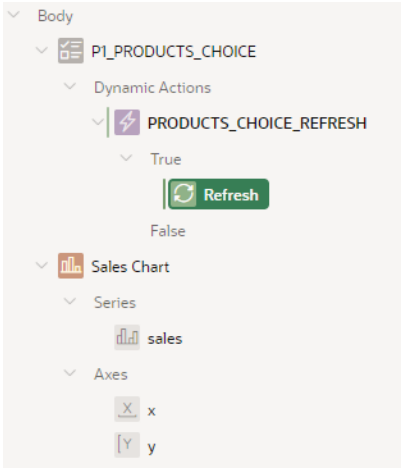


Figure 8.18: Dynamic Action to refresh Sales Chart in tree overview.

When we run and test our application, the **filtering** and **refreshing** of our graph should be fully functional, and the application should resemble the one shown in Figure 8.19.

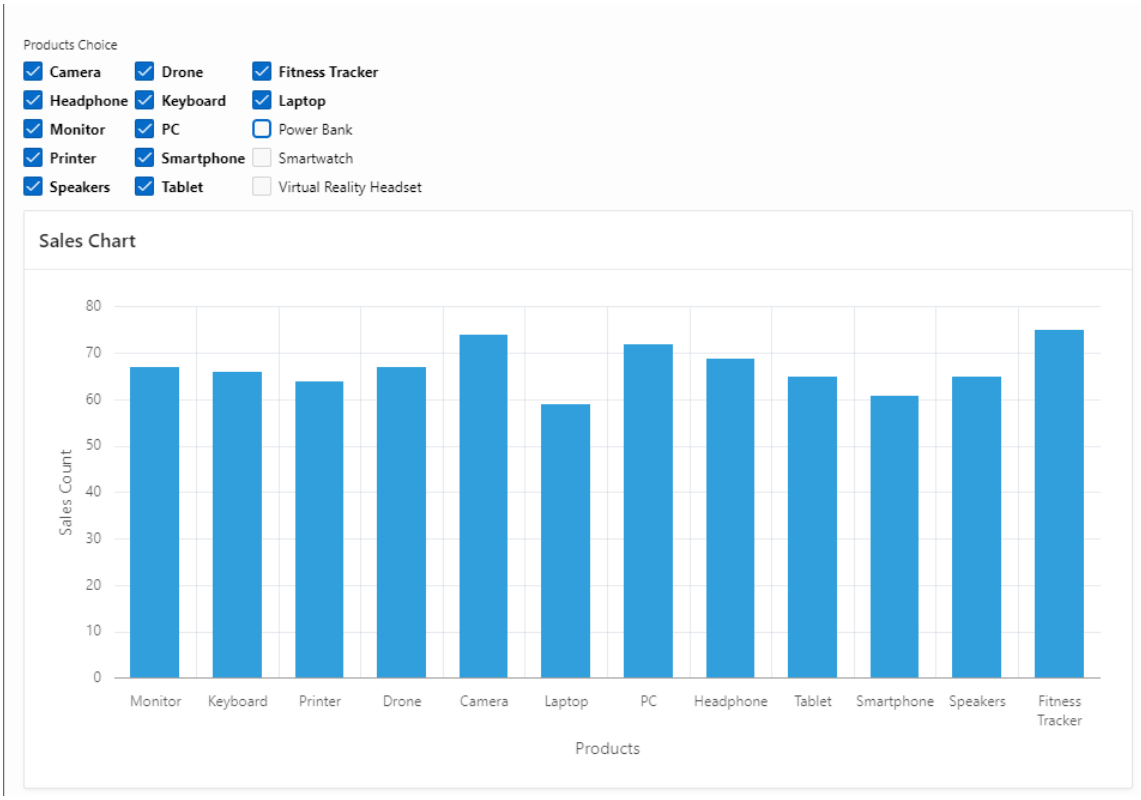


Figure 8.19: Application after implementing product filtering.

8.4 Adding Sorting to Bar Chart

We can enhance the user experience by adding the ability for users to sort values shown in the chart by various criteria. To choose sorting criteria, we will use a **Select List** item, as shown in Figure 8.20.

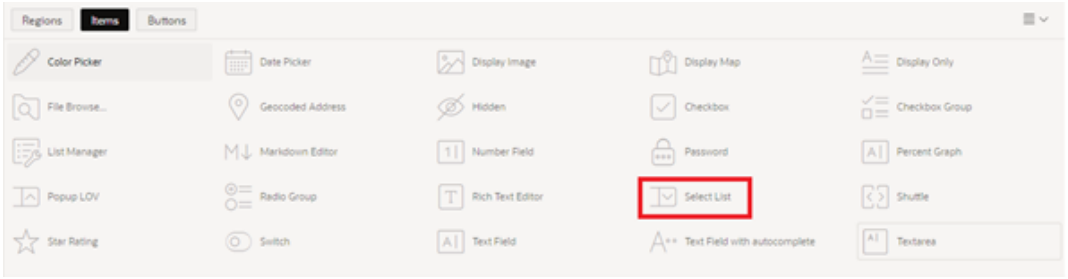


Figure 8.20: Selecting and placing Select List item.

We will place the **Select List** item inside our **Sales Chart** region under the **"Sort Order"** section and name it **"P1_CHART_SORTING"**, as shown in Figure 8.21.

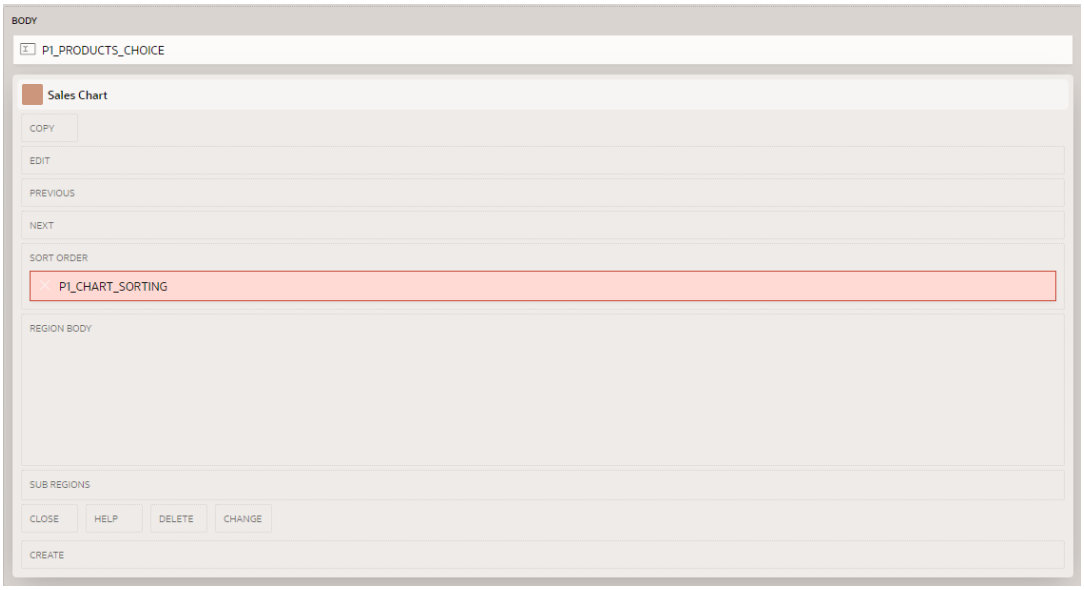


Figure 8.21: Sales Chart region after placing and renaming Select List item.

Now, we need to define a list of values for the Select List item that we've placed. In the settings of the **"P1_CHART_SORTING"** item, we need to change the **"Type"** attribute inside the **"List of Values"** section to **"Static Values"**. Then, we need to press the button showing default values **"Display1, Display2"** right next to the **"Static Values"** attribute.

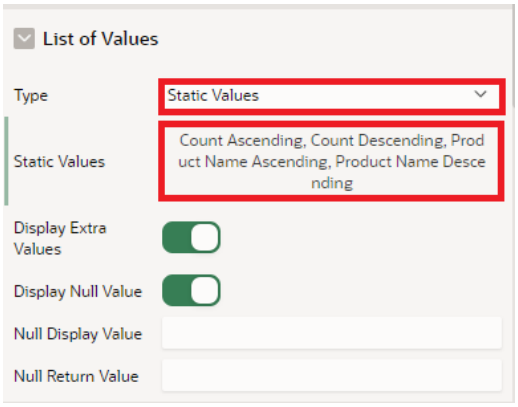


Figure 8.22: Setting list of values for Select List item.

Now you can see that a new window has opened with the title **"Static Values."** Inside this window, we need to define display and return values. Under **display** values, we will write sorting descriptions, and under **return** values, we will write numbers beginning with **1**, as seen in Figure 8.23. If we want these display values to be ordered as we entered them from top to bottom, then we need to disable the **"Sort at Runtime"** option.

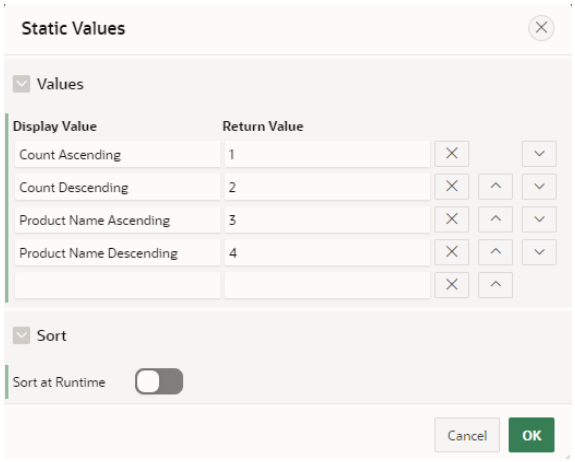


Figure 8.23: Setting static values for Select List item.

We need to add the **Sales Chart** refresh **Dynamic Action** in the same way as we set it up for the **"P1_PRODUCTS_CHOICE"** item. The result of that can be seen in Figure 8.24.

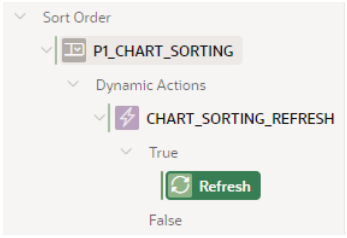


Figure 8.24: Adding refresh Dynamic Action for Select List item.

Now that we have configured the **Select List** item, we need to link it with the **Sales Chart** region. We can do that by pressing the button right next to the **"Order By Item"** attribute, which by default says **"No Order By Item"**, as highlighted in Figure 8.25.

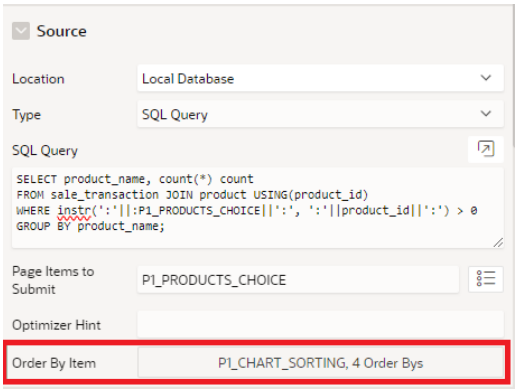


Figure 8.25: Setting Order By Item for Sales Chart region.

After pressing the button, you should see that a new window has opened with the title **"Order By Item."** Inside this window, we first need to set the attribute **"Name"** inside the **"Item"** section to **"P1_CHART_SORTING"**. Then, we need to fill the "Clause" column as seen in Figure 8.26. These clauses are practically **"ORDER BY"** clauses used in **SELECT** statements but without using the **"ORDER BY"** keyword.

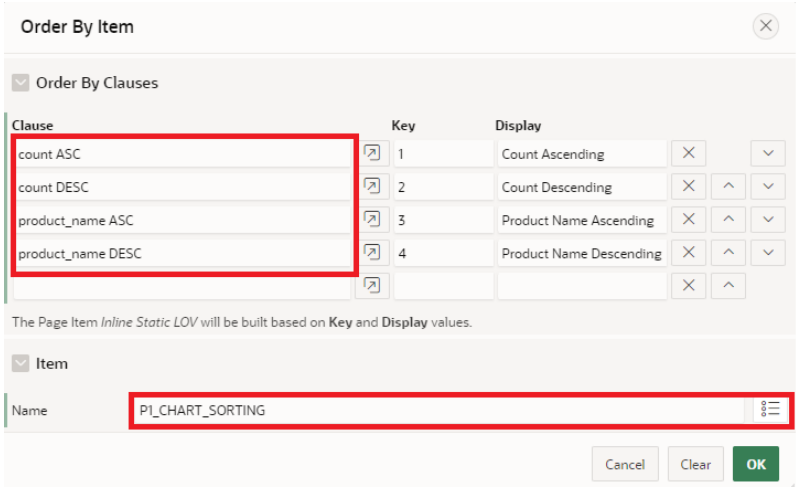


Figure 8.26: Setting Order By clauses for Select List item used for sorting.

If you have followed all these steps closely, chart **sorting** and **refreshing** should be working when you save the page and run the application.

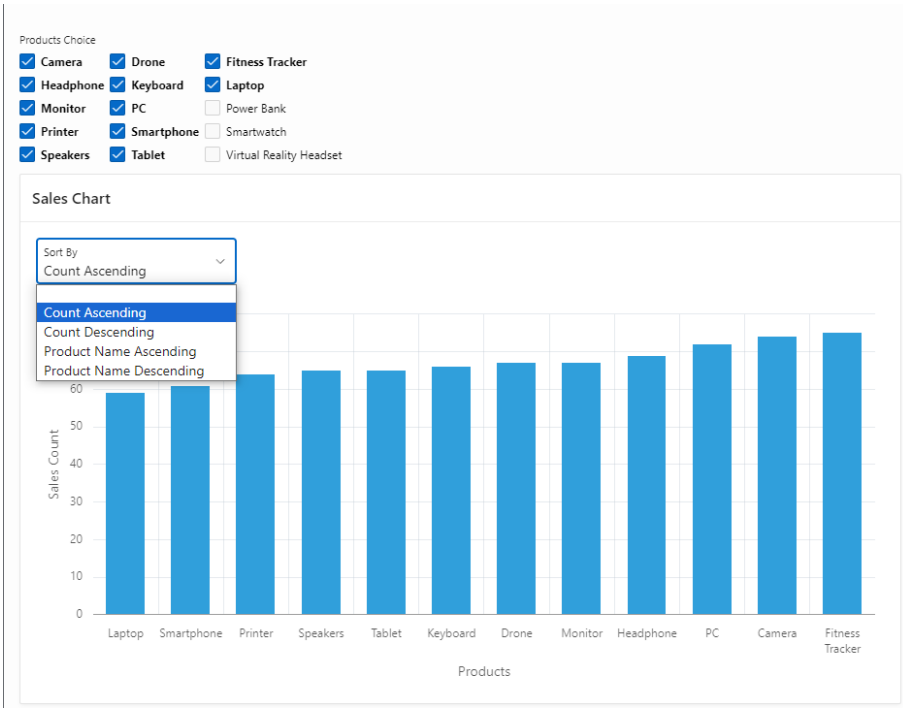


Figure 8.27: Application after implementing chart sorting.

8.5 Creating Different Types of Charts

In case you want to implement a different type of graph, you can do that by selecting the chart region and going into the **"Attributes"** tab on the right side of **Page Designer**, and then changing the value of the **"Type"** attribute. We will create one more **Line with Area** graph showing monthly sales count, which we will name **"Monthly Sales Chart"**, and we will be able to filter it by year.

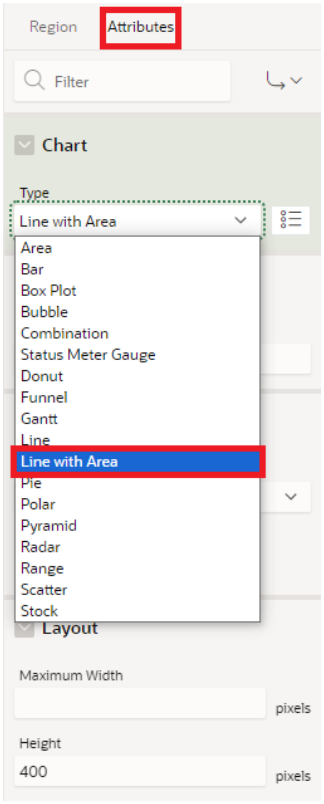


Figure 8.28: Changing the type of graph to Line with Area.

The next step is adding a **Select List** item that we will use to **filter** the chart by **year**. We will place this item inside the **"Monthly Sales Chart"** region under the **"Edit"** section. The SQL Query used for getting the list of values is shown in Figure 8.29.

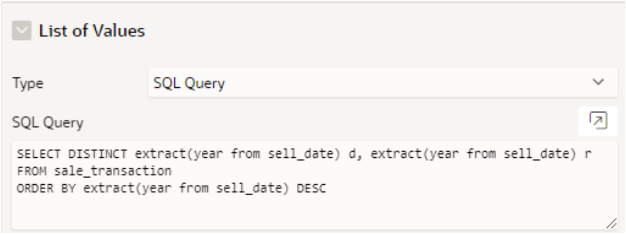


Figure 8.29: List of Values for Select List item used for filtering product sales by year.

After preparing the **Select List** item that we will use for filtering, we can write the source SQL Query for the series in the **"Monthly Sales Chart"** region. The SQL Query that we will use is shown in Figure 8.30. We will also map **"Label"** to **"SELL_MONTH"** and **"Value"** to **"COUNT"**. We are also ordering values by month number ascending and displaying the full month name on the **X** axis. We will also set the label of the **X** axis to **"Month"** and the **Y** axis to **"Sales Count"**.

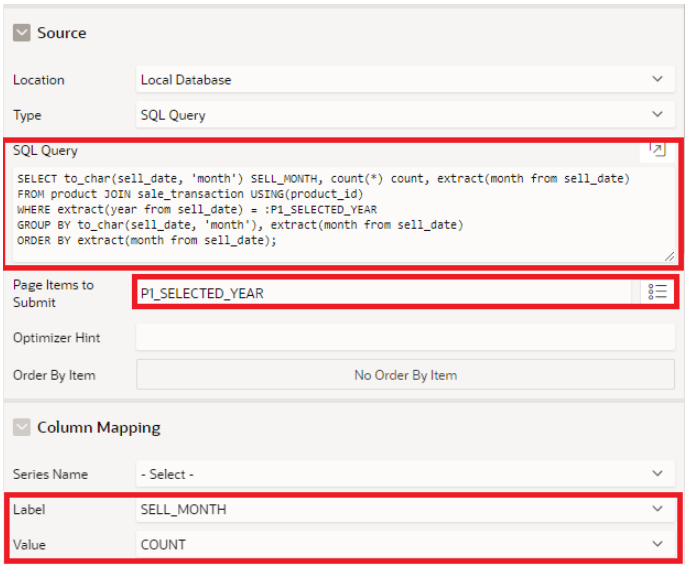


Figure 8.30: Setting source SQL Query and column mapping for Monthly Sales Chart region.

If you've configured everything correctly, your chart should resemble the one shown in Figure 8.31.

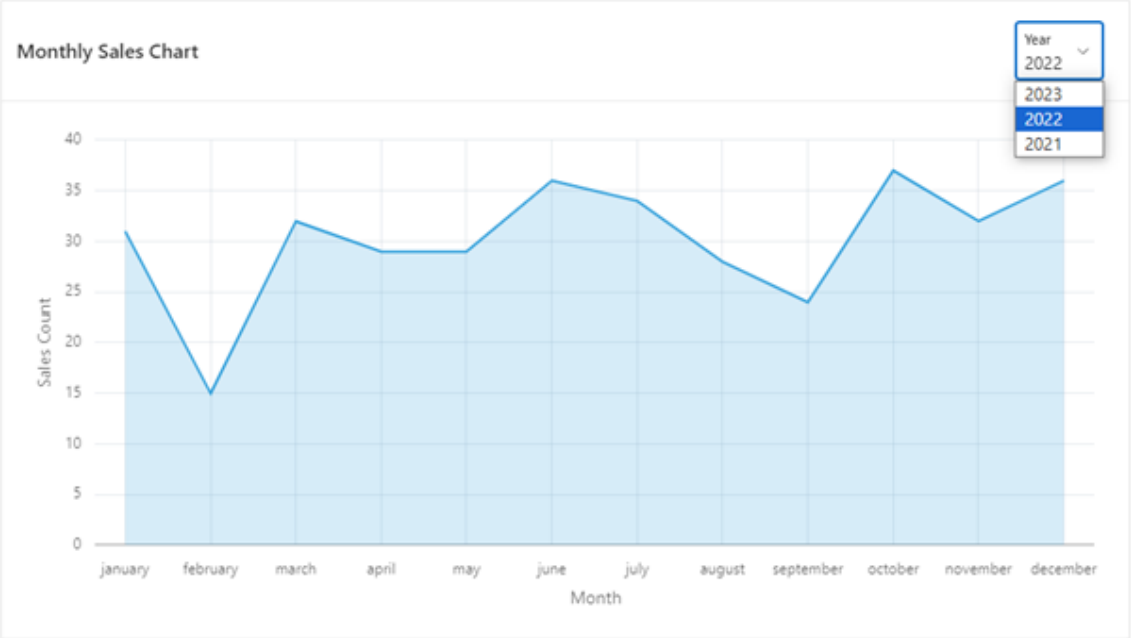


Figure 8.31: Monthly Sales Chart.

8.6 Questions

1. What is the primary purpose of Oracle APEX chart regions?
2. Which types of charts can you create using Oracle APEX chart regions?
3. How can you refresh a chart region in Oracle APEX when user interactions, such as filtering or sorting, occur?
4. What does the INSTR function do, and why is it used in Oracle APEX chart regions?

8.7 Answers

1. The primary purpose of Oracle APEX chart regions is to visually represent data, allowing users to gain insights and make data-driven decisions.
2. Oracle APEX chart regions support various chart types, including bar charts, line charts, pie charts, and more.
3. You can configure a Dynamic Action to refresh a chart region in Oracle APEX when user interactions change the data, such as filtering or sorting.
4. The INSTR function is used to find the index of the first occurrence of a substring in a string. In Oracle APEX chart regions, it's used to check if specific values match selected criteria, like product IDs in a list.

9. How to manage menus?

VERONIKA ŠALGOVÁ, MICHAL KVET, MIROSLAV POTOČÁR

9.1 How to manage menus

A navigation menu is automatically created after creating an application. It consists of list entries linking to the application pages. It is possible to change how and where a navigation menu displays by editing Navigation Menu attributes on the User Interface page. Types of navigation menus available in Oracle APEX are:

- Side Menu,
- Top Menu,
- Mega Menu.

9.2 Side Menu

This type of menu can be expanded or collapsed by clicking on the menu icon from the header. The navigation items are displayed using a tree component. Sub-items can be expanded or collapsed as well. We can see a full menu or collapse it to a narrow icon bar.

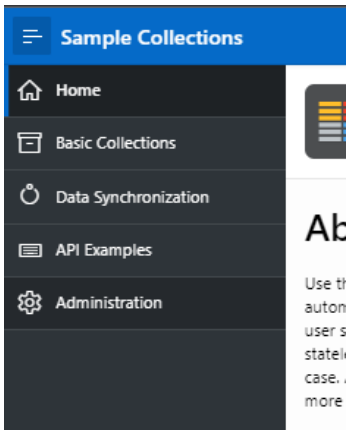


Figure 9.1: Expanded Side Navigation Menu

It is also possible to configure a navigation menu to be completely hidden, which is a default option. A narrow icon bar can be displayed when collapsed using Template Options.

9.3 Top Menu

A Top Navigation Menu is displayed at the top of the application. Oracle APEX provides two templates to display the menu, such as the *Top Navigation Tabs* and the *Top Navigation Menu.* The first type is suitable for simple applications with a few tabs. It is shown in Figure 9.2.

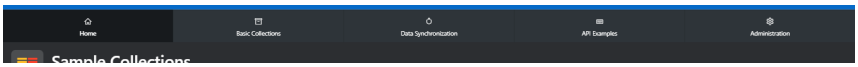


Figure 9.2: Top Navigation Tabs

The Top Navigation Menu provides a menu bar. It is ideal for more complex applications with several layers of hierarchy within the navigation. It is shown in Figure 9.3.

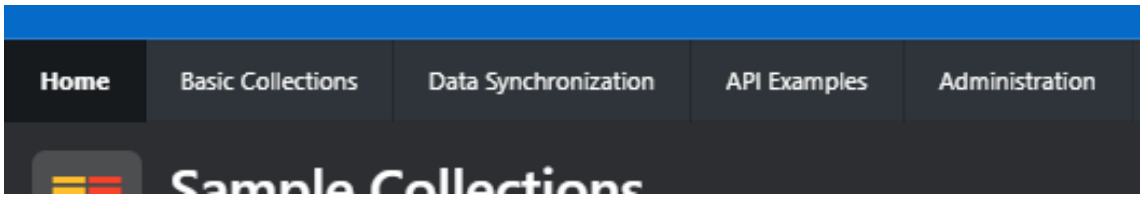


Figure 9.3: Top Navigation Menu

The Top Navigation Tabs template automatically positions to the bottom of the screen for small screen or mobile devices. The Top Navigation Menu resembles the most desktop applications.

9.4 Mega Menu

This type of menu is displayed as a pop-up panel that can be opened or closed from the icon from the header. Mega menus are helpful for displaying all navigation items at once.

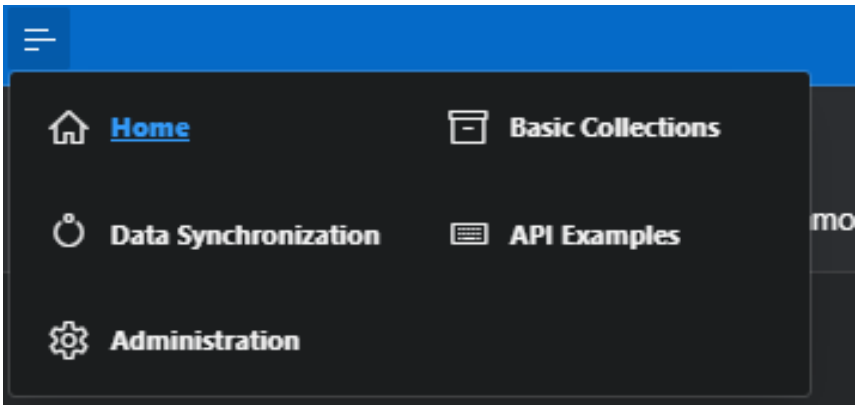


Figure 9.4: Top Navigation Mega Menu

In addition, it is possible to choose the number of columns in the menu.

9.5 Editing Menu Lists

Individual menu items form a list that can be created and edited similarly to any other list. It can be accessed in *Shared Components* under the Navigation region. You can choose a *Navigation Menu* or a *List* option. It is possible to change the position and way of displaying the menu by editing the Navigation menu attributes on the User Interface page. You get there by clicking on

Shared Components and *User Interface Attributes*. Under the *Navigation menu* tab, you can edit the attributes, which are shown in Figure 9.5.

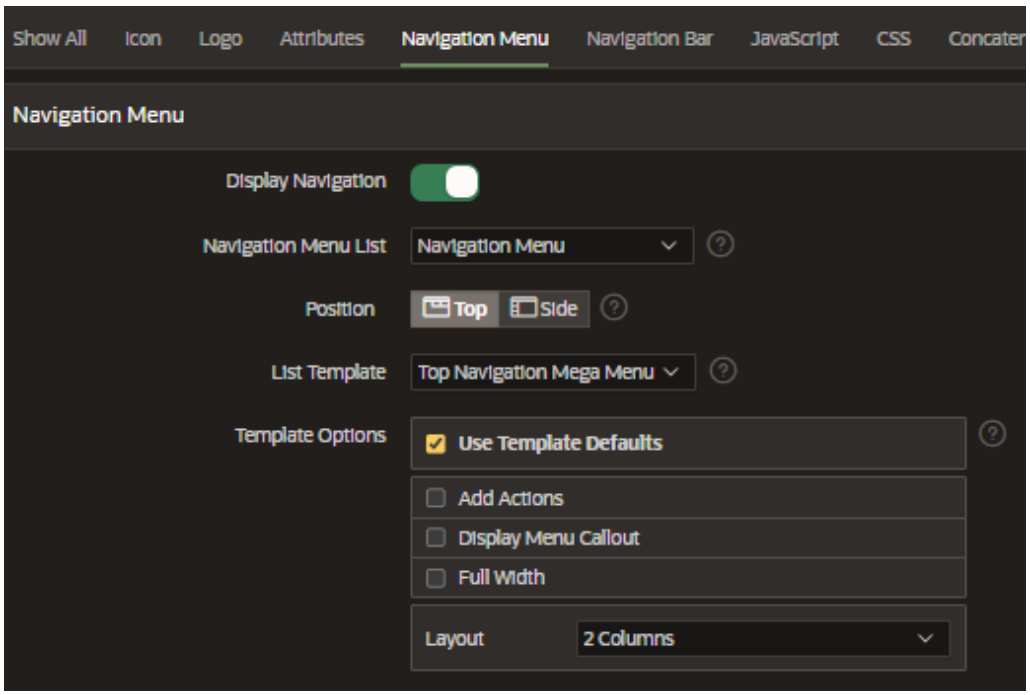


Figure 9.5: Navigation menu attributes

- Display Navigation – to turn the navigation on or off.
- Navigation Menu List – to choose the list utilized for the navigation menu.
- Position – to select the *Top* or *Side* position of the navigation menu.
- List Template – to choose from various templates to render the navigation menu.
- Template Options – to set the template options used for the navigation menu list.

After clicking on *Apply changes* button you can run the application and see the changes.

9.6 Questions

1. The menu used in the Oracle APEX applications is commonly placed in the left part of the screen. Which type does it refer to?
2. Where can you find the definition of the menu items to be edited?
3. Which menu type allows you to show all the items simultaneously without the necessity to expand sub menu?

9.7 Answers

1. Side Menu.
2. Individual menu items form a list that can be created and edited similarly to any other list. It can be accessed in Shared Components under the Navigation region.
3. Mega menu allows you to show all the items simultaneously just in one click with no necessity to expand sub menus.



10. How to collaborate in a team?

P. STANISZEWSKI, M. SOŃTA, A. KIERZKOWSKI

10.1 Collaborative knowledge production is the essence of low-code development

"Being collaborative distinguishes us as humans even more than our opposable thumbs" as we are naturally programmed to cooperate as a preferential approach (see i.e. [1]; [3]), and networked platforms are equipped with features to encourage collective and collective knowledge sharing.

The benefits of smart collaboration on digital platforms are twofold:

- the productivity aspect when the individuals use collective intelligence to develop a better product and
- the motivational aspect when the process activates the creative energies and community-driven approach to knowledge development.

10.2 Being online together

Oracle's APEX is one platform created to network and interact while working on the apps, and this approach fulfills the definition of a collaborative activism platform. Yet, this type of collaborative experience demands "a particular way of being together" ([3]). One part of the collaborative experience is the option to view, comment and co-create the content to produce knowledge collectively, whilst boosting creativity and spirit through shaping social relations is another dimension of the collaborative use of APEX's story. The Oracle's APEX community with more than ten regional community events per year, among many others:

- APEX World, The Netherlands
- APEX World, APEX Alpe Adria, Austria/Croatia/Slovenia
- APEX World, APEX Connect, Germany
- APEX World, ODTUG Kscope, USA

gained the stage of development to develop the commercial platform that has been promised to stay free for the users forever in a collaborative open-source way and the ambition to follow the directions of energizing community spirit.

With the number of interactions and projects, the way of community development is getting even more immersive, which brings opportunities "[...] modes of being together online, particularly in very immersive environments, may have an additional cognitive and perceptive layer that is yet unexplored" ([3]) with the ambition to reach the maturity of digital activism and inviting various

non-IT trained audiences.

10.3 Tech savviness through collaboration

If we search for academic articles about APEX, most of the papers cover business use cases, cases of of interdepartmental efforts to create a team which involves human resources, corporate social responsibility, communication with various stakeholders and similar.

This approach, accessibility to the platform, and wide range of educational content about APEX make this tool a chance to enter the world of low-coding and practice business awareness about the practical app applications. With this mindset, APEX opens an opportunity for non-IT business professionals to collaborate on IT projects and enables space to raise the level of tech savviness. Those competencies and attitudes create better employability environments for all the stakeholders involved in the software development process.

Ways of collaboration are usually related to the methodology of the project. Even if low-code platforms like APEX are designed to achieve goals very quickly, we typically need more than one developer to realize a project in the assumed time and budget. In such a situation, we need the right tool to help developers cooperate and address challenges and opportunities typical for coworking in IT projects. Depending on the scale and/or organization's requirements, it might be connected with aspects like:

- Standardization – when we want to ensure that all application elements are consistent. It is related to the UI standards, naming conventions, programming techniques, coding style, and all other standards typical for selected technology. For example, in the APEX world, you can just think about different ways of naming items, processes, or validations, or consider standards in selecting page numbers.
- Training – developing application with other developers helps increase skills. Other team members might be more advanced in aspects we are not very keen on. Even when using a low-code platform like APEX, which theoretically allows every user to be a "full-stack" developer, we can find specializations. For example, some people feel much more comfortable with the database side, others with pure APEX, and others with front-end features and languages. Having a bigger team helps the rest to develop skills they didn't have and increase the general quality of the application.
- Time – it's natural that more developers can deliver applications faster, resulting in better performance of the team or department responsible for a particular system or platform.
- Quality – the bigger team can implement a code review process and check the quality of elements delivered by team members. It increases the quality of the final product and limits the number of errors.
- Feedback – collaboration also means communication between developers and users. It is beneficial to have information from users about the quality of the application, reported bugs, or change requests.

Oracle APEX, as a mature platform, covers different collaboration strategies with built-in features, but also with options to be extended by external tools like collaboration suites, task management applications, or code verification tools.

Even if we have just started our journey with APEX, it is essential to know how we can collaborate with other users with native features to make such work more productive and be sure how to address typical problems like task assignments, working on the exact pages/features or controlling the status of planned work.

The most important part at the very beginning is communication. If the team is more than one person, you should have ways to plan work for the following days, inform about status, and control time and deadlines. Nothing is more important than regular meetings and communication if something is unclear. It's good to think about daily meetings, weekly planning, or other regular events. You can also implement one of the well-known methodologies, like Scrum, to have such

habits written in one place. Just start doing it, and you will find out the best collaboration way for your team.

In the beginning, when you think about some collaboration tools, you can consider basic spreadsheets to monitor who is doing what. Of course, at the market, there are plenty of dedicated applications for task management like Jira, Monday, ClickUp, Redmine, etc., and of course, with the team and scale growing, you can consider using one of them as well. However, most of those actions can be covered with features included in the Oracle Application Express platform.

10.4 Features description

10.4.1 Page locking

Page locking is a useful feature that prevents multiple developers from editing the same page simultaneously. When a developer is editing a page, the Page Blocking feature prevents other developers from making changes to the same page until the first developer finishes their work and releases the page. This helps prevent conflicts and ensures that changes are made orderly and controlled (see Figure 10.1).

Page	Name	Alias	Updated	Updated By	Type	Group	Lock	Run
0	0	0	3.1 years ago	-	Global Page	Unassigned		
1	Dashboard	dashboard	9 seconds ago	pstaniszewski@pretius.com	Report	Customers		
2	Customer Details	customer-details	2 years ago	-	DML Form	Customers		
3	Categories	categories	1 seconds ago	pstaniszewski@pretius.com	Interactive Report	Administration		
4	Category Details	category-details	3.1 years ago	-	DML Form	Administration		
5	Customer Types	customer-types	1.6 years ago	-	Interactive Report	Administration		
6	Status Details	status-details	3.1 years ago	-	DML Form	Administration		
7	Maintain Update	maintain-update	2 years ago	-	DML Form	Customers		
8	Contact Types	contact-types	1.6 years ago	-	Interactive Report	Administration		
9	Contact Type Details	contact-type-details	3.1 years ago	-	DML Form	Administration		
10	Validate Customer	validate-customer	2 years ago	-	Report	Customers		
11	Build Options	build-options	1.6 years ago	-	Interactive Report	Administration		
12	Administration	settings	2 years ago	-	Report	Administration		
13	Customer Usage Metrics	customer-usage-metrics	2 years ago	-	Legacy Calendar	Customers		

Figure 10.1: Page Blocking feature in APEX.

You can lock a specific page by clicking the lock icon on the page lists in the Application Builder or do it at the level of the page view.(see Figure 10.2).

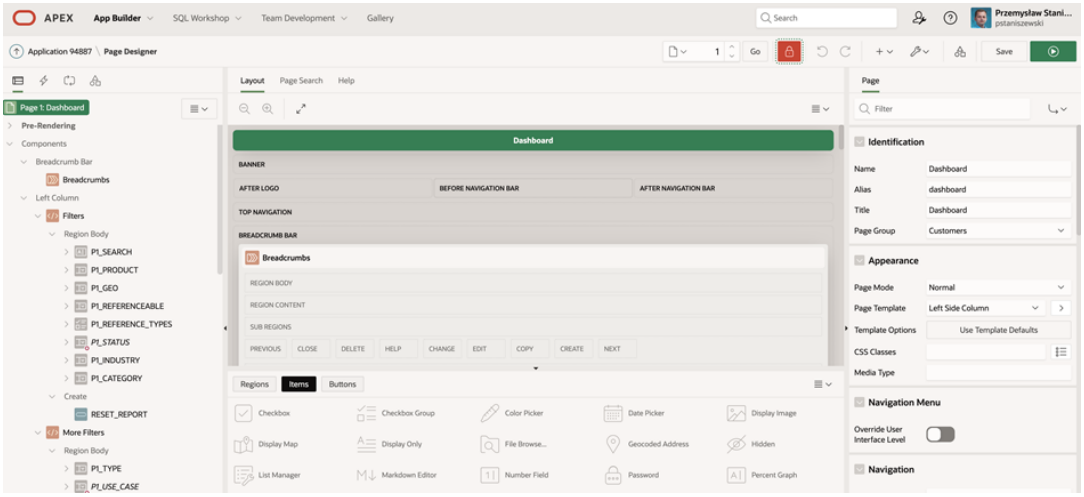


Figure 10.2: Locking a specific page.

While locking or unlocking the page, you can add a comment so you or other developers can see more information about the reason of changes, or documenting work progress.

10.4.2 Comments

Oracle APEX allows developers to add comments to pages, regions, items, and many other application elements. Comments can be used to provide context for other developers, explain the reasoning behind a design decision, or simply document changes made to an application. By using comments, developers can better communicate with each other and ensure that everyone is on the same page (see Figure 10.3).

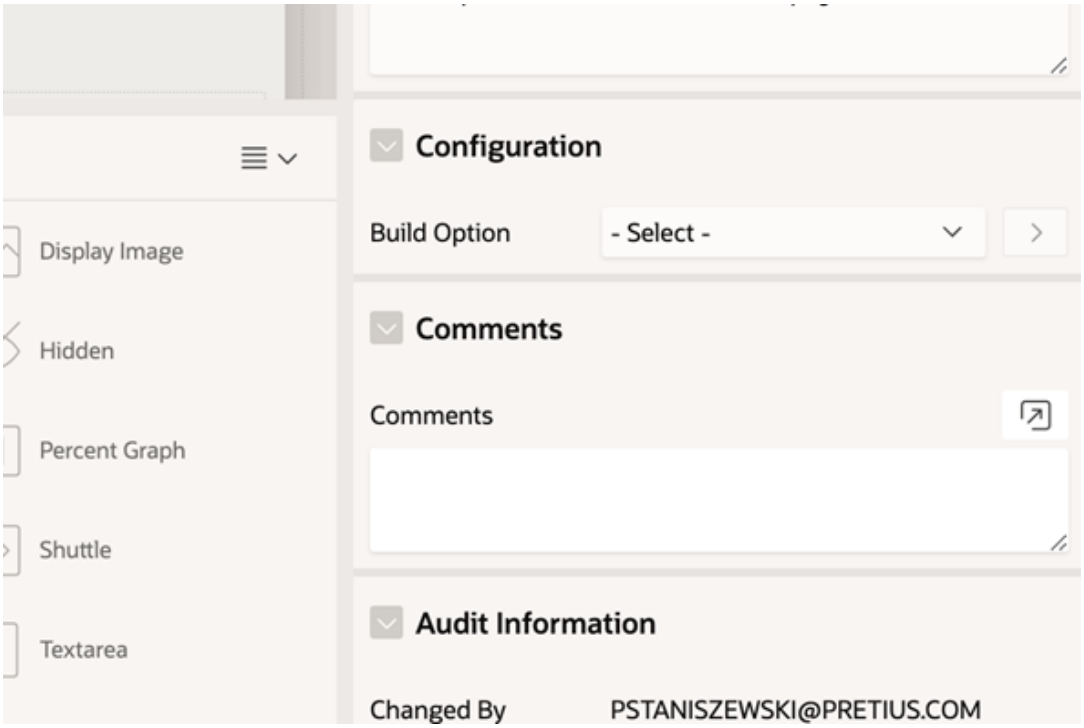


Figure 10.3: Adding comments.

Documenting your code for other team members or future developers that will take care of app development or maintenance is important. Even if using a low-code platform can self-define how a particular feature has been done, adding some information is still a good practice if a specific element is created uniquely.

10.4.3 Build options

Build options are another feature that can help teams collaborate more effectively in Oracle APEX. Build options allow developers to specify different build settings for different environments, such as development, testing, and production. This makes managing changes across different environments easier and ensures that the correct settings are applied when an application is deployed (see Figure 10.4).

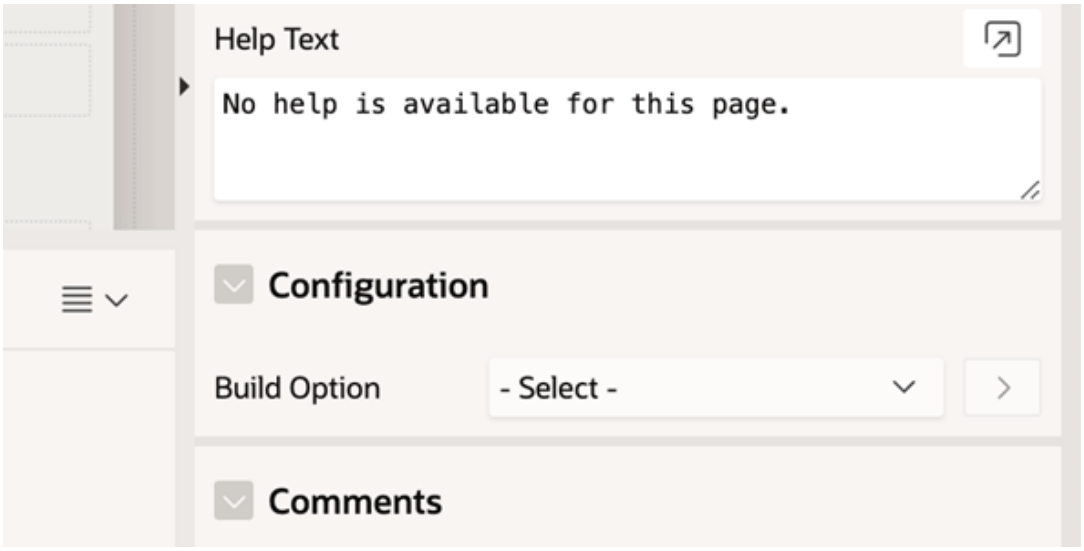


Figure 10.4: Build options.

You can treat them as information if a particular feature (page, item, process) should be included in the next release that will be exported/imported to another environment. While exporting/importing your application, you can select if a particular "build" is included. Just imagine the situation when one developer works on some feature and cannot make it before the deadline. Instead of deleting this page from the application, you can just create a dedicated "build" for it and exclude it during application export.

Using this feature can be a first step in defining CI/CD process for your application, which is essential in professional software development.

10.4.4 Team development

Oracle APEX's Team Development feature enables teams to work together on the same application in a controlled and collaborative environment. With this feature, team members can manage projects, in particular:

- Define milestones and project stages;
- Create, assign, and monitor tasks/issues;
- Apply classifications using Label Groups and Labels;
- Create issues templates, to make task creation or reporting consistent;
- Link Issues to an Application and Page;
- Comment and discuss existing issues. Change their status and monitor the progress of the project stage;
- Subscribe for a different types of issues;
- Monitor feedback provided by application users and decide what to do with it.

Team Development is one of four main sections available in APEX IDE. Its availability depends on your version of APEX. It has been introduced in APEX 19.2, but in the latest versions it is not available out of the box, but requires additional actions to make it available (see Figure 10.5).

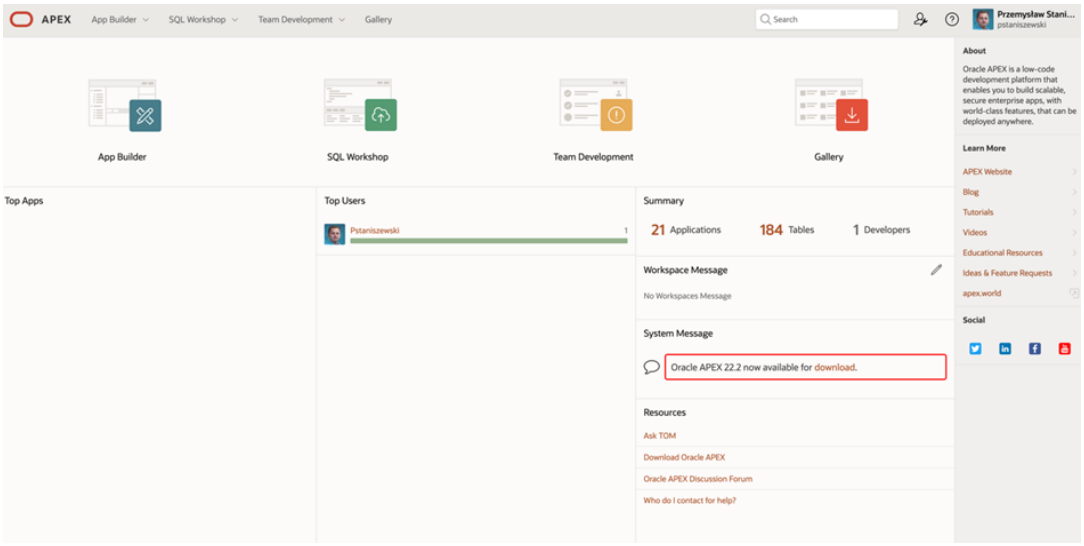


Figure 10.5: Team Development feature.

To turn it on, you need to select the "Team Development" option in the main menu. Then, you will be moved to the dedicated installation page. If you want, you can include examples of Issues, Milestones, Templates, and Labels, but you can also skip this part and activate the feature without it (see Figure 10.6).

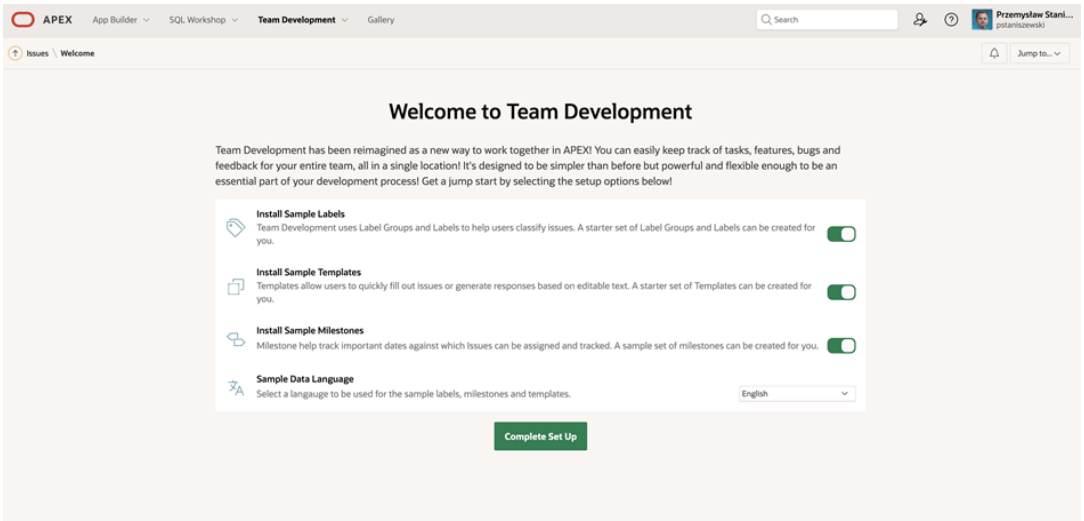


Figure 10.6: Activation of Team Development feature.

If your team would like to start using “Team Development” you should first configure it’s dictionary for your work style. Then, go to the Utilities section and define the types of your issues, project stages, and milestones, or templates for typical communication at the project (see Figure 10.7).

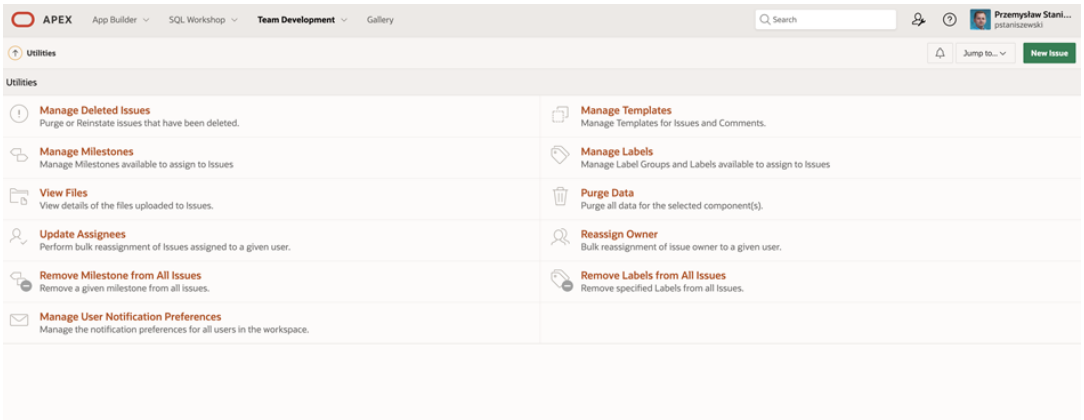


Figure 10.7: Customization of Team Development feature.

The most important part of this feature is the Issue section, where you can find all reported issues, categorize them, and assign them to particular persons. Every issue can be set to more than one developer, allowing each to add comments, discuss it within its context and change its status. If required, users can upload files and basically run all issue-related communication inside "Team Development". It is also important that all issues can be connected to a particular application and page, so it's straightforward to start an investigation and monitor which problems are related to which part of the application. Issue status is a simple version of issue workflow that can be found in many task management systems – users can report them, mark them as "in progress", return to the reporter, and close with information about the success or different reasons for closing (see Figure 10.8).

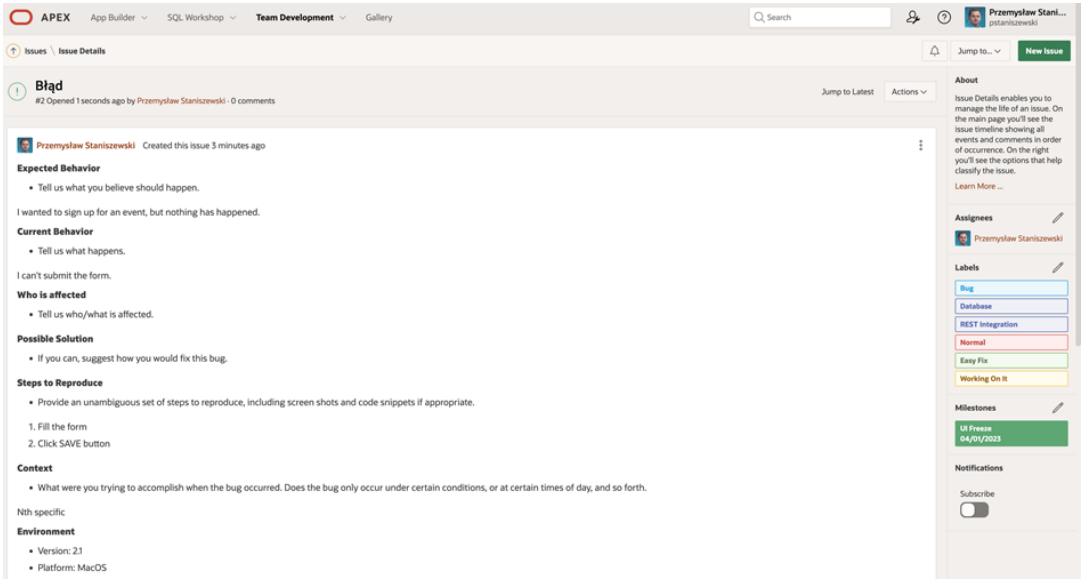


Figure 10.8: Issue-related communication inside Team Development feature.

Developers can also subscribe to issues assigned to them, which provides e-mail information when there is any action at the level of the issue. The application allows users to limit the number of e-mails by sending them as hourly or daily summaries.

We strongly recommend to start utilizing "Team Development" if your team does not have another task management system implemented, or you use very basic tools like spreadsheets to manage the project.

10.4.5 Feedback

Feedback Pages are a built-in feature in Oracle APEX that allows end-users to provide feedback on the application's functionality and performance. Feedback Pages can be customized to fit the application's needs and can include various types of feedback, such as bug reports, feature requests, and general feedback.

It is crucial to get information from end-users and treat them as an extension of the development team. The more closely we collaborate not only internally, but also with actual users of the application, the better quality of the application we will have. It also allows the team to adapt the application to real needs and helps build trust between the development team and users (see Figure 10.9).

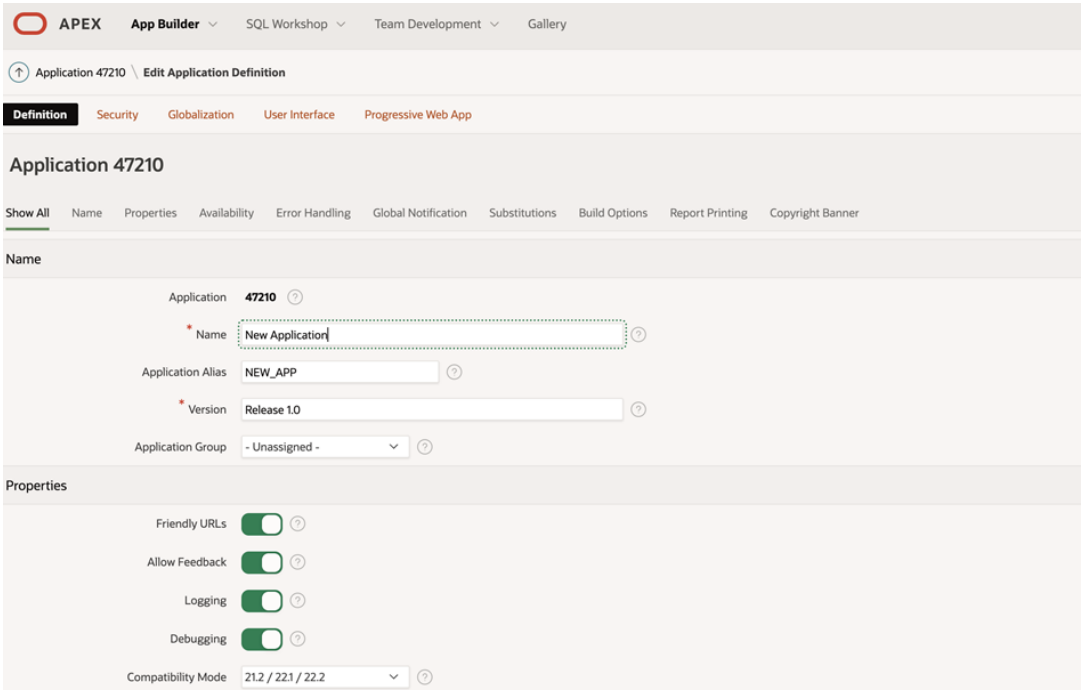


Figure 10.9: Allowing feedback in Application definition.

To turn it on, you need to select an option during application creation. If the application already exists, go to "Edit Application Definition" and mark the "Allow Feedback" option in the Properties section.

After activation, it adds a dedicated icon in the application menu. After clicking it, the end user can submit feedback (see Figure 10.10).

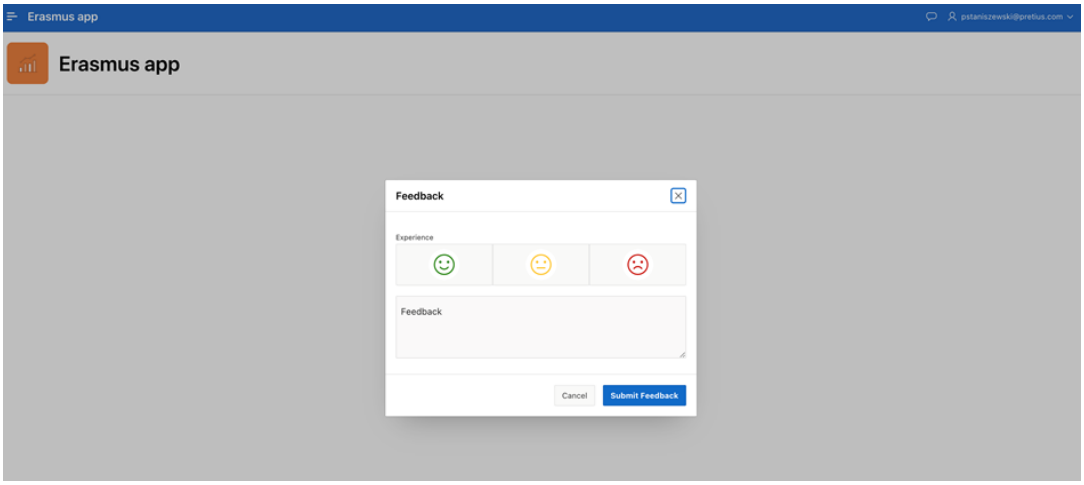


Figure 10.10: Submitting feedback in application.

The look of the Feedback page can be adjusted as it's a standard APEX page. All information the end-user provides is available in the Team Development section and can be analyzed by the team (it is also possible to track it at the level of a particular application in its Administration section). In addition, developers receive detailed information related to reported feedback (application ID, page number, user's data, information about its environment, etc.) and can transform it into an issue.

10.4.6 Standardization

While developing applications in the team, it is also very important to keep consistency between the development standards of different team members. As in every aspect of collaboration, communication is the key to success – at the beginning of the project, the team should agree on the way of development and utilization of APEX components. The most important aspects of that include:

- Naming convention (for items, processes, validations, dynamic actions, etc.);
- Page numbering and naming (alias);
- Allowed elements (e.g., for dictionaries we use just Interactive grids, every form must be displayed as a modal page);
- UX/UI standards (e.g., look and feel of typical pages, alignment policy for different data time, consistent formats of data, recommended icons for particular actions, etc.);
- Data consistency (e.g. having common dictionaries defined in one place the shared components).

APEX gives a couple of solutions to help teams with standardization management. Most of them can be found in the "Utilities" section at the level of the selected application (see Figure 10.11).

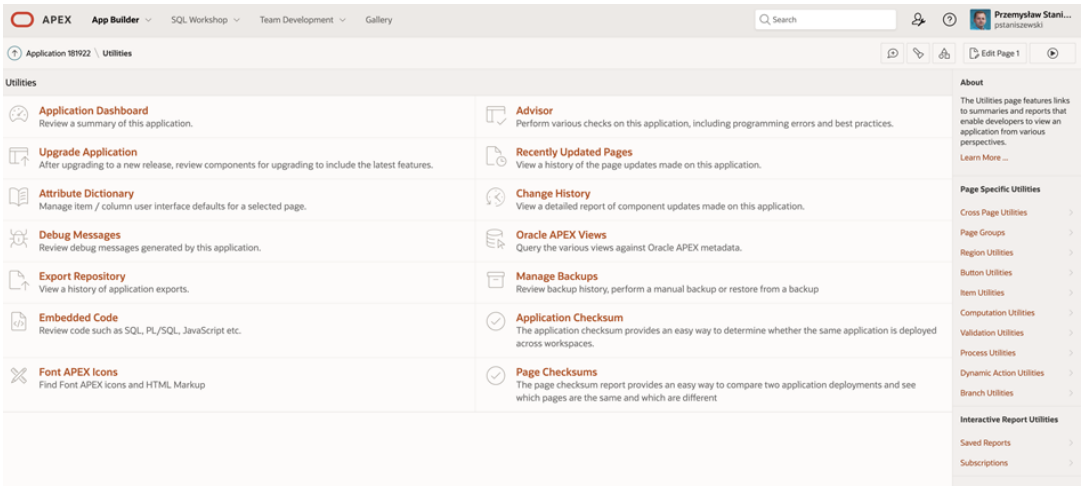


Figure 10.11: Utilities menu.

The most important at the beginning of development include:

- Cross-page reports (Page Specific Utilities) – allow developers to compare different APEX elements (page attributes, reports, buttons, items, etc.) in the application to check consistency and update them at once without manually changing attributes in multiple places.
- Change history/Recently updated pages – allows tracking actions taken by developers.
- Defaults definition (Attribute Dictionary) – allows the creation of a common dictionary to keep naming consistent, formatting, data type, etc., for elements that repeat in many places in the application. For example, "Update date" can exist on different reports or items in the application, but we always want to have the same label and date format for every occurrence.

Of course, APEX gives many additional features and strategies to keep consistency at the database and application level. For those who would like to extend their knowledge, please read the APEX documentation, and check more advanced options like Global Page, Blueprints, User Interface Defaults, Master Theme Application and Subscribe mechanism, APEX Views, or Plug-ins.

10.5 Conclusions

As a mature development platform, Oracle APEX provides many features to help with collaboration. As low-code developers, we can utilize them and professionally create and maintain APEX-based applications without extending the platform or integrating it with external tools. If you are new to APEX and new to cooperation strategies in programming, start with the most fundamental steps described in this chapter:

- Communication and regular meetings with the Team (live or remote – use tools like Google Meet, Zoom, or MS Teams for video/voice, and Slack, Google Chat, or MS Teams for fast messaging);
- Start using collaboration tools, e.g. APEX Team Development, to plan the development and verify the progress;
- Start documenting your code and standards. And verify them!

Collaboration in low-code development sometimes might sound exaggerated, but it is worth trying and gives just as much profit as in standard programming. What's more, most of the typical collaboration strategies like "code review", "pair programming", or "mob programming" can be used as well with the low-code platform, even if sometimes the reviewed element is not pure code but a process, page, or configuration.

Depending on the scale of the project, or standards implemented in the organization, at some point in time, you may need to extend your collaboration process. For example, in large projects, you will need to start using code repositories, automated building, and deploying tools, or software

dedicated to the testing and verification of the quality of your code. Utilizing elements of professional CI/CD processes like Git, Docker, Flyway, Liquibase, Jenkins, Gradle, SonarQube, Selenium, Cypress, and many others is also possible with the APEX-based application. However, in many situations, elements included in the platform are enough to create a game-changing application with a team of fellow developers.

10.6 Questions

1. What is page locking feature and why does it matter?
2. How can developers communicate during the development with APEX?
3. What can team members utilize by using Team Development feature?

10.7 Answers

1. Page locking is a useful feature that prevents multiple developers from editing the same page simultaneously and therefore preventing conflicts.
2. Developers can add comments to pages, regions, items, and many other application elements. Comments can be used to provide context for other developers, explain the reasons behind a design decision, or simply document changes made to an application. By using comments, developers can better communicate with each other and ensure overall quality of software.
3. Team Development feature enable team members to manage development projects, in particular: to define milestones and project stages; create, assign, and monitor tasks/issues; apply classifications using Label Groups and Labels; create issues templates, to make task creation or reporting consistent; link Issues to an Application and Page; comment and discuss existing issues; change their status and monitor the progress of the project stage and subscribe for a different types of issues.

11. How to benefit from a gallery of applications and plug-ins?

VJERAN STRAHONJA, DIJANA OREŠKI

This chapter explores starter apps, sample apps and plug-ins available in Oracle APEX. All Oracle APEX workspaces come with Sample Apps and Starter Apps (see Figure 11.1) and plug-ins.

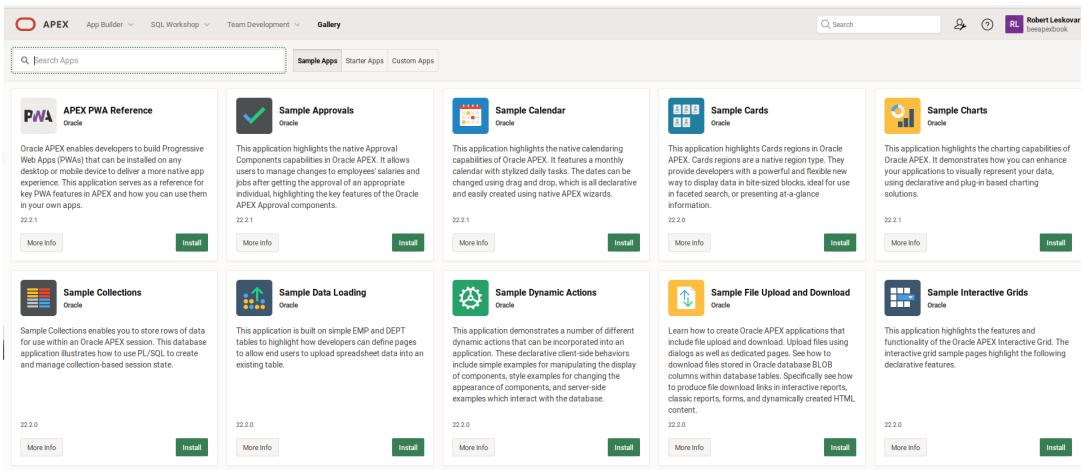


Figure 11.1: Oracle APEX workspaces come with Sample Apps and Starter Apps.

Starter apps are stand-alone point-solutions that meet simple requirements and do not require complex solution. On the other hand, sample apps are focused on the specific functionality and provide a guide on how to make use of a particular feature. Plug-ins enable users to extend applications with custom functionality.

11.1 How to install sample and starter apps?

Users can download sample and starter Apps from the Gallery. If enabled by the instance administrator, apps can be installed using the Gallery.

To **install** apps from the Gallery:

1. Go to the Workspace home page.
2. Select **Gallery**. The sample apps, starter apps, and custom apps by means of cards are displayed when the **Gallery** page first loads.

3. Select **Sample Apps** or **Starter Apps**. Cards are used to list the apps. Unless the program has previously been installed, each card contains an Install button. The Run and Remove App icons are present if the application has already been installed.
4. Select **Install App** to do the installation of the app
5. Expand the Advanced Settings section. (Not obligatory)
   a. Indicate whether the Application ID will be assigned manually or automatically (the default).
   b. Change the Parsing Schema.
6. Click Install Application to finish the installation.

By selecting the Remove App icon from the Gallery, users can uninstall apps that have been installed from there. Applications that were imported from GitHub and downloaded do not have an uninstall icon, so they must be deleted in order to be uninstalled from the workspace. Steps to **uninstall** apps from the Gallery are as follows:

1. Go to the Workspace home page.
2. Select the Gallery icon. The Sample Apps, Starter Apps, and Custom Apps cards are displayed when the Gallery page first loads.
3. Select Starter Apps or Sample Apps. Directly downloaded programs from the Gallery will have a Remove icon and a Run icon visible.
4. To remove the app, click Remove.

11.2 Starter Apps

Starter apps are useful apps that offer standalone point solutions, created to satisfy straightforward requirements that don't call for a substantial and unnecessary complex answer. These apps can be used "out of the box" or can be enhanced with your own unique features (see Figure 11.2).

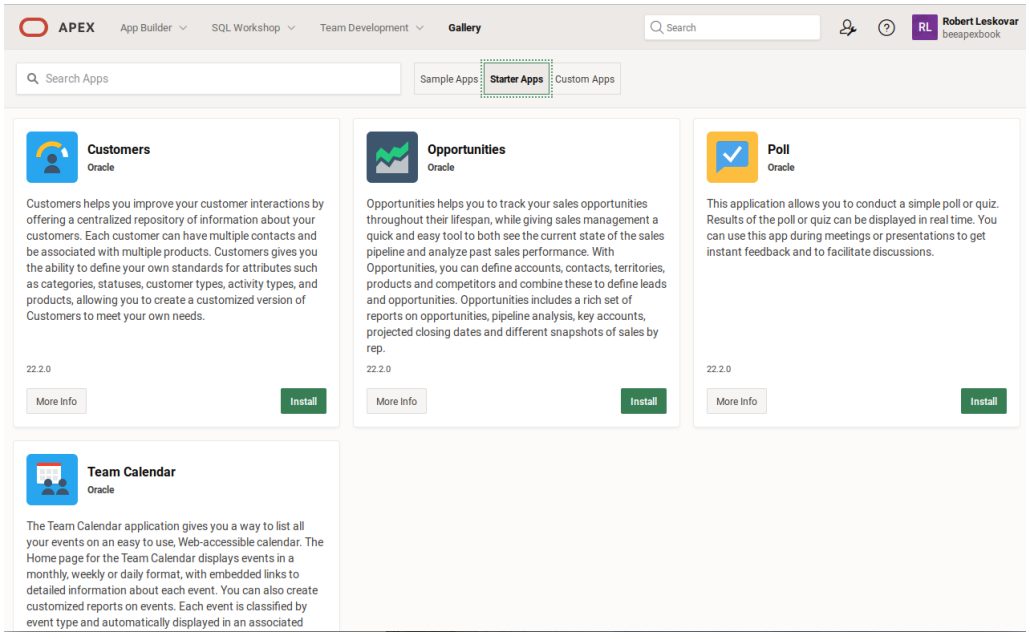


Figure 11.2: Starter Apps in Gallery.

Examples of starter apps are:

- **Customers** – By providing a consolidated database of client data, Customers app enables users to enhance customer interactions. Each customer has the potential to be connected to several contacts and items. Customers allows you to set your own criteria for features like

categories, status, customer kinds, activity types, and goods so you may design a special version of Customers to suit your requirements.

- **Opportunities** – This application provides sales management with a quick and simple solution for both, seeing the present condition of the sales pipeline and analyzing historical sales performances. It helps you track your sales opportunities throughout their life cycle. You may define leads and opportunities by combining accounts, contacts, regions, goods, and rivals. Opportunities offers a comprehensive collection of information on opportunities, pipeline research, significant accounts, anticipated closing dates, and other sales representative snapshots.
- **Poll** – Users can use this application to create a quick survey or test. The poll or quiz's results can be seen immediately. This tool can be used to facilitate discussions and gain immediate feedback during meetings or presentations.
- **Team Calendar** - With the help of the Team Calendar app, users can easily list all of events on a Web-based calendar that is simple to use. The Team Calendar's home page lists events in a monthly, weekly, or daily style along with integrated links to more specific details for each event. Additionally, you can design unique reports on events. Each event is categorized according to its category, and the corresponding color is automatically shown. Users can develop new event types or change the attributes of already existing event types. Users have the option to build their own groups to better serve their needs, and they may send emails to individuals or groups informing them of upcoming meetings.

Here is an example of application installation for Team Calendar (see Figure 11.3).

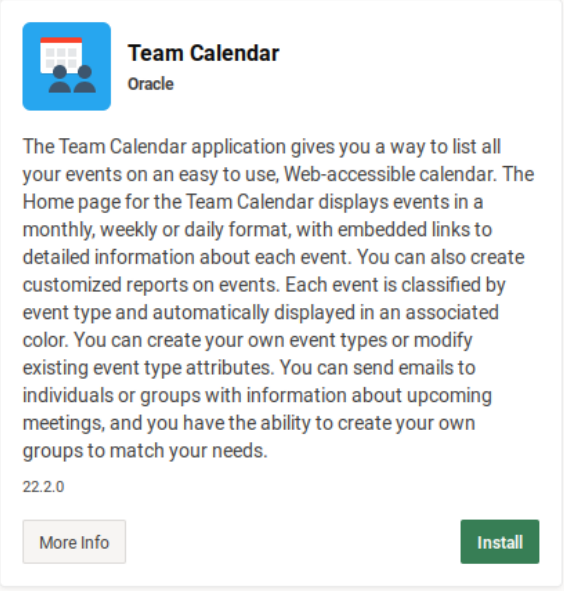


Figure 11.3: Installation of starter app Team Calendar.

Click Install.

11.3 Sample Apps

Sample apps are created to highlight specific capabilities and act as a developer's manual for utilizing a specific feature (see Figure 11.4).

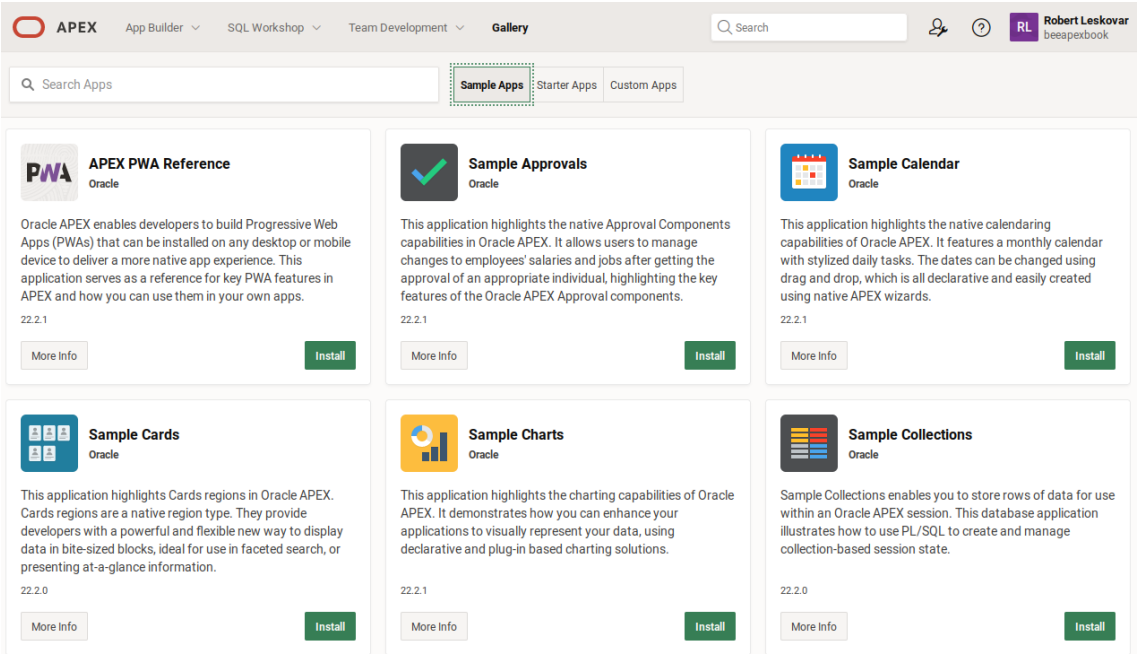


Figure 11.4: Sample Apps.

Among others we can choose:

- **APEX PWA Reference** - To provide a more native app experience, developers can create Progressive Web Apps (PWAs) using Oracle APEX, which can be deployed on any desktop or mobile device. The main PWA capabilities in APEX and how to apply them in your own apps are included in this application.
- **Sample Approvals** - This application showcases Oracle APEX's native Approval Components' capabilities. The Oracle APEX Consent components are highlighted, allowing users to manage changes to employees' payments and jobs after receiving the right person's approval.
- **Sample Calendar** - This application showcases Oracle APEX's built-in calendaring features. A monthly calendar with styled daily tasks is included. Drag and drop, which is completely declarative and quickly generated using native APEX wizards, may be used to update the dates.
- **Sample Cards** – In Oracle APEX, this application highlights the Cards regions. Regions with cards are a form of native region. They give developers a strong and adaptable new method to present data in bite-sized bits, which is perfect for usage in faceted search or presenting information quickly.
- **Sample Charts** – This application showcases Oracle APEX's charting features. It explains how you can use declarative and plug-in based charting tools to improve your applications so that they visually portray your data.
- **Sample Collections** – This application provides users to store rows of data in Sample Collections for usage in an Oracle APEX session. This database application demonstrates the creation and management of collection-based session state using PL/SQL.
- **Sample Data Loading** - This application is constructed using straightforward EMP and DEPT tables to demonstrate how developers may design pages that let users add spreadsheet data into a table that already exists.
- **Sample Dynamic Actions** – This application showcases a variety of dynamic actions that can be added to applications. These declarative client-side behaviors include straightforward examples for modifying how components are displayed, style examples for customizing how components look, and server-side examples for interacting with databases.

- **Sample File Upload and Download** – Learn how to build Oracle APEX apps with file upload and download functionality. Utilize both, dedicated pages and dialog boxes to upload files. Check out the process for downloading files from Oracle database BLOB columns in database tables. See how to make file download links in forms, interactive reports, traditional reports, and dynamic HTML content.
- **Sample Interactive Grids** – The Oracle APEX Interactive Grid's features and functionality are highlighted in this application. Those features are highlighted on the interactive grid sample sites.
- **Sample Maps** – Numerous examples of viewing coordinate data on a map are provided in this application. Use the heat map feature, lines, polygons, or map markers. The Oracle Spatial capabilities (available in every Oracle Database) may be readily integrated with the APEX Map Region to carry out a "Within Distance Search," "Nearest Neighbor Search," or other spatial analyses.
- **Sample Master Detail** – This application showcases Oracle APEX's native master detail capabilities. There are four possible master detail page layouts in the application. The first two layouts use editable Interactive Grids to display master detail on a single page. The final two layouts present the master detail on two pages, using a combination of editable Interactive Grids, form inputs, vintage reports, and modal popups.
- **Sample Reporting** – This application showcases Oracle APEX's reporting features. Using SQL, users are able to declaratively construct Interactive Reports, Interactive Grids, Faceted Search Reports, Cards Reports, and Classic Reports.
- **Sample REST Services** – This application demonstrates how to use Oracle APEX to connect to external REST services. Oracle.example.hr, a prototype RESTful Service, is used by the app. Examples in this application demonstrate how to filter, how to add pagination, and how to produce a straightforward tabular report on REST service data.
- **Sample Trees** – Find out how to use a SQL query to construct a tree control. The Oracle APEX application in this example demonstrates several ways to incorporate tree controls.
- **Universal Theme Reference** - By giving you a simple way to navigate through the numerous templates, template choices, and theme styles, this app introduces users to Universal Theme. The examples show how users may quickly manage your page layout to produce a stunning application.

11.4 Plug-ins

Plug-ins allow for the Application Express framework to be readily extended with custom item types, region types, processes and Dynamic Actions. Once defined, plug-in based components are created and maintained very much like standard Application Express components. Plug-ins enable developers to create highly customized components to enhance the functionality, appearance and user friendliness of their applications. Plug-Ins allow users to add additional functionality to APEX apps with plug-ins that is not built-in to the platform. The APEX Community has developed a large range of different plug-ins useful for expanding existing features or adding new features. There are hundreds of plugins for Oracle Apex categorized into five types:

- Dynamic action
- Region
- Item
- Process
- Authentication

Oracle Application Express offers various plugins within each category. Plug-ins give you the option to add new features to your application declarative, thereby enhancing the built-in types. Because plug-ins are intended to be reused, developers can share them with the Oracle Application Express Plug-in community by using the Plug-in Repository as well as export and import them to

different workspaces. The steps involved in implementing a plug-in are as follows:

- Add a plug-in to your application workspace or create one from scratch.
- To utilize the plug-in, edit or create a dynamic action type, item, region, process, or authorization scheme.
- Launch your application to check the functionality of the plug-in.

A central location for developers to share and download plug-ins is the Plug-in Repository. On the Oracle Technology Network, the repository can be found.

- To see the repository for plug-ins, go to the page for Shared Components:
  - Click App Builder on the Workspace main page.
  - Decide on an application.
  - Choose Shared Components on the application's home page.
- The page for shared components appears.
- Select Plug-ins from the Other Components menu.
- Press the View Plug-in Repository button.

The repository for Oracle Application Express Plug-ins is displayed. To access the Plug-ins page:

- Go to the Shared Components page.
  - Click App Builder on the Workspace main page.
  - Select an application.
  - Choose Shared Components on the application's home page.
- Select Plug-ins from the Other Components menu. The Plug-ins tab is automatically selected when the Plug-ins page loads. The whole list of plug-ins is displayed.

To create a plug-in:

- Go to the Shared Components page.
  - Click App Builder on the Workspace main page.
  - Select an application.
  - Choose Shared Components on the application's home page. The Shared Components page appears.
- Select Plug-ins under Other Components,
- Click Create. The Create Plug-in wizard appears.
- Click Next after choosing the method for creating a plug-in under Create Plug-in.
- Within Name:
  - Name – Fill in the name of the plug-in.
  - Internal Name - Fill in the internal name of the plug-in. Name should be unique within the application.
  - Type - Decide which kind of component can make use of this plug-in. The options under Callbacks and Standard Attributes vary depending on the plug-in type you choose. See field-level Help for more information.
  - Category - only appears if Dynamic Action is the specified kind. On the user interface, choose the category the plug-in is listed under.
- Within Source:
  - PL/SQL Code - Enter a PL/SQL anonymous block of code containing the steps required to render, validate, run, and execute this plug-in's callbacks. You can alternatively store this code in a PL/SQL package in the database for performance reasons.
  - Do not validate PL/SQL code - To only parse the PL/SQL code at runtime, choose this option. If not, the plug-in is built once the code has been parsed.
- Within Callbacks, configure the attributes. The plug-in type determines the attributes that are displayed. Check out the field-level help to examine samples and learn more about attribute.
- Within User Interfaces, choose which displays the App Builder must be able to support for this plug-in. Options include:
  - Desktop

  - Mobile
- For Standard Attributes, select the attributes that apply to this plug-in. Standard Attributes dispense with certain plug-ins.
- Within Information:
  - Version - To specify the plug-in version, enter a string.
  - About URL - Enter a URL leading to the homepage of the plug-in-author in's or to more details about the plug-in.
- Click Create Plug-in.

Please note that the above applies to APEX version 22.1. Newer version 23.1 delivers an even better way to create a Plug-in. To import a plug-in:

- Go to the Shared Components page:
  - Choose App Builder on the Workspace home page.
  - Choose an application.
  - Select Shared Components on the Application home page.
- Select Plug-ins within Other Components
- Select Import. The import Plug-in page appears.
- For Specify File:
  - Import file - Enter or navigate to the import file's name.
  - File Type - Choose Plug-in.
  - File Character Set - Choose the character set encoding for import files.
  - Click Next.
- For File Import Confirmation, select Next.
- For Install, select Install Plug-in.

To export a plug-in from the Plug-in page:

- Go to the Shared Components page:
  - On the Workspace home page, click App Builder.
  - Select an application.
  - On the Application home page, click Shared Components.
- Within Other Components, select Plug-ins.
- Under Tasks, select Export Plug-in.
- On the Export Plug-in page:
  - Application - Choose the application to export the plug-in from.
  - Plug-in - Choose plug-in.
  - File Format - Choose file format of the export.
  - Choose Export
- Choose Export.

In the next few lines one example of plug-ins is provided. The example refers to the updated sample calendar app and dynamic action plugin. This example plug-in can be used to format days in the new date picker item. Users can disable it, add tooltips or classes to each day in the day grid of the date picker. There are no prerequisites for using this plugin. How to use it:

- Create an application
- Import the Plug-in in Shared Components - Plug-ins
- Add a new date picker
- Add a Dynamic Action on Page Load and select as action the imported plug-in
- Set an URL of a iCalendar standard file format or enter a SQL Query that should format the calendar days
- Select as Affected Element the desired date picker and run the page

This chapter demonstrated that plugins can offer several benefits. No matter how strong it may be, no single piece of software can provide every function for every user. The gap between form and function is filled by plugins. They make it simpler to add particular functionality to software

and apps. As such, plugins are time-saving. The biggest benefit of using plugins is that they reduce the amount of time required for development, which lowers the overall cost. Furthermore, plugins allow users to customize the features and functionality. Most plugins enable to turn on and off particular settings. It is a simple process to remove a plugin, if there is a need for that.

11.5 Questions

1. What are sample apps and what are their benefits?
2. What are starter apps and what are their benefits?
3. What are benefits of plugins?

11.6 Answers

1. Sample apps are created to highlight particular capabilities and act as a developer's manual for utilizing a specific feature.
2. Starter Apps are useful apps that offer standalone point solutions, created to satisfy straightforward requirements that don't call for a substantial and unnecessary complex answer.
3. Plug-ins enable developers to create highly customized components to enhance the functionality, appearance and user friendliness of their applications. Plug-Ins allow users to add additional functionality to APEX apps which are not not built-in.



12. How to manage packaged and multilingual applications?

R. LESKOVAR, U. RAJKOVIČ, A. BAGGIA

12.1 Application and packaged application

This section will teach you how to utilize the potential of the Quick SQL feature to generate application with almost no knowledge of DDL and completely within Oracle APEX. After generating the application out of Quick SQL we will prepare packaged application and migrate it to new workspace (i.e. “production site”).

In Chapter 11 we presented applications that are supplied in APEX by Oracle. These applications are classified into two groups: a) productivity and b) sample applications. Productivity applications address specific business needs. The embedded features more or less completely satisfy requirements imposed by business situation. Sample applications demonstrate different functionalities. Both can inspire developers to tailor them to fulfill business needs. These applications are examples of packaged applications. When you install (APEX term is import) these packaged applications, notice that database objects (tables, sequences, triggers, functions, procedures and packages) are created and data is populated in tables.

The shortest definition of **packaged application** says that it is a pre-built application that can be installed and configured in an Oracle APEX workspace. APEX uses two important concepts closely related to application migration (export and import): a) supporting objects and b) shared components. Supporting object control installation, upgrade and de-installation by defining prerequisites, substitution strings, build options, pre-installation validations, scripts and messages. Shared components define application logic, security, navigation and search, user interface, files and reports, data sources, workflows (if installed), globalization and other components. So packaged applications are the one with defined supporting objects. At this moment you might ask yourself: "Why and how can developers create custom packaged application?".

Answer to "why": developers would create custom packaged application if there is a requirement to migrate application to new workspace with supporting objects and shared components. Data base object definitions (with DDL statements) and operations on these objects (DML statements) are executed during installation, update and de-installation. Simplified: during installation, tables could be created and populated with data, during upgrade, tables could be modified and during de-installation, tables could be truncated or dropped. The similar could be forced for other database objects such as triggers, sequences, stored procedures, functions and packages. The key benefit of packaged application is that it is ready to run with data definitions and data itself.

Answer to "how" include the following steps:

- define the scope of application
- define the scope of application
- create and execute script to define tables
- create and execute script to insert sample data
- generate application
- make adjustments in application
- prepare packaged application and test it in new workspace

Before going step-by-step let's look at the application import and export wizard in APEX. To move the application into new workspace, the application needs to be exported and then imported. The export wizard in APEX writes a text file (PL/SQL statements) on: a) developer's computer or b) to remote workspace via REST service (see Figure 12.1).

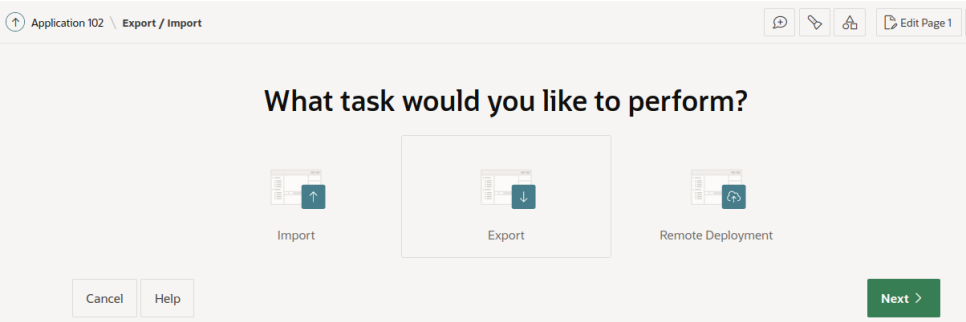


Figure 12.1: Application import and export wizard.

There are several options in export like splitting into multiple files, choosing file character set, opt in or opt out public or private reports and similar. In most cases, the beginner simply allows APEX to apply default settings. Whilst the developer logs into a new workspace and launches import wizard which loads the exported text file. Definitions of tables, views, triggers, sequences, functions, procedures and packages are not migrated with application if we do not prepare supporting objects. If a target workspace already contains database object definitions and data in the tables then we should prepare supporting objects to handle such situations.

The first part of this chapter will use APEX wizards as much as possible. Then simple SQL INSERT statements not supported by wizard in APEX will be presented. To generate INSERT statements the reader can import data in spreadsheet and then export INSERT statements with other tools (i.e. SQL Developer or TOAD). The application in this chapter will introduce a page component called lists of values (LOV). It belongs to shared components. List of values have two purposes: to allow the user to fill the form field on the the web page easily (drop-down, radio button) and with just prescribed values to avoid typing mistakes. Two types of list of values are available: dynamic are based on SQL query while static offers fixed set of values. The benefit of dynamic list of values is that the domain depends on table or view. The end user can change the domain by inserting, updating and deleting rows in the table. The drawback is that a table must be created and maintained. If the domain does not change and the number of rows in the table is small then a static list of value could be applied. In that case only the developer can change the domain (it is "hard-coded").

The second part will demonstrate the creation of packaged application, exporting it, and importing it in new workspace.

The third part of the chapter will provide instruction to create a multilingual application. As a supplementary study material, three exported applications are provided as well as video clip which shows the development process in detail.

12.2 Application

12.2.1 Scope of application

The application aims to manage jobs and competences for a HR department. There can be a few hundred jobs in one company. One job may include a specific subset of all known competences. Required level of competence is prescribed by HR department for each job. The levels of competence describe the ability to: assist, use, master, tailor and innovate. So, the idea is to develop an application to prepare and manage job profiles. The application uses three tables: JOB, COMPETENCE and JOB_COMPETENCE with minimal number of data fields (for the sake of simplicity and conciseness). Each table has a primary key and table JOB_COMPETENCE has two foreign keys to relate each row with rows JOB and COMPETENCE table. Foreign keys ensure data consistency: namely a row in the JOB_COMPETENCE table can only exist if rows with the same primary keys exist in tables JOB and COMPETENCE. Table names in this chapter will be prefixed with "CH12_" to assure uniqueness and to prevent interference with tables used in other chapters.

12.2.2 Create tables

We can create tables in APEX:

- with no knowledge of data description language by clicking in Object browser. But illustrating this process with a lot of figures would take too much space in this book.
- with knowledge of DDL in SQL Workshop > SQL Commands by entering syntactically correct commands.
- with basic knowledge of data modeling (see Chapter 2) and Quick SQL.

To ensure broad understanding and conciseness, Quick SQL will be used (find Quick SQL under SQL Workshop > Utilities > Quick SQL). The following text will be inserted:

```
CH12_JOB
  job_description vc100

CH12_COMPETENCE
  competence_description vc100

CH12_JOB_COMPETENCE
  id_job num /fk CH12_JOB
  id_competence num /fk CH12_COMPETENCE
  required_level vc10 /check 'assist','use','master', 'tailor','innovate'
```

After entering the above text and pressing **Generate SQL** button the following screen appears (see Figure 12.2):

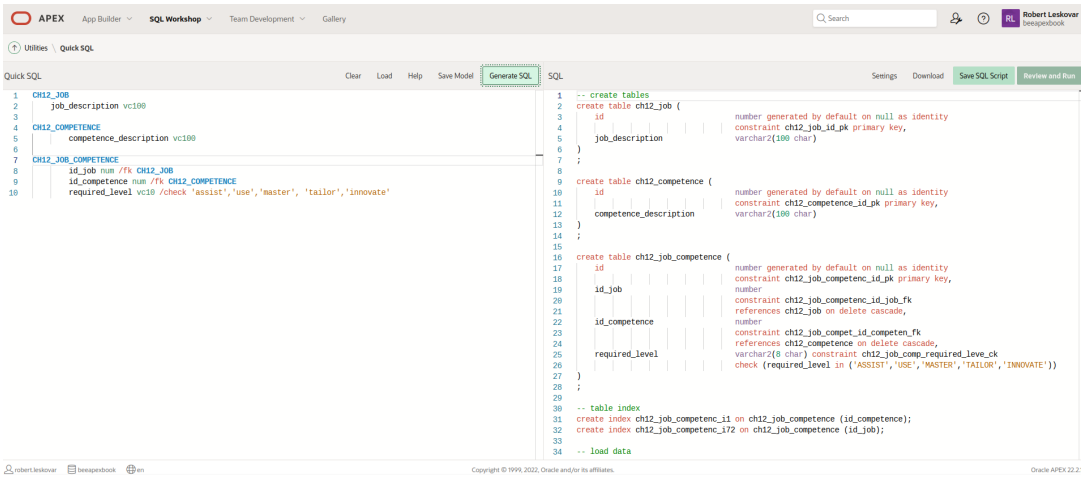


Figure 12.2: Transforming Quick SQL to SQL commands.

Then click **Save SQL Script** button and name the script as CH12CREATE. We can now execute the script. Return to Scripts, and find the Run icon in the row where script CH12CREATE is shown. Click **Run** icon. Three tables and three indexes should be successfully created. You can check the tables in Object browser.

12.2.3 Insert data

In APEX select SQL Workshop > SQL Scripts > Create. Now it is time to write some SQL INSERT commands to fill data into tables. Table CH12_JOB will have seven rows each describing a distinct job. Table CH12_COMPETENCE will have eleven rows describing eleven digital competencies defined by McKinsey's DELTAs (distinct elements of talent). And we will instruct it to insert five competences to job "Junior Apex developer". Let's name script CH12INSERT and write the script (see Figure 12.3).

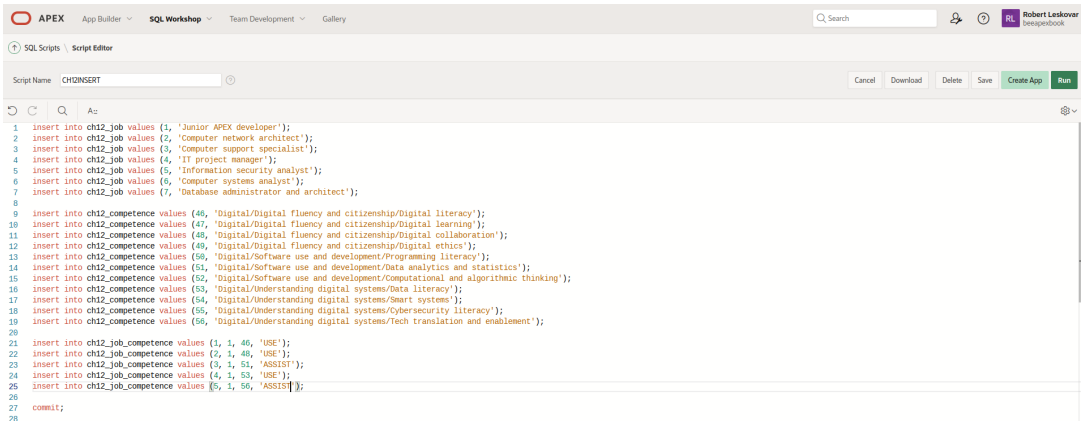


Figure 12.3: Script CH12INSERT insert data in three tables and commit transactions.

Click the **Create** button. Now execute script CH12INSERT. Return to Scripts, click on the **Run** icon in the row where script CH12INSERT is shown. Click the **Run now** button. Twenty-four rows should be successfully created. You can check the content of tables in Object browser too.

12.2.4 Generate application

After populating the tables with test data navigate to SQL Workshop > Scripts and edit script CH12CREATE. Click on the **Create App** button (see Figure 12.4). You can notice that three tables

are listed.

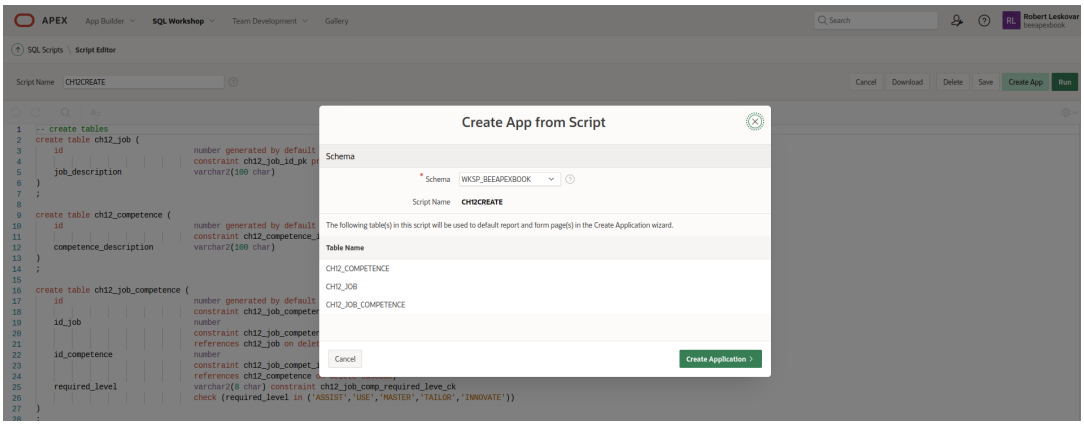


Figure 12.4: Create application from script.

Click the Create Application button. Enter the name of the application (i.e.CH12 Application), click Check all features and then the Create Application button (see Figure 12.5).

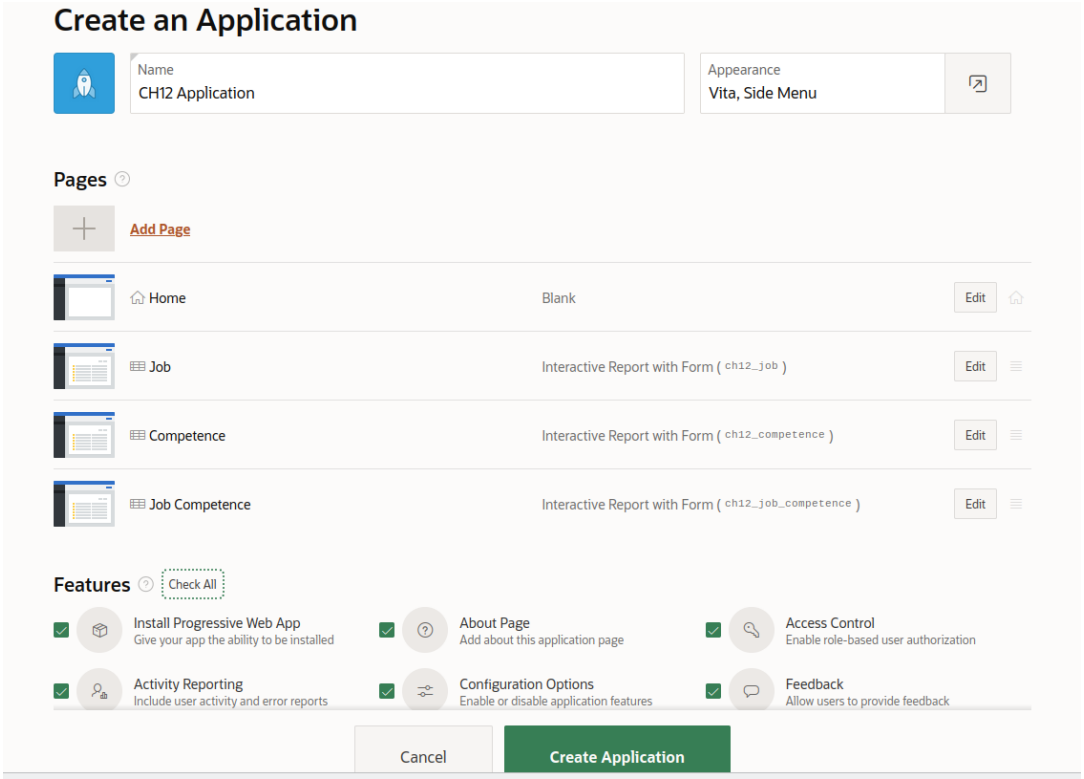


Figure 12.5: Selecting application name and all features.

After generation is finished, we could run the application. In newer versions of APEX, the generator also creates two list of values (CH12_COMPETENCE.COMPETENCE_DESCRIPTION and CH12_JOB.JOB_DESCRIPTION), abbreviated as LOV. List of values are components on the page where one value is displayed and understood by user while the page actually uses corresponding key or code for insert, update or delete. For example: on the page form the last name of the employee is displayed and corresponding employee identification number is stored when the confirmation button is pressed. The generated LOVs are based on tables and no prior

knowledge of SQL is needed. In APEX terminology they are called dynamic since change in the table would produce a new list of values. You can find generated LOVs in Shared Components > List of Values and determine how they are set. In this chapter, we learn how to create dynamic list of values (based on simple SQL query) and static ones based on fixed set of values. By applying list of values (LOV) in application pages we will make the application more error resistant and also more user-friendly. That means that the user can not fill arbitrary values into the form field thus we mitigate typing errors. Navigate to Shared components (see Figure 12.6).

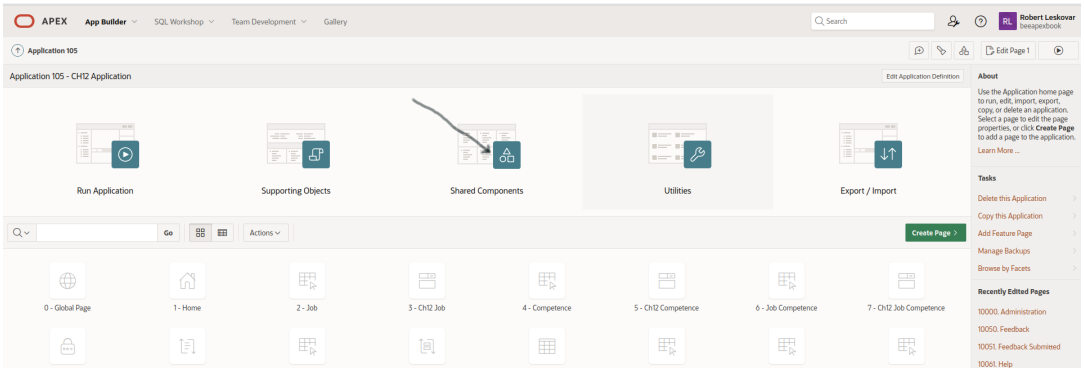


Figure 12.6: Selecting Shared components.

In shared components select List of values (in Other Components) and click Create. We will create list of values from scratch (see Figure 12.7).

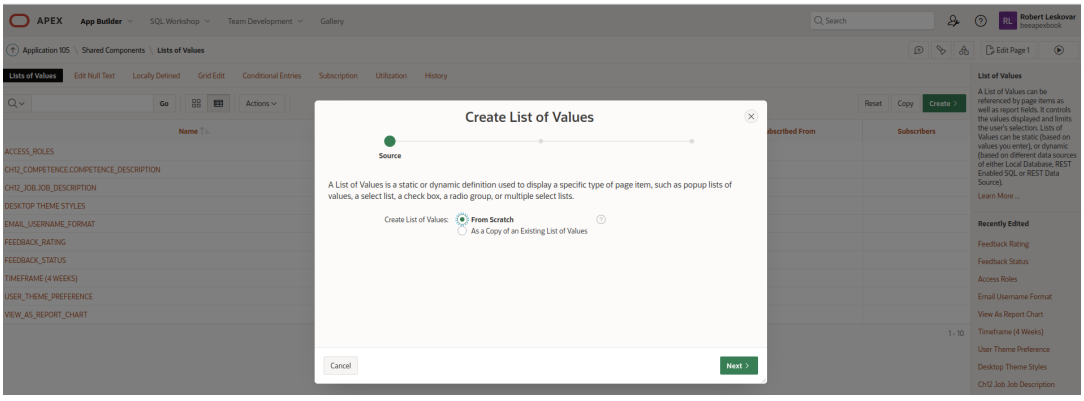


Figure 12.7: Create list of values from scratch.

Name LOV as CH12_LOV_COMPETENCE_DESCRIPTION and select Dynamic type (see Figure 12.8)

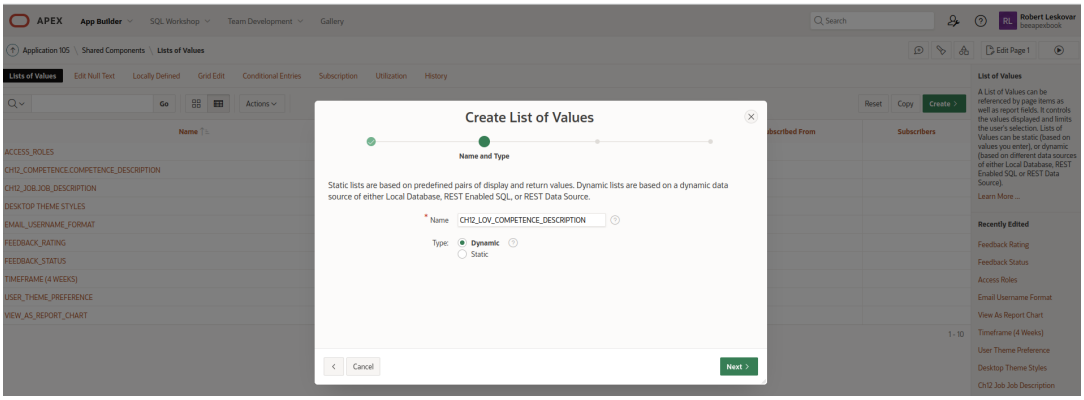


Figure 12.8: Defining name and type: CH12_LOV_COMPETENCE_DESCRIPTION is dynamic.

Select SQL Query as source type and enter query "select competence_description as d, id as r from ch12_competence" as SQL select statement (see Figure 12.9).

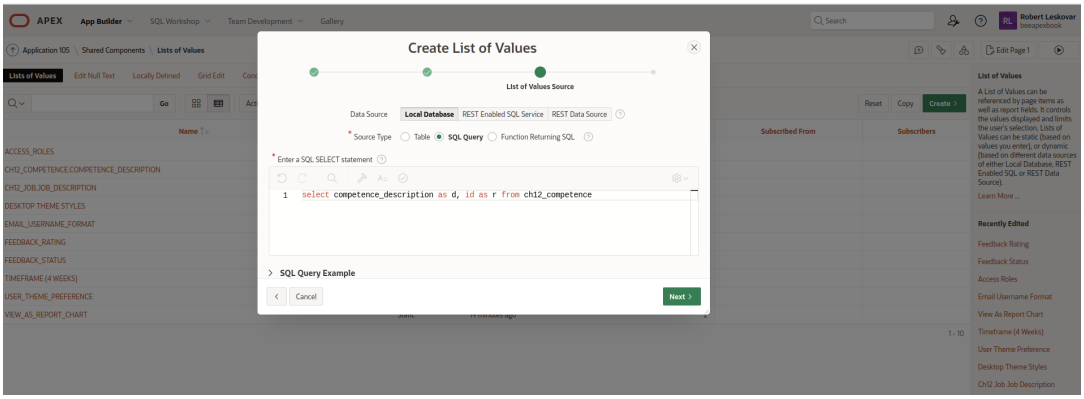


Figure 12.9: Entering SQL SELECT command.

This list of values will display description of competence (display column) on the form page while returning ID (return column). Next LOV will be static, named CH12_LOV_COMPETENCE_LEVEL and created as shown on Figure 12.10 and Figure 12.11.

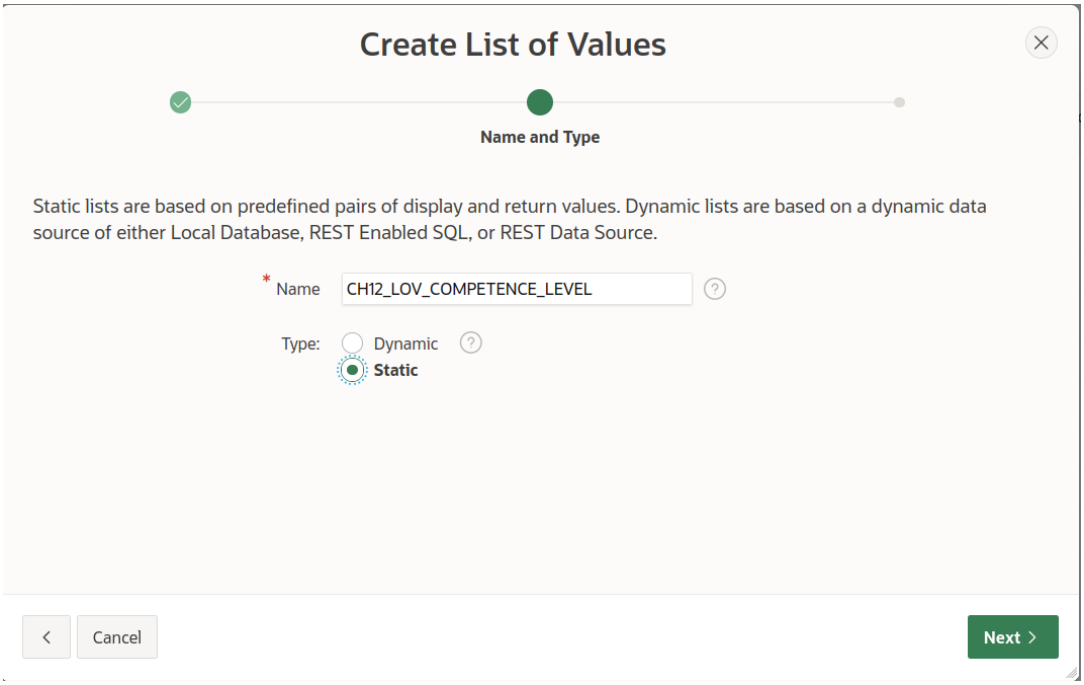


Figure 12.10: Define name and type: CH12_LOV_COMPETENCE_LEVEL is static.

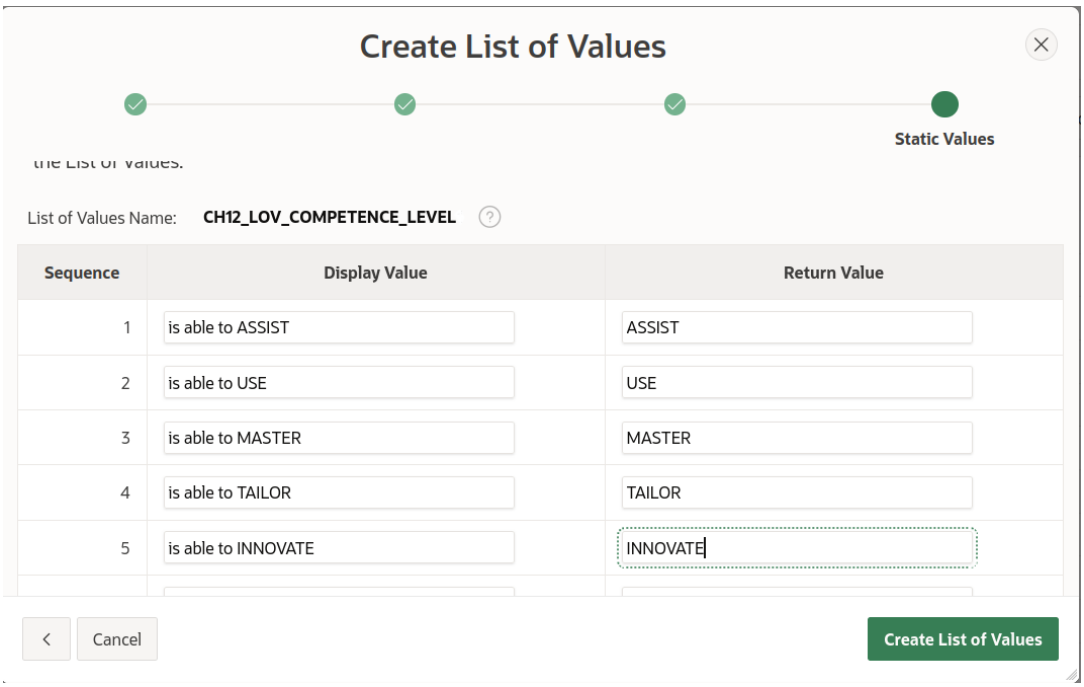


Figure 12.11: Define display and return values: CH12_LOV_COMPETENCE_LEVEL is static.

Navigate to application pages (see Figure 12.12). Select page 7 (Ch12 Job Competences).

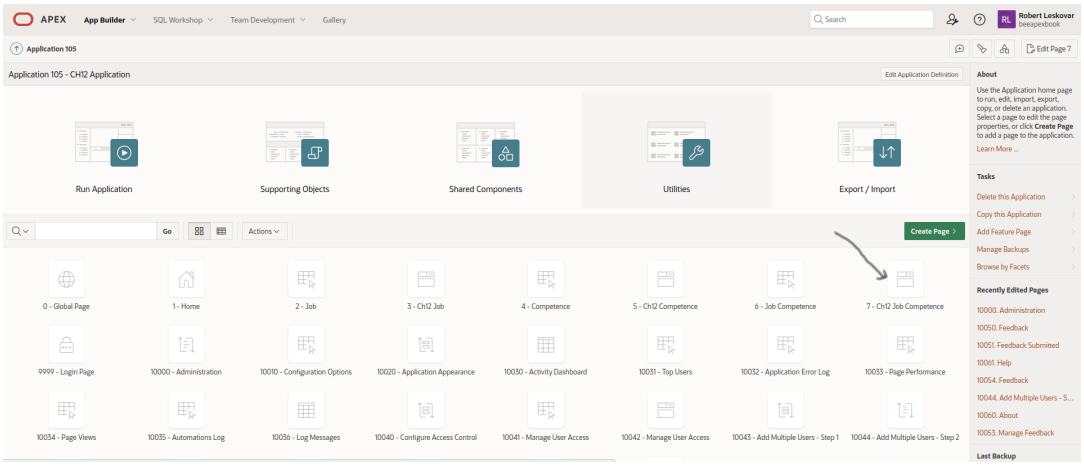


Figure 12.12: Select page 7 (Ch12 Job Competences).

Modify page item P7_ID_COMPETENCE:

- type is "Select list" in Identification
- type is "Shared components" in List of values
- list of values is CH12_LOV_COMPETENCE_DESCRIPTION List of values

Modify page item P7_REQUIRED_LEVEL:

- type is "Select list" in Identification
- type is "Shared components" in List of values
- list of values is CH12_LOV_COMPETENCE_LEVEL List of values

Save the page 7, navigate to page 6 and run application. You should see Figure 12.13.

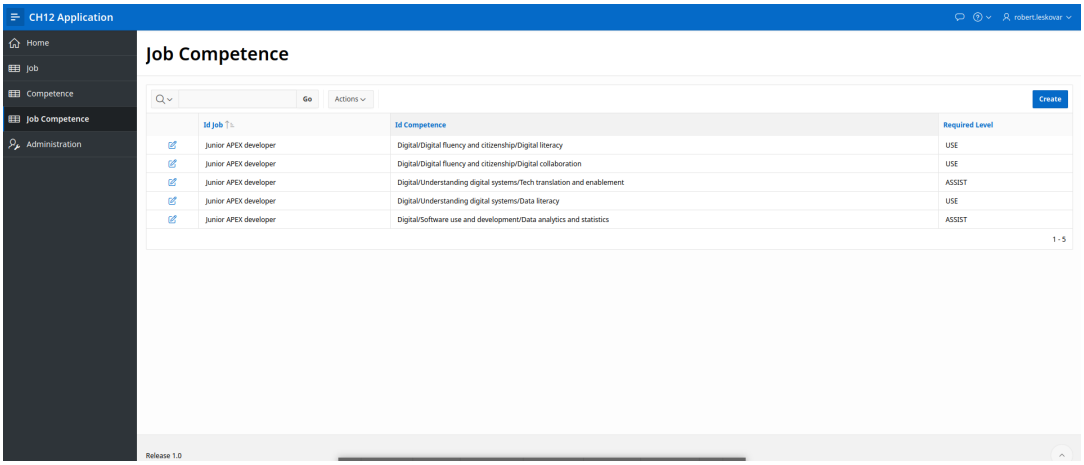


Figure 12.13: Page 7 report (Ch12 Job Competences).

Now edit first entry. Both select lists provide expected data.

12.3 Packaged application

Since we have already created some scripts (CH12CREATE, CH12INSERT), we are able to build packaged application very quickly. Navigate to application and select Supporting Objects (see Figure 12.14).

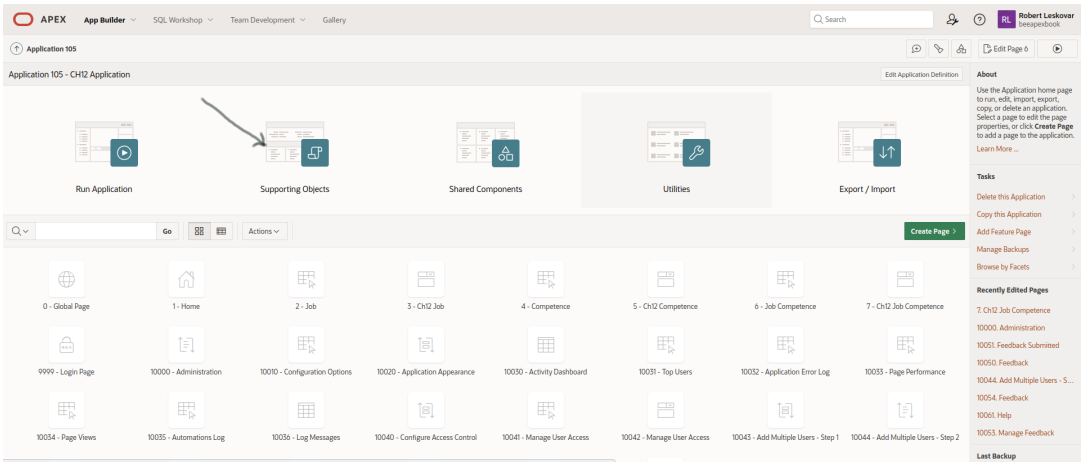


Figure 12.14: Select Supporting Objects.

Now we can set installation (prerequisites, application substitution strings, build options, pre-installation validations, installation scripts and messages), upgrade (upgrade scripts, upgrade message), de-installation (deinstallation script, deinstallation message). See Figure 12.15.

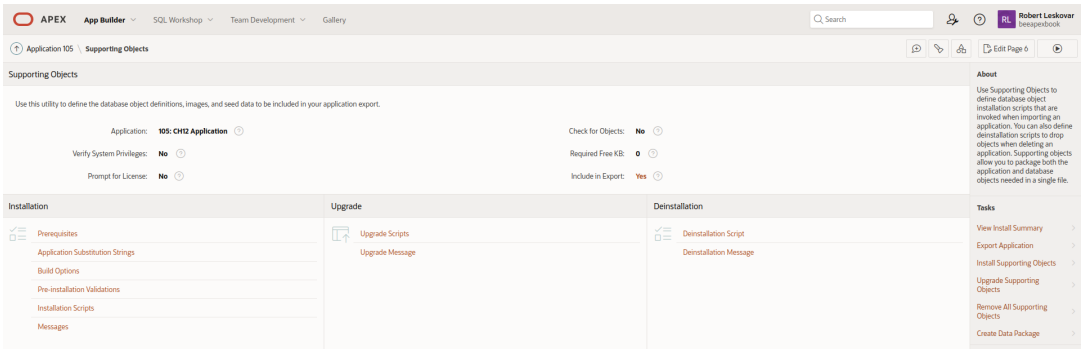


Figure 12.15: Setting prerequisites.

To protect the user from accidentally deleting existing data we will apply check on existence of three tables (see Figure 12.16). Click Apply changes button.

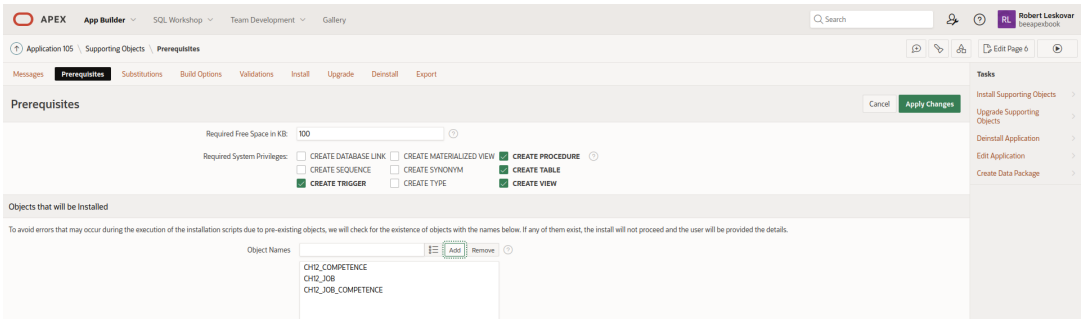


Figure 12.16: Set check on existence of three tables.

In Application Substitution String we will ask if the name should be "BeeAPEX Chapter 12 App" (see Figure 12.17).

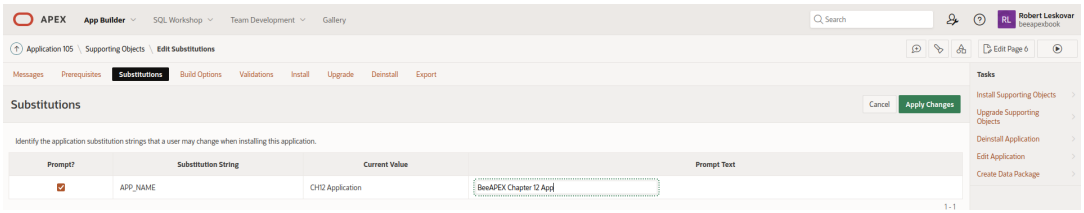


Figure 12.17: Set prompt to rename application.

Click Apply changes button.

For this packaged application we will also apply the following setting in Supporting Objects:

- prerequisites (see Figure 12.16)
- application substitution substrings (see Figure 12.17)
- installation scripts: since we already prepared scripts CH12CREATE and CH12INSERT we will use their content (see Figure 12.18)
- deinstallation scripts (see Figure 12.19)
- set messages: wellcome message to "Chapter 12: ...", licence message to "CC BY", installation success message to "Success!", installation failure message to "Failure", deinstallation message to "Application CH12 deinstalled"



Figure 12.18: Set installation scripts.

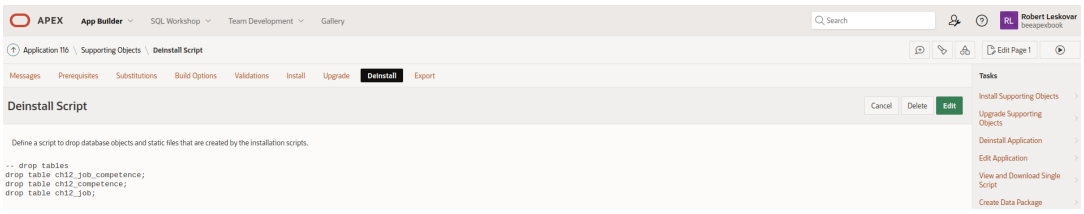


Figure 12.19: Set deinstallation scripts.

Now export the packaged application with wizard (see Figure 12.1). You can accept all export settings proposed by APEX. Save exported application on your computer and remember filename. Log into new workspace. Click import wizard in new workspace and drag filename into drag and drop zone (see Figure 12.20).

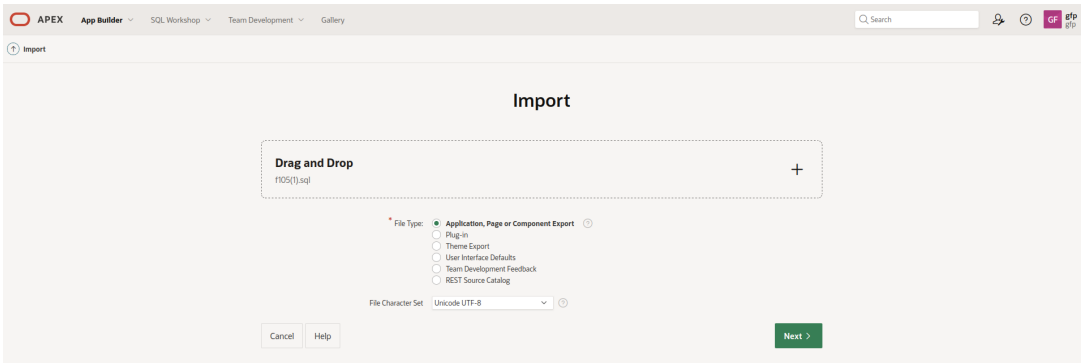


Figure 12.20: Import the application into another workspace.

Proceed with click on Next button (also to confirm file import) and click Install application button. Confirm installation of supporting objects with Next button, accept licence (see Figure 12.21), rename application to "Imported CH12 Application" (see Figure 12.22) and click Install button. Do not run the application yet because you have to set permissions for a new user separately. Users and user permissions are not exported for security reasons. Just one remark: to migrate users and permissions we could deep dive into APEX API and write installation and deinstallation scripts. Navigate to shared components in imported application. Select application access control and give currently logged in user appropriate permissions. In our case we will add APEX user GFP administrator, contributor, and reader role (see Figure 12.23). Confirm with click on Add Assignment button. Now run application as user with given permissions.

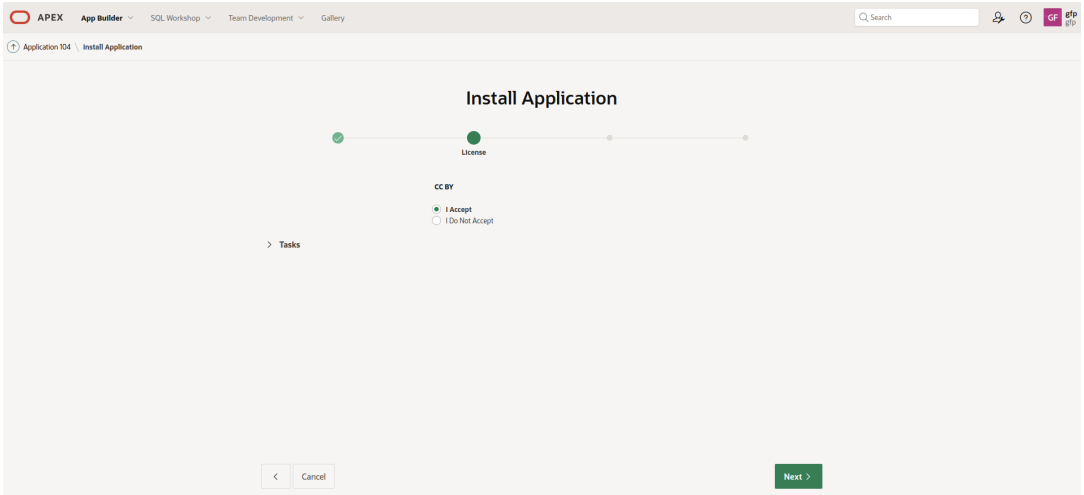


Figure 12.21: Licence agreement.

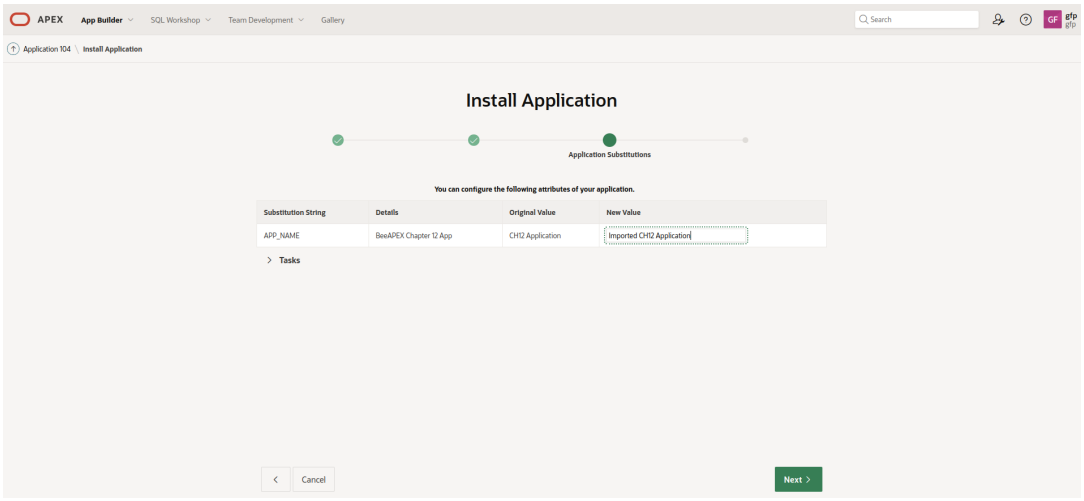


Figure 12.22: Rename imported application.

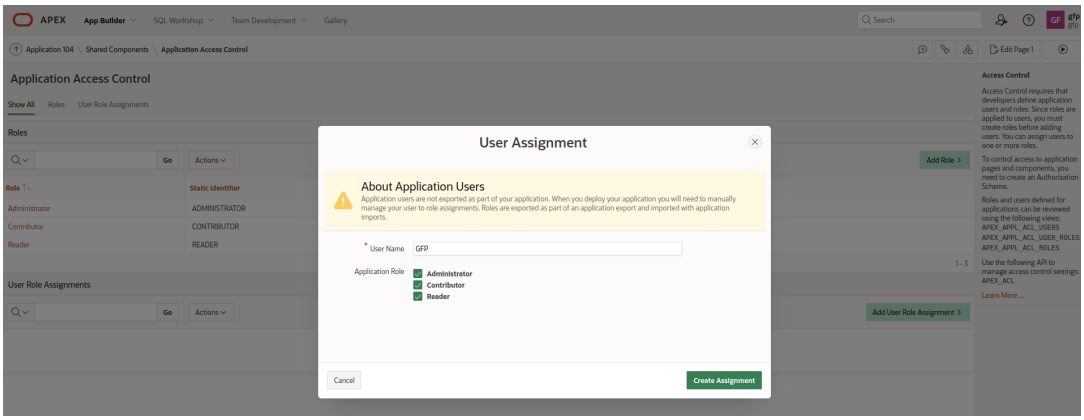


Figure 12.23: Adding a role to user.

12.4 Multilingual application

This topic will cover only one aspect of multilingual applications - strings in application pages such as reports and forms. The primary reasons for adapting an application to a specific language are: a) users are fluent in other languages only and b) there are requirements imposed by organizations or states to support more than one language. The scale of adaptation may be limited to translation of page item labels, application messages, APEX internal messages, number and date formats or as complex as translating the text stored in database tables (i.e., status, level, grade). Specific cultural and linguistic context may cause pure string translation to generate funny, offensive or non-competent content. So, localization of the application means much more than just translation. APEX has a lot of possibilities to apply translation (based on primary application language, browser, application preference, item, session). The following part provides the simplest instructions to implement multiple languages in application by changing application primary language. The steps are:

- Navigate to Shared components > Globalization > Application Translations > Define application languages: click Create button; for each language set unique integer and value (i.e. add two digits to application number like 10801 for sl; 10802 for hr; 10803 for de-at; 10804 for el; 10805 for sk; 10806 for pl). Figure 12.24 shows all languages set.
- Navigate to Shared components > Globalization > Application translations > Seed translatable

text. Select all languages as shown in Figure 12.25. Click Seed button and wait until seeding is finished.

- navigate to Shared components > Globalization > Application Translations > Download XLIFF translation files; we can choose to download all translatable elements or only those elements requiring translation. Example in Figure 12.26 shows export of Page 2, Slovenian language and elements requiring translation.
- edit exported file in preferred editor (Notepad++, Kate, Sublime etc.). For demonstration purposes we changed only a few "target" tagged strings: in lines 48, 52, 56, 60, 72 and 76 (see Figure 12.27). Save changes.
- navigate to Shared components > Globalization > Translate application > Apply XLDIFF translation files > Upload Files. Select the file (see Figure 12.28) and click Upload. Choose the proper language in "Apply to Translation" (see Figure 12.29). Click Apply checked. Click Publish.
- navigate to Shared components > Globalization > Translate application > Publish translations. Select all languages that you translated and click Publish (see Figure 12.30). Wait until you receive message that application is successfully published.
- navigate to Shared components > Globalization > Globalization attributes. Change Application Primary Language to new translation (see Figure 12.31). Click Apply changes.

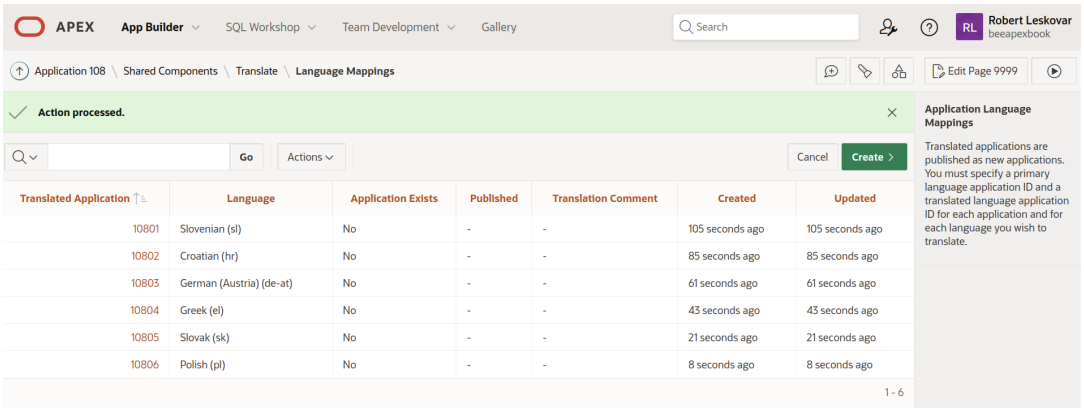


Figure 12.24: Defined languages for translation.

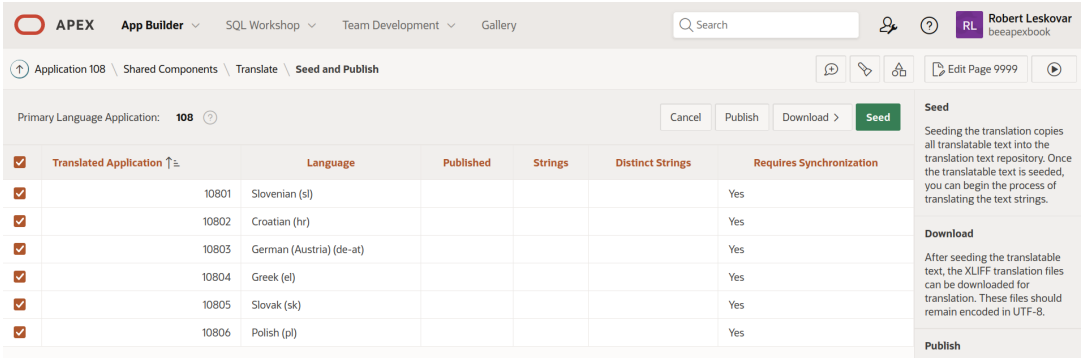


Figure 12.25: Seed translatable text.

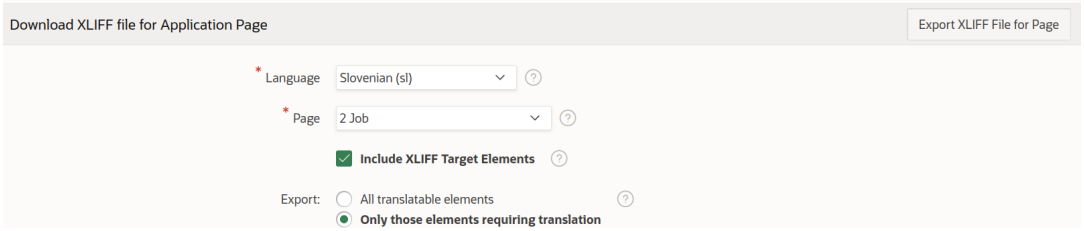


Figure 12.26: Export strings for particular language and page.

```
45  </trans-unit>
46  <trans-unit id="S-13-23279527479873043-108">
47  <source>Create</source>
48  <target>Ustvari</target>
49  </trans-unit>
50  <trans-unit id="S-20-23278046036873035-108">
51  <source>Ch12 Job</source>
52  <target>Ch12 Del. mesto</target>
53  </trans-unit>
54  <trans-unit id="S-20-23280741017873048-108">
55  <source>Breadcrumb</source>
56  <target>Drobtina</target>
57  </trans-unit>
58  <trans-unit id="S-143-23278046036873035-108">
59  <source>Job</source>
60  <target>Del. mesto</target>
61  </trans-unit>
62  <trans-unit id="S-146-23278230062873035-108">
63  <source>The maximum row count for this report is #MAX_ROW_COUNT# rows.  Please apply a filter to reduce the num
64  <target>The maximum row count for this report is #MAX_ROW_COUNT# rows.  Please apply a filter to reduce the num
65  </trans-unit>
66  <trans-unit id="S-147-23278230062873035-108">
67  <source>No data found.</source>
68  <target>No data found.</target>
69  </trans-unit>
70  <trans-unit id="S-149-23278547600873039-108">
71  <source>ID</source>
72  <target>Šifra</target>
73  </trans-unit>
74  <trans-unit id="S-149-23278989319873042-108">
75  <source>Job Description</source>
76  <target>Opis delovnega mesta</target>
77  </trans-unit>
78  </body>
79  </file>
80  </xliff>
```

Figure 12.27: Translation of "target" tagged strings in lines 48, 52, 56, 60, 72 and 76.

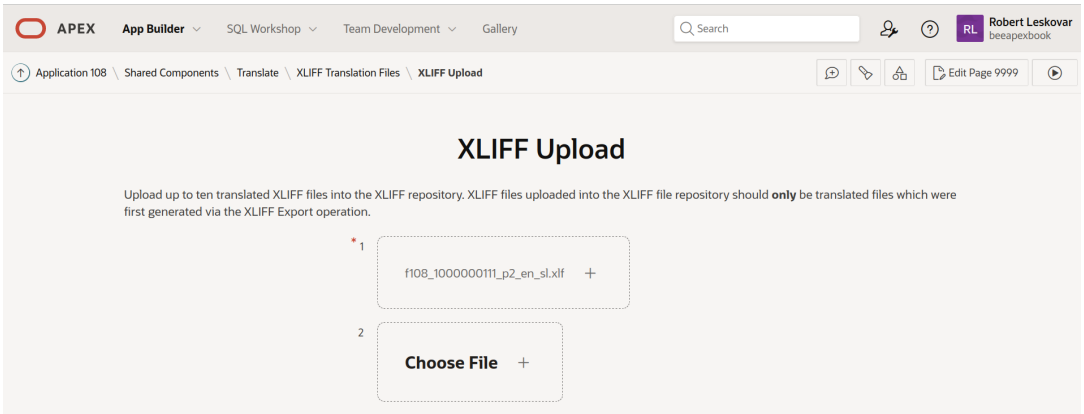


Figure 12.28: Uploading XLDIFF translation files.

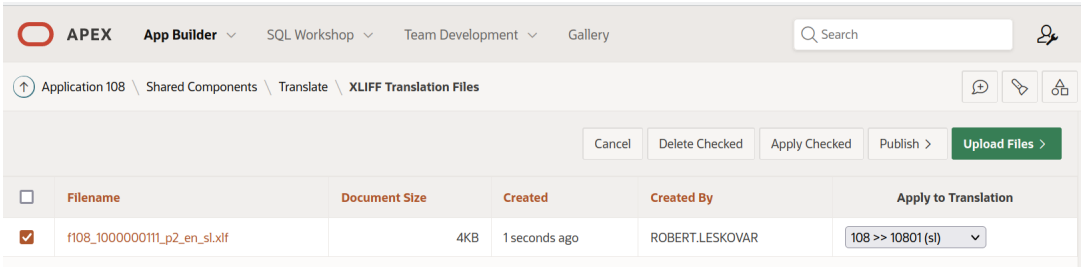


Figure 12.29: Applying changes and publishing.

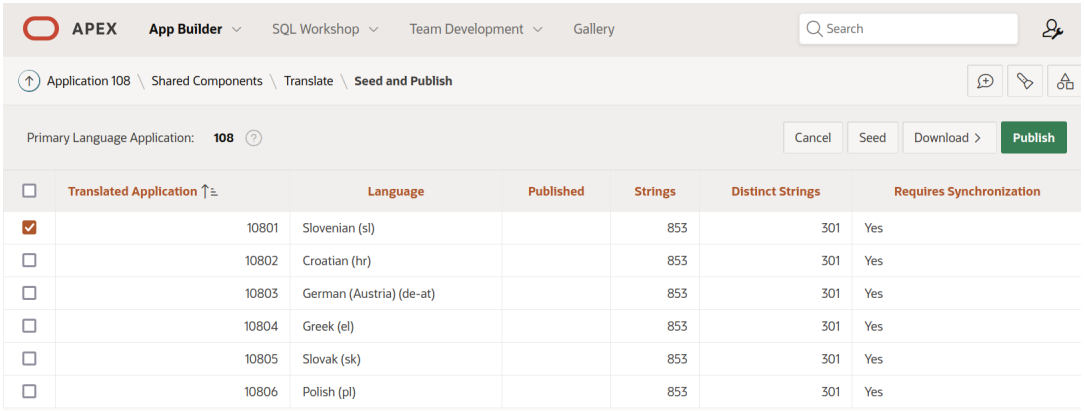


Figure 12.30: Final publishing of the application translation.

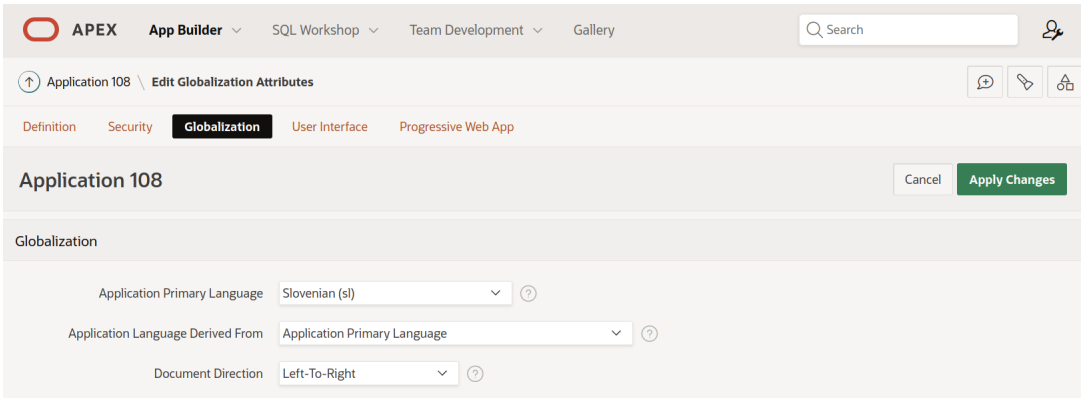


Figure 12.31: Setting application primary language.

Now check Page 2 in translated application (see see Figure 12.32).

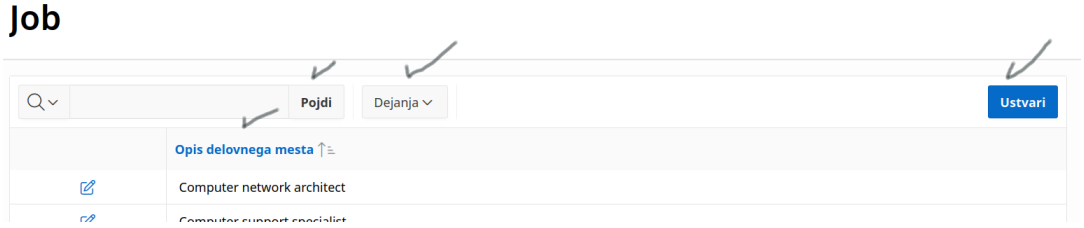


Figure 12.32: Translated page.

Concluding remarks

Development usually takes place in a test environment. When migrating the approved application to a new workspace, packaged applications are very convenient since they reduce time for migration. For security reasons eventual users and their roles are not migrated. The authorization of users in a new workspace is a serious task. This chapter also provided basic instructions for multilingual applications. It should be noted that for large and critical mission applications the translation process must be supported by capable automatic translation tools as well as language and culture human specialists.

12.5 Supplementary learning material

You can find the following supplementary learning material:

- exported applications
- video guides

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 1 > Chapter12 in the Scripts section and video guides in Collection of video guides. Material for short courses in in Short courses section.

12.5.1 Exported applications

There are three related applications:

- initial application. Check if tables starting with CH12 exist. If tables exist you can drop them with CH12DROP script. Then import application in file CH12_Application_initial.sql. After you import application you must define tables. Execute script CH12CREATE and optionally populate tables with CH12INSERT script. Navigate to imported application > Shared Components > Application Access Control. Add User Role Assignment for currently logged user. Allow all actions with Administrator role, select, insert, update and delete with Contributor role and select only with Reader role.
- packaged application. Check if tables starting with CH12 exist. If tables exist you must them with CH12DROP script. Then import application in file CH12_Application_packaged.sql. Navigate to imported application > Shared Components > Application Access Control. Add User Role Assignment for currently logged user. Allow all actions with Administrator role, select, insert, update and delete with Contributor role and select only with Reader role.
- packaged multilingual application. Check if tables starting with CH12 exist. If tables exist you must drop them with CH12DROP script. Then import application in file: CH12_Application_packaged.sql. Navigate to imported application > Shared Components > Application Access Control. Add User Role Assignment for currently logged user. Allow all actions with Administrator role, select, insert, update and delete with Contributor role and select only with Reader role.

12.5.2 Video guides

The following video guides are provided:

- create initial application
- import initial application
- copy initial application to packaged and creating packaged application
- import packaged application
- import packaged multilingual application

12.6 Questions

1. What is packaged application and what is their benefit?

2. How do you export packaged application to include database object definition and data for tables?
3. What is the role of XLDIFF file in creating multilingual application?

12.7 Answers

1. Packaged application is a pre-built application that can be installed and configured in an Oracle APEX workspace. The main benefit of a packaged application is the easy and smooth migration as well as immediate availability of prepared data.
2. Packaged application is to be exported with wizard. One can include scripts which define database objects (like tables) and instruct inserting data into tables. The list of scripts and the order of execution of these scripts is set in wizard. Packaged application can also contain de-installation scripts.
3. XLDIFF file is tagged and therefore provide easy access to target values for translation strings. These files can contain an entire application or just a specific page in a selected language. File with translated target strings is imported into application. Translation repository needs to be published to be available to end-user.

Constructing application in APEX

13 Intranet news for employees 225
R. Leskovar, U. Rajkovič, A.Baggia . . . . . .

14 GreenDi - Catalog of plants 235
Vjeran Strahonja, Dijana Oreški, Darko Andročec, Ana Kutnjak . . . . . .

15 GreenDi - User Authorisation and Management 243
Vjeran Strahonja, Darko Andročec, Ana Kutnjak, Larisa Hrustek . . . . . .

16 Small Innovation System 251
R. Leskovar, U. Rajkovič, A.Baggia . . . . . .

17 Business process management 261
R. Leskovar, U. Rajkovič, A.Baggia . . . . . .

18 GreenDi – Exchange of Plants and Seeds 291
Vjeran Strahonja, Dijana Oreški, Darko Andročec, Ana Kutnjak . . . . . .

19 Book review management system . . . 297
A. Kutnjak, L. Hrustek, A. Baggia and R.Leskovar . . .

20 Bill-of-material and cost calculation . 309
R. Leskovar, U. Rajkovič, A.Baggia . . . . . .

21 Nutrition and diet management 325
A. Angeioplastis, G. Myllis, A. Tsimpiris, D. Varsamis . .

22 Office Hours Scheduling 345
J. Mańko, M. Sońta, R. Leskovar . . . . . .

23 Telco case 365
Veronika Šalgová, Jozef Kostolný, Michal Mrena, Michal Kvet, Miroslav Potočár . . . . . .

24 Car rental case 389
A. Angeioplastis, G. Myllis, A. Tsimpiris, D. Varsamis . .



13. Intranet news for employees

R. LESKOVAR, U. RAJKOVIČ, A.BAGGIA

13.1 Business view of the case

A medium sized company with seven hundred employees is an established player in a global industry of top sports and leisure equipment, and advanced composites. The headquarters are in Slovenia with subsidiaries are in the USA, Canada, Germany, and Japan. Their annual revenue is around 90 million euros. The majority of employees are located in Slovenia in one location. The company had four divisions, and each division has several departments. In the past they used an internal semi-annual bulletin to communicate various achievements, success stories, announcement of new products and similar. Then they built a static (pure HTML) intranet page to disseminate important news. But this opportunity to empower employees and strengthen their commitment to the company was short lived the idyllic garden has turned into problem.

13.2 Problem definition

The process of publishing was loosely defined. IT employee received the request to publish news from the issuers - department and division managers. The request was communicated by mail, phone or personally, loosely and often with scarce material (text, pictures, video). The start and stop date of the news was not determined. The frequency of requests to publish news increased to approximately one hundred news items per month and some requests overlapped (two or more department managers had similar or the same topics). An IT administrator had to resolve the issues between and with managers. Start and stop dates for news appearance was so often left on the IT administrator's shoulders, that it became a source of tensions for IT employees an managers alike. Also the managing of the static page became time consuming and extremely error prune. The solution required both, process modification and a new publishing platform.

Process modification and a new publishing platform . A low code application development environment was selected to develop the new publishing platform. The company already uses Oracle database, therefore Oracle APEX is selected as the development environment. Three types of platform users are identified. First the administrator, who has access to all data and privileges to manage users and roles. In addition to the basic employee data, we also track their employments in the departments and their locations. Each employee has a specific role defined, which can change over time. The second type of a user is publisher, who does not have the right to manage employee data and define privileges, but has the privilege to publish news. The third type of user is

viewer/reader of the news, who has the privilege to read internet news.

Requirement specification Based on the modified process, the platform will enable users to:

- Reader/Viewer: login, access the dashboard, view news and attachments to news
- Publisher: login, access the dashboard, view and publish news and add attachments to published news
- Administrator: login, access the dashboard, view and publish news, view and manage departments, roles, employees, employee roles, manage application

13.3 Use cases

13.3.1 Narrative description of use case

The publishing platform enables access to three different types of employees: administrator, reader/viewer, and contributor/publisher, each having different privileges on the intranet portal. Publisher can publish news, reader can read news and administrator can manage the platform. Each one of these use cases requires a user to sign-in to the intranet platform. See Tables 13.1, 13.2 and 13.3,

13.3.2 Semi-structured description

We can summarize three distinct user stories or use cases: publishing, reading and managing.

Adding an attachment to the news is actually an extension to UC described above and we could describe it as new use case. For the sake of readability we proceed with other two main use cases, however the application will have a feature to add attachments to the news.

13.3.3 Use case diagram

The above story is depicted on use case diagram Figure 13.1.

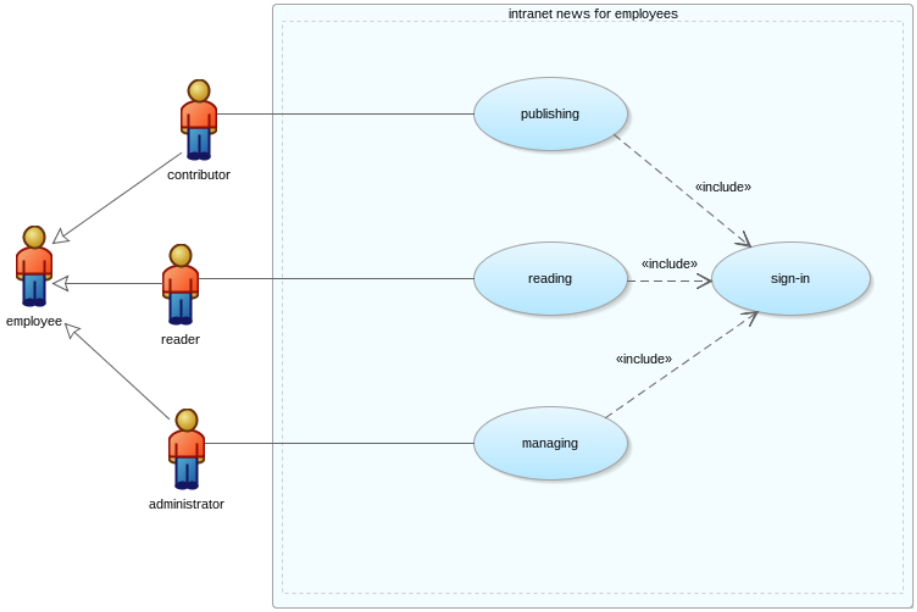


Figure 13.1: Use case diagram.

Again for the sake of simplicity extension use case (adding attachment to the news) is skipped in Figure 13.1.

Table 13.1: Use case description: publishing internet news.

Keyword	**Value**
ID:	*Ch13-01*
Title:	*Publish news on intranet portal*
Description:	*Responsible person from HR department, with a publisher role defined, uses intranet APEX portal to publish news. Each news has start and end date. Default start date is the day of publishing.*
Primary Actor:	*Employee with publishing role*
Preconditions:	*The employee has to be listed in the employees table and the user has to be added to the CH13 Publisher role. Access to the web application has to be enabled.*
Postconditions:	*After successful publishing of news, the news and its attachment are available to other intranet portal users.*
Main:	*Scenarios*
Success Scenario:	1. *Open the web browser and sign-in to the intranet application* 2. *Select Publish news* 3. *Enter the news Title, description and end-date* 4. *Confirm creation of the news* 5. *Add attachment(s)* 6. *Enter attachment details* 7. *Confirm adding the attachment* 8. *Review the published news*
Extensions:	• *1a. Sign-in fails* • *1a* Extend:* • *1a1. Show error message* • *1a2. Open sign-in window* • *4a. Datatype error* • *4a* Extend:* • *4a1. Show error message* • *7a. Datatype error* • *7a* Extend:* • *7a1. Show error message*
Frequency of Use:	*The publishers publish approximately 1000 news per year, average 5 per day*
Status:	*Finished*
Owner:	*Employee with publishing role*
Priority:	*moderate*

Table 13.2: Use case description: reading intranet news

Keyword	**Value**
ID:	*Ch13-02*
Title:	*Read news on intranet portal*
Description:	*Intranet user reads news*
Primary Actor:	*Intranet user*
Preconditions:	*The employee has to be listed in the employees table and the user has to be added to the CH13 Viewer role. Access to the web application has to be enabled.*
Postconditions:	-
Main	*Scenarios*
Success Scenario:	1. *Open the web browser and sign-in to the intranet application* 2. *Select View news* 3. *Select Download to view the attachment*
Extensions:	• *1a. Sign-in fails* • *1a* Extend:* • *1a1. Show error message*
Frequency of Use:	*Employees read news on daily basis.*
Status:	*Finished*
Owner:	*Intranet user*
Priority:	*low*

13.4 Data model

13.4.1 Narrative description of data model

There are six entities in the logical data model. The CH13 DEPARTMENT entity has three attributes: ID, name and location of the department. Each department can have many employees. Basic data about employees is stored in the CH13 EMPLOYEE entity: ID, first and last name, birth date and email. An employee can be a manager. Each employee has only one manager, whilst a manager can manage several employees. Each employee can has a role, with roles defined by ID, name and description. For each employee and each role, a start and end date are defined. Over time, employee can have several roles, also more employees can be assigned to one role. Nevertheless, one definition of an employee role can not be transferred to another employee or role. An employee can publish more news over time. Each news is identified by ID, title, description, start date and end date. Each news can have more attachments. An attachment is identified with ID, file name, mime type, date created and content.

13.4.2 Logical data model

The above story is depicted on logical data model Figure 13.2.

Table 13.3: Use case description: managing the intranet portal.

Keyword	**Value**
ID:	*Ch13-03*
Title:	*Managing the intranet portal*
Description:	*Manage the intranet portal*
Primary Actor:	*Intranet portal administrator*
Preconditions:	*The employee has to be listed in the employees table and the user has to be added to the Administrator role. Access to the web application has to be enabled.*
Postconditions:	*Users, roles and employees are ready to be used in the intranet portal.*
Main	*Scenarios*
Success Scenario:	1. *Open the web browser and sign-in to the intranet application* 2. *Add and manage departments data* 3. *Add and manage roles data* 4. *Add and manage employees and their roles* 5. *Publish news* 6. *View news* 7. *Administration*
Extensions:	• *1a. Sign-in fails* • *1a* Extend:* • *1a1. Show error message* • *1a2. Open sign-in window* • *2a. Datatype error* • *2a* Extend:* • *2a1. Show error message* • *3a. Datatype error* • *3a* Extend:* • *3a1. Show error message* • *4a. Datatype error* • *4a* Extend:* • *4a1. Show error message* • *5a. Datatype error* • *5a* Extend:* • *5a1. Show error message*
Frequency of Use:	*The frequency of use depends on new employees and changes in the privileges scheme. Approximately 5 times per week.*
Status:	*Finished*
Owner:	*Intranet portal administrator*
Priority:	*high*

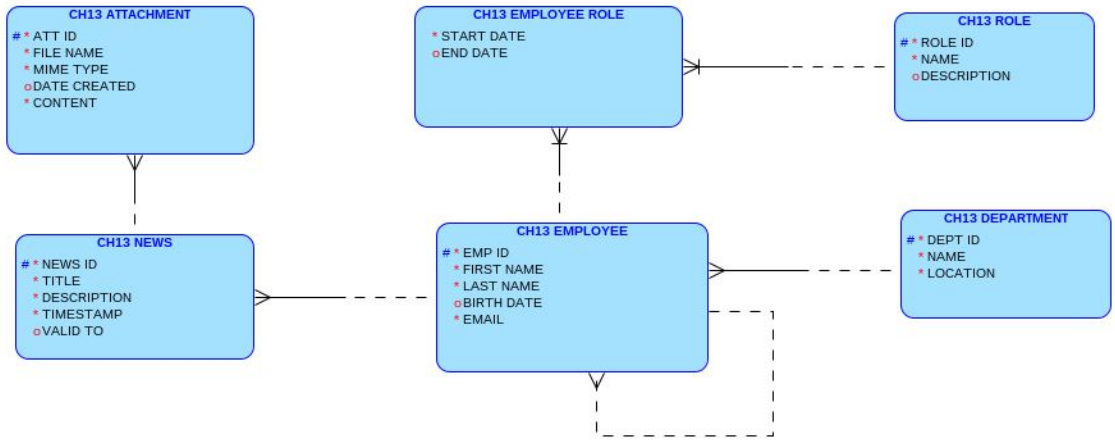


Figure 13.2: Logical data model.

13.4.3 Relational data model

Automatic transformation from logical data model to relational data model in Oracle SQL Data Modeler is provided by function *Engineering to relational*. The result is shown on Figure 13.3.

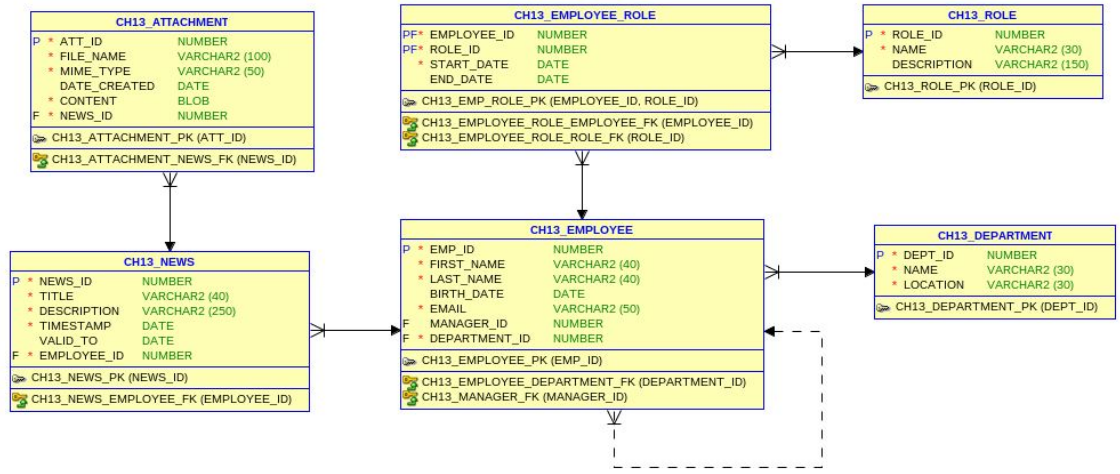


Figure 13.3: Relational data model.

Oracle SQL Data Modeler also generates SQL script for table, sequence and trigger creation. Select all tables on relational model and use function *File* > *Export* > *DDL File* to get a script like this:

```
CREATE TABLE CH13_NEWS (
NEWS_ID NUMBER NOT NULL
...
);

CREATE TABLE CH13_DEPARTMENT (
DEPT_ID NUMBER NOT NULL
...
);
```

Now it is time to generate tables in Oracle database. We can import the generated script and execute it in APEX.

13.5 Application interfaces

In the following figures, application interfaces are presented for all three roles. First, the dashboard for Contributor (Publisher) is presented in Figure 13.4.

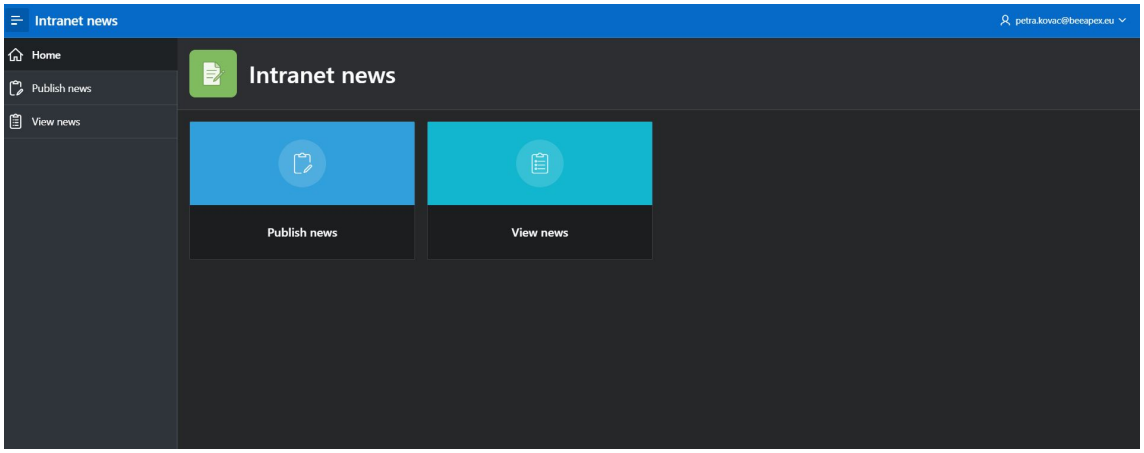


Figure 13.4: The Contributor (Publisher) dashboard.

The page for publishing news with attachments and its details is presented in Figure 13.5.

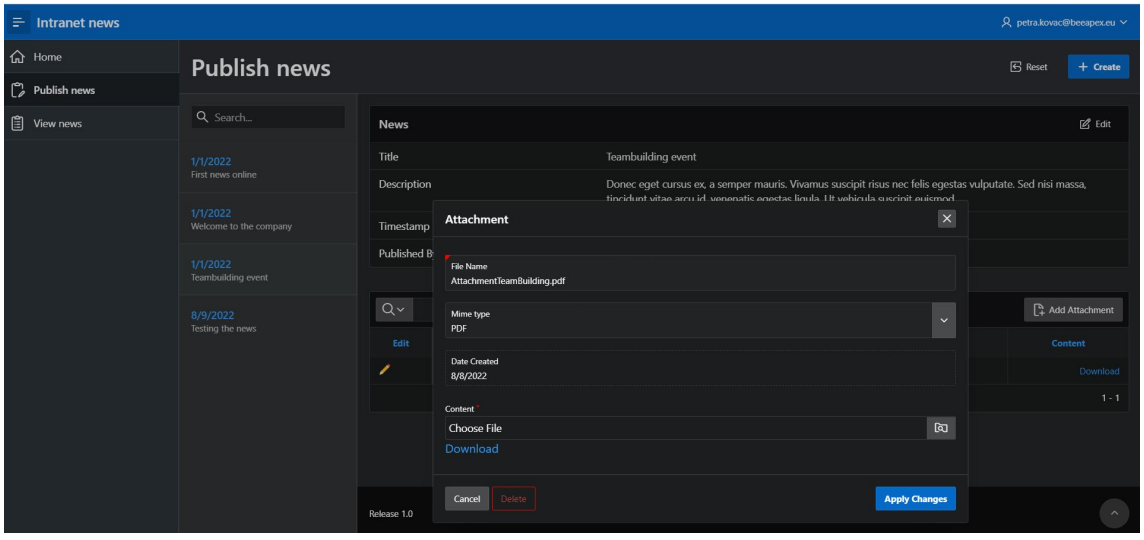


Figure 13.5: Publishing news with attachments.

The Viewer (Reader) role has the most limited dashboard. Only the Home and the View news is available for now, as presented in Figure 13.6.

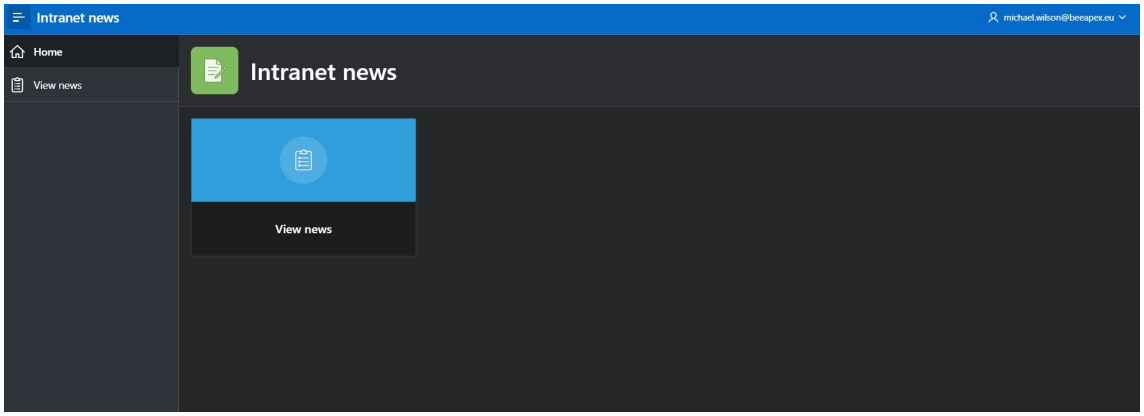


Figure 13.6: Viewer (Reader) dashboard.

The main feature for the Viewer (Reader) is to access news (see Figure 13.7).

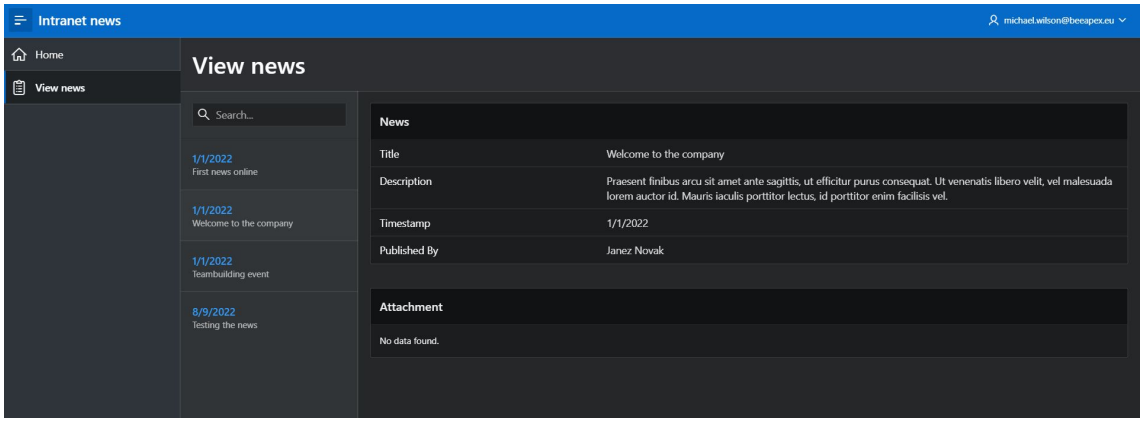


Figure 13.7: Viewer (Reader) access to news.

In addition to viewing and publishing news, the administrator can manage departments, roles and employee with roles data, as depicted in Figure 13.8. Also, the administration page is available, where user management is possible.

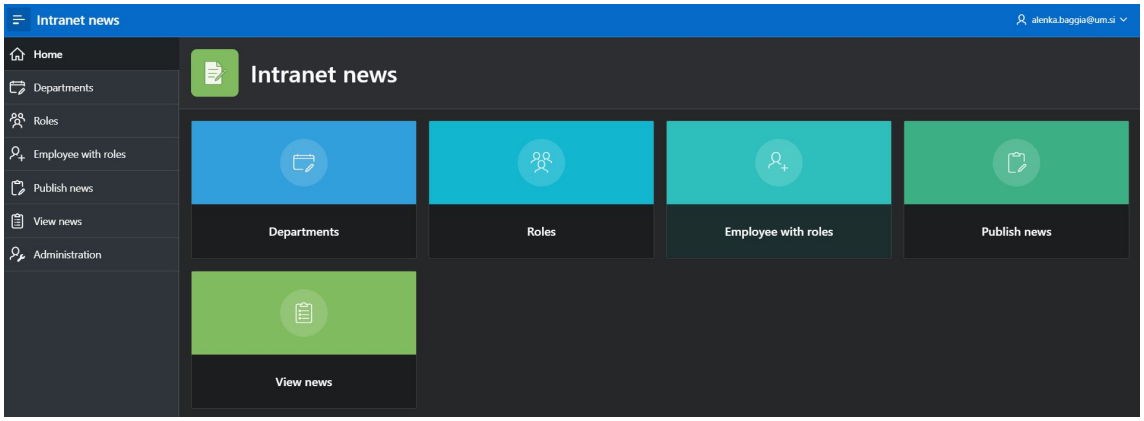


Figure 13.8: Administrator dashboard.

The page for editing employee roles data is presented in Figure 13.9. This page is only available to the Administrator of the intranet portal.

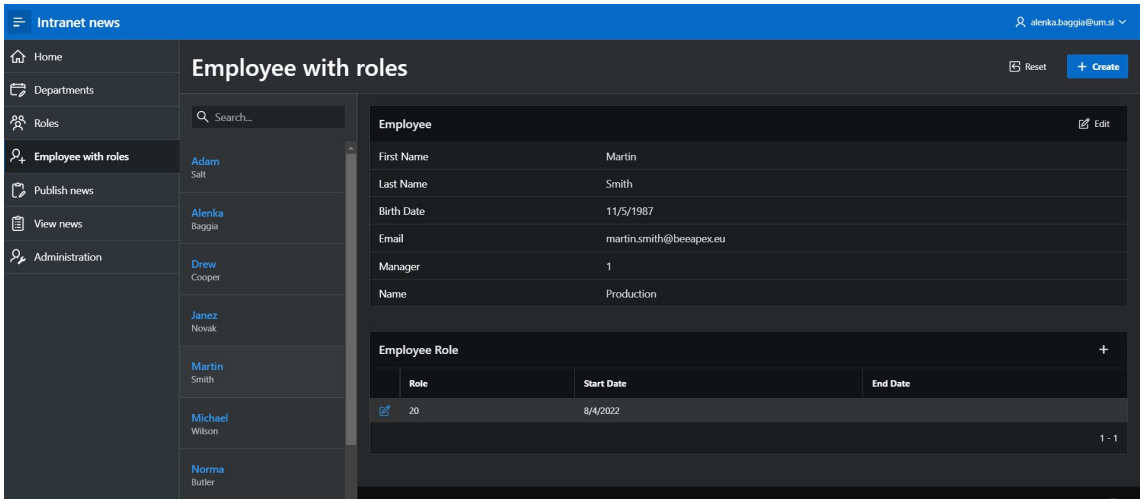


Figure 13.9: Editing employee role by Administrator.

13.6 Supplementary learning material

You can find the following supplementary learning material:

- exported application
- scripts for creating, dropping and inserting
- video guide

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter13 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

13.6.1 Exported application

Exported application is packaged. Installation create tables and populate data. De-installation removes all data base objects used in this application.

13.6.2 Video guides

Video guide show every step in application development.

13.7 Questions

1. How would you change logical data model to implement the following requirement: one attachment can belong to more news.
2. How would relational data model change if the requirement (one attachment can belong to more news) is implemented?
3. Which user roles are implemented in Chapter 13 application and why?

13.8 Answers

1. One-to-many relation should be allied in direction from attachment to news.
2. Pair attachment-news in logical model would be transformed into three relational tables: attachment, news and attachment_news. The last one would contain at least two fields (IDs of attachment and news) both serving as a foreign keys.
3. In Chapter 13 the following user roles are implemented: Administrator, Contributor and Reader. These roles are implemented to enable authorized users to do their job and to prevent

unauthorized users to perform unapproved actions.



14. GreenDi - Catalog of plants

VJERAN STRAHONJA, DIJANA OREŠKI, DARKO ANDROČEC, ANA KUTNJAK

14.1 Business view of the case

A group of enthusiasts gathered in the GreenDi nonprofit organization has been launching and implementing various initiatives related to urban gardening, healthy eating, growing old domicile varieties, cooperation with local farmers, etc. So far, they have used social networks in their work. That's why they decided to launch their own GreenDi, a peer-to-peer seed and plant sharing platform, with the aim of promoting a biodiversity and facilitating the activities of their past activities and enable new ones. In order to develop and implement such a platform, a specification of requirements and basic models, including a data model and use cases need to be made.

14.2 Problem definition

The open part of the platform has the following characteristics:

- GreenDi users are not limited to a geographical location but can be global. However, since this is a plant exchange, the formation of local networks related to a certain geographical area is expected.
- The platform is divided into several thematic units such as traditional and indigenous varieties and seeds (vegetables, fruits, medical herbs, indoor and ornamental plants), with the idea of expanding to industrial and other plants.
- The basis of the open part is a catalog of plants, with the name(s) in the user's language, Latin and English, classification of plant (taxonomy), description, habitats, cultivation and use, photos and other basic information.
- Basic information about the plant is open for search and viewing without any registration.
- The GreenDi catalog of plants is linked to several open web pages, databases and external services that contain information related to plants. URL addresses, hyperlinks and other types of tagged data are used.
- The catalog includes plants that are exchanged on the GreenDi platform, but also those that have not yet been exchanged or included in an offer. The catalog also has links to commercial suppliers, with whom members have good experience and who nurture organic farming.

Catalog Management is described by following sentences:

- The Catalog of Plants is managed by the administrators of the GreenDi platform.
- Catalog Management includes entering and changing information about plants.

- Some functionalities of the Catalog Management are made available to users authorized by the administrator (authorized user).
- Advanced functionalities, are available to the administrators, include linking to other pages that contain information about the plant, videos on cultivation, use of plants etc.
- All registered users (Members) can, if they wish, share their opinion about the plant, write a review, add some more information, etc., but that is not part of this functionality.

14.3 Use cases

Use case diagrams describe communication between actors and other systems in environment which we develop. Use cases present functionalities of authorization and user management which are presented on Figure 14.1.

14.3.1 Narrative description of use case

The open part of the GreenDi platform enables any user from any location to view the data of the Catalog of plants, without registration. At the same time, the plants are classified into categories that allow the user to navigate more easily. Name, taxonomy, description, habitats, cultivation and use, photos and other basic information, as well as links to external data sources are kept for each plant.

14.3.2 Semi-structured description

Table 14.1 presents UC.

14.3.3 Use case diagram

This story is depicted on use case diagram 14.1. On the right side of the Use Case diagram is the External Source actor. This actor presents an external system from which GreenDi Catalog of Plants retrieves information about plants, seeds and similar. There are many examples of such plant databases that offer open services or open datasets. That means that the database or data is freely available for use, without any restrictions on access or use. Some examples of online plant databases that provide open services or open data include:

1. The Global Biodiversity Information Facility (GBIF, https://www.gbif.org) provides free and open access to biodiversity data, including data on plants. GBIF allows users to search and access data from a range of sources, including herbaria, museums, and research institutions. It refers to The International Plant Names Index (IPNI), database of the names and associated basic bibliographical details of seeds, plants, ferns and lycophytes (https://www.gbif.org/dataset/046bbc50-cae2-47ff-aa43-729fbf53f7c5).
2. WFO Plant List (https://wfoplantlist.org/plant-list) was launched in May 2021 and provides a user-friendly, citable static list of all plant species. The entire WFO Plant List dataset can also be downloaded as a whole. The data can also be accessed via the WFO Plant List API.
3. The Flora of Italy dataset can be downloaded from the University of Roma website and includes high-resolution images of over 10,000 plant species in Italy.

These are just a few examples of the datasets available that include images of plants and seeds in Europe, but there are many other datasets available that provide similar information. In this example, we will not realize communication with the External Source because it requires more advanced knowledge.

Table 14.1: Use case description: browsing catalog of plants

Keyword	**Value**
ID:	*ch14-01*
Title:	*Browse catalog of local plants*
Description:	*Catalog of local plants is publicly accessible for all interested persons. The application enables searching with filters on all data base fields.*
Primary Actor:	*Any person*
Preconditions:	*Basic information about the plant is open for search and viewing without any registration.*
Postconditions:	*If the Browse catalog of plants is called from another use case (extend), the ID of the selected plant is transferred.*
Main	*Catalog of plants*
Success Scenario:	*First scenario, Catalog of plants:* 1. *Any user on the home page opens the Browse catalog of plants form, which is a Search Form, based on the principle of Query by Example (QBE).* 2. *Search is possible by the following data: a) Thematic unit - list of codes (all by default or some specific name specific, such as vegetables, fruits, medicinal herbs, indoor and ornamental plants, etc.); b) Type of plant - hierarchically organized classification of plant (all by default or some specific name), c) Name (s) in the user's language, d) Latin name e) English name (s)* 3. *The search result is displayed in Tabular Form.* 4. *It is possible to sort the Tabular Form by columns.* 5. *By choosing a row, it is possible to open the form with the Details of the Plant Form, that contains: type of plant, thematic unit, local names, English names, Latin names, description, habitat, cultivation, use, photos, hyperlinks and status.* *Second scenario, Catalog management:* 1. *Catalog Management can be performed by an active user in the type Administrator, so the first step is the login procedure (UC: Log-In) where the fulfillment of these conditions is checked.* 2. *Catalog Management involves changing or adding information about a plant that is already in the catalog, deleting that plant (actually hiding it because the plant is not physically deleted), or adding a new plant.* 3. *The user confirms or rejects the change.*
Extensions:	• *Search Form (QBE)* • *Tabular Form* • *Details of the Plant Form*
Frequency of Use:	*Approx. maximum is 1000 searches per minute.*
Status:	*Development status*
Owner:	*public, anonymous user*
Priority:	*high*

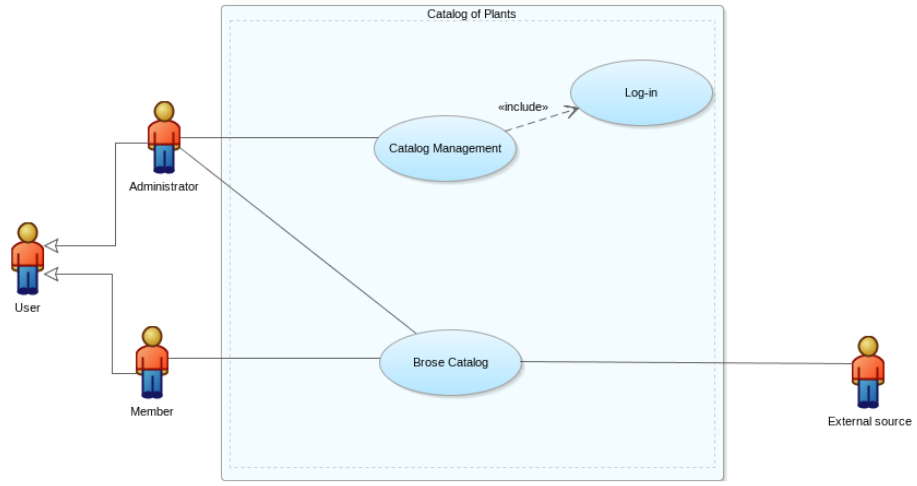


Figure 14.1: Use case diagram - Catalog of plants.

14.4 Data model

14.4.1 Narrative description of data model

The logical data model (Entities-Relationship model) contains several entities. The basic one is Plant, described with the following attributes: Plant_ID (plant identification attribute), Name local (local name of the plant), Name English (name of the plant in English), Name Latin (the name of the plant in Latin), Description (description of the plant), Habitat (description of the habitat where the plant can be found), Cultivation (method of cultivation), Use (use of the plant), Photo (one or more photos of the plant), Hyperlinks (links to external sources of information) and Status (active plant being exchanged, archived plant). Plant is a hierarchically organized code list. This means that each Type of plant can contain 0, 1 or more subtypes of plant and can be included as a subtype in 0 or 1 super-type of plant. The Thematic unit is also a code list that defines varieties of plants and seeds (vegetables, fruits, medicinal herbs, indoor and ornamental plants). Each Plant belongs to 0, 1 or more Thematic units. Each Thematic unit classifies 0, 1 or more Plants. Thematic unit attributes are: Thematic unit_ID (code, or identification attribute of Thematic unit; Name of thematic unit (one or more names). The model also shows the User entity type. Also, few entity types were added to this data model: Plant optional data, Type of optional data and Type of plant and Thematic unit. Plant optional data comprise the following attributes: Plant optional_ID (identification attribute), Description (content of additional optional data, i.e. link description, if any) and Link (optional). Type of optional data contains attributes: Type_ID (identification attribute), Name of type (general information, use, cultivation, personal opinion, review). Each Plant can have 0, 1 or more Plant optional data. Each Plant optional data refers to 1 and only 1 Plant. Each Plant optional data belongs to 1 and only 1 Type of optional data Each Type of optional classifies 0, 1 or more Plant optional data. Plant optional data is entered by 1 and only 1 User.

14.4.2 Logical data model

Logical data model is presented on Figure 14.2.

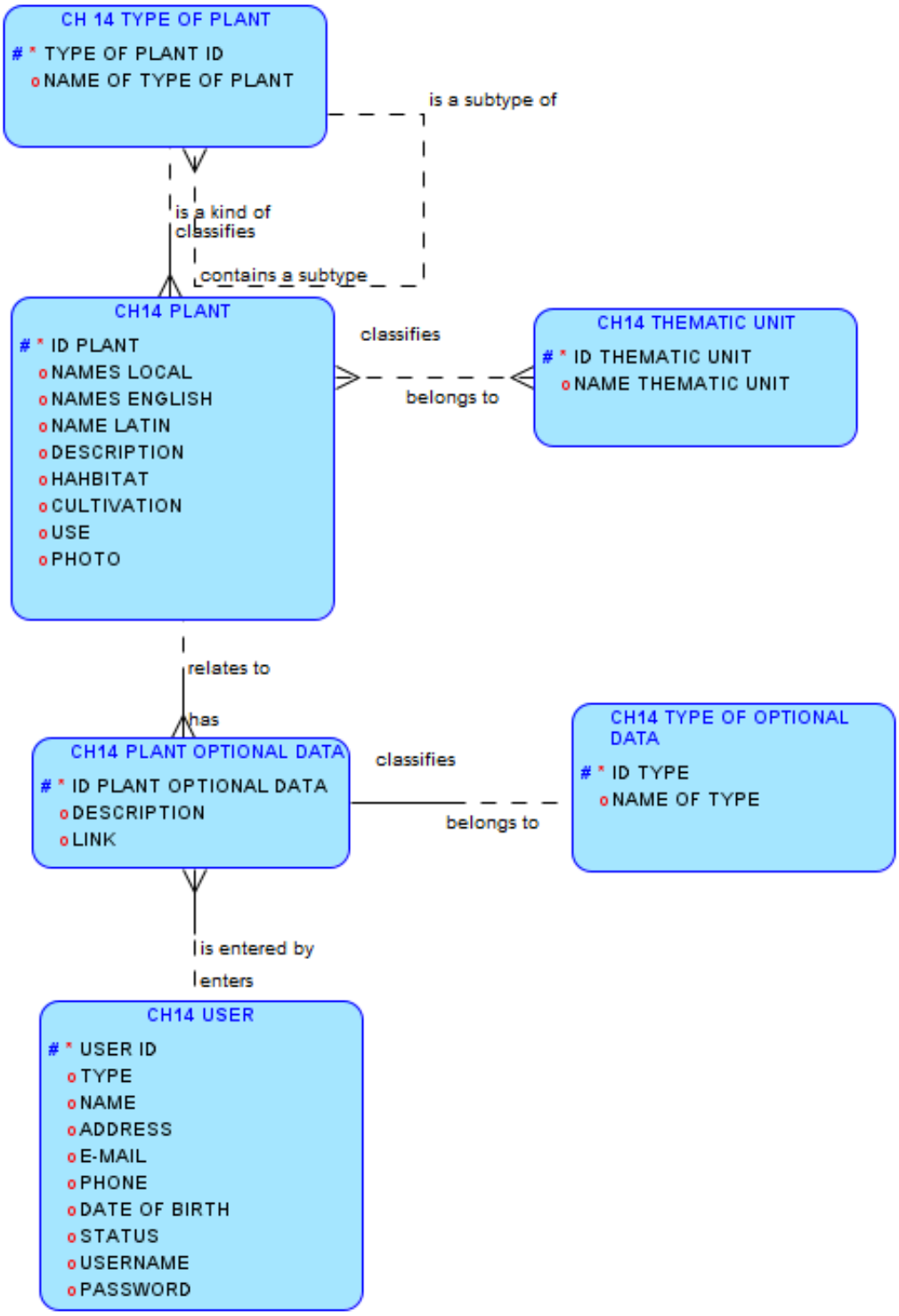


Figure 14.2: Logical data model.

14.4.3 Relational data model

Automatic transformation from logical data model to relational data model in Oracle SQL Data Modeler is provided by function *Engineering to relational*. The result is shown on Figure 14.3.

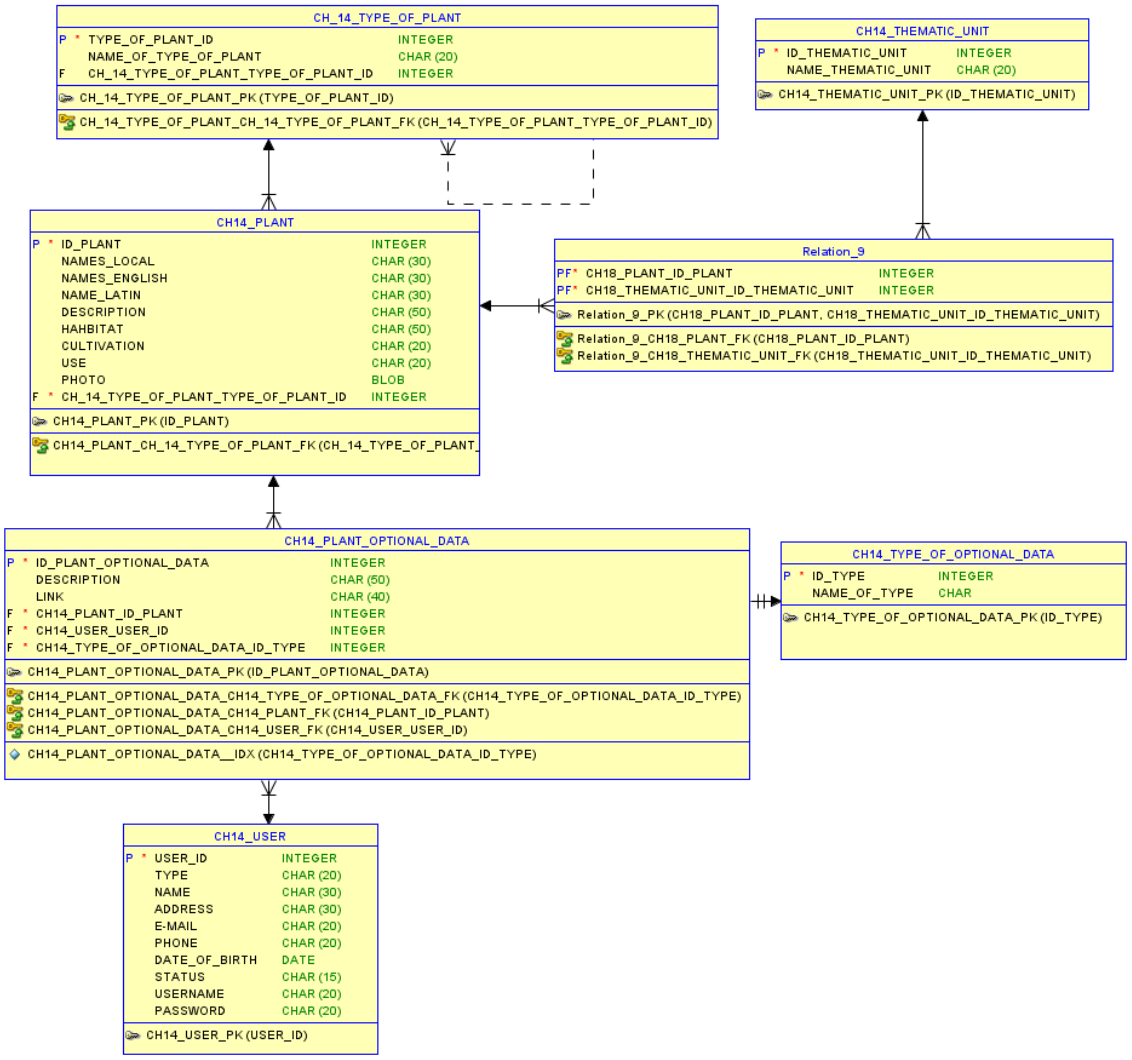


Figure 14.3: Relational data model.

Now, let Oracle SQL Data Modeler generate SQL script. Select all tables on relational model and use function *File* > *Export* > *DDL File* to get a script. We can import the generated script in APEX and execute it.

14.5 Application interfaces

Details of the interactive grid for Plant Form are presented on Figure 14.4.

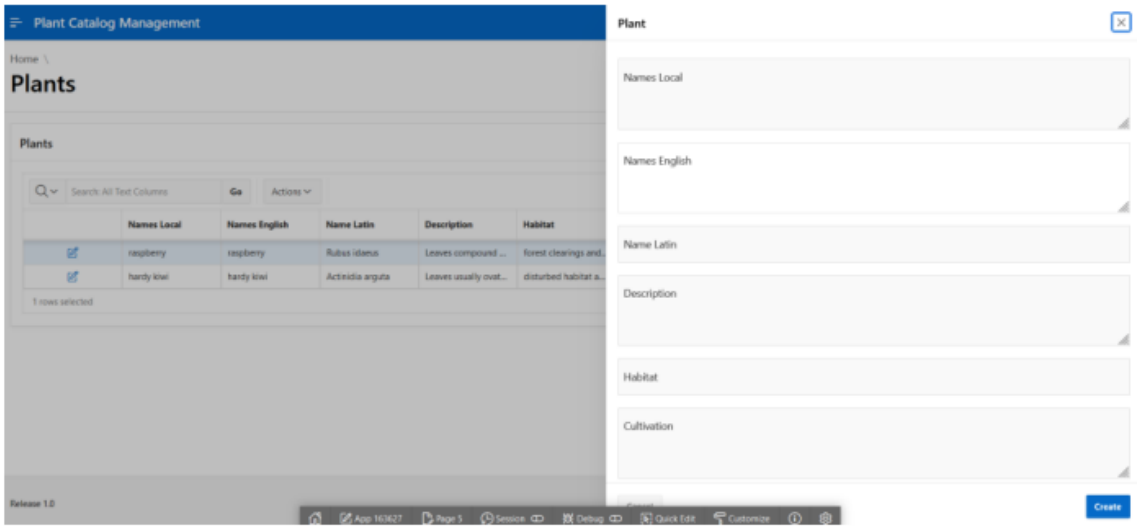


Figure 14.4: Interactive grid for Plant Form.

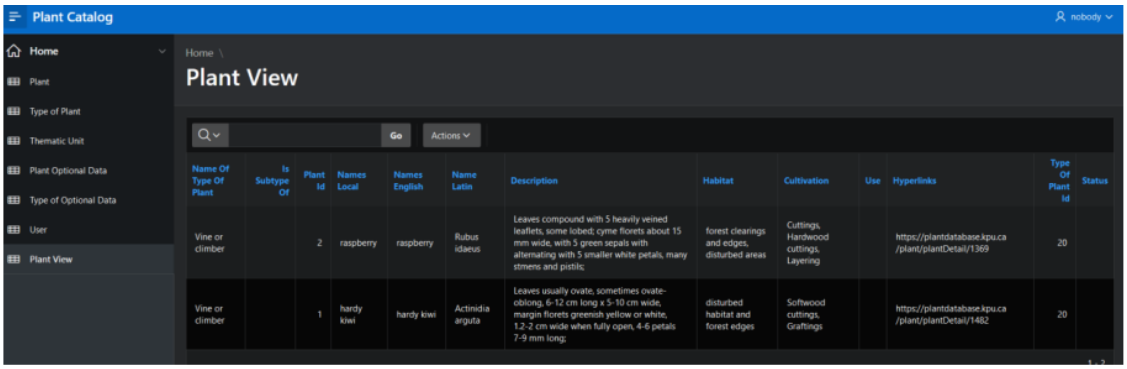


Figure 14.5: Public part – open browsing of plant data.

Browsing the catalog of local plants is implemented with the search form.

14.6 Supplementary learning material

You can find the following supplementary learning material:

- script for creating and populating tables
- script for dropping tables
- exported packaged application
- video which demonstrates how to generate application

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter14 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

14.6.1 Exported application

Exported application is packaged and uses only two tables (ch14_type_of_plant and ch14_plants). Installation creates tables as well it populates data. De-installation removes all data base objects used in this application.

Packaged application is tested and will run in a new workspace if the following requirements

are meet:

- add APEX user before running application. Only in development and testing workspace navigate to Shared Components > Application Access Control > Add User Role Assignment; enter APEX user and set this user roles Administrator, Contributor and Reader. In production consultation with skilled personnel before deployment in a must.

If user is not granted appropriate role than imported application will crash. It is necessary to clear web browser cookie (i.e. Firefox: Settings > Cookies and Site Data > Manage Data) after application crashes due to unmet requirements.

14.6.2 Video guides

Video guide shows all steps in application development

14.7 Questions

1. Investigate how communication with an External Source, or another database or system that contains plant data, could be implemented.
2. List some of the formats in which images of plants and seeds are stored.
3. In which data type are images, videos and similar objects saved in the Oracle database?

14.8 Answers

1. In principle, there are two ways of realizing communication with such an external source. One way is to download the dataset, i.e. the complete database into the Catalog of plants. Another way is to connect with the help of an Application Programming Interface (API). It is a set of protocols, tools, and standards that defines how software components should interact and communicate with each other. APIs allow different software systems to communicate and share data with each other in a standardized way.
2. Some of the most common formats are:
   a. JPEG (Joint Photographic Experts Group) - this is a popular format for digital photos due to its compression capabilities, which allow for smaller file sizes without significant loss of quality. It supports millions of colors and can handle high-resolution images.
   b. BMP (Bitmap) - this format is commonly used in Windows operating systems and can support both grayscale and color images. It does not support compression, so files can be large, but it is a good option for storing high-quality images.
   c. PNG (Portable Network Graphics) - this format is known for its lossless compression, which means that the image quality is not compromised when the file size is reduced. It supports transparency and is often used for web graphics and digital art.
   d. TIFF (Tagged Image File Format) - this format is commonly used for storing high-quality images, such as those used in printing. It supports lossless compression, multiple layers, and can handle both gray scale and color images.
3. Images, videos, and similar objects are typically stored in the Oracle database using the BLOB (Binary Large Object) data type. BLOBs can store binary data, such as images and videos, up to 4 gigabytes in size. This data type is ideal for storing large objects like multimedia files, as it allows for efficient storage and retrieval of binary data.



15. GreenDi - User Authorisation and Management

VJERAN STRAHONJA, DARKO ANDROČEC, ANA KUTNJAK, LARISA HRUSTEK

15.1 Business view of the case

A brief business overview of the GreenDi platform is described in chapter 14. It focuses on the open part of the platform GreenDi. Advanced functionalities are possible only for registered users (members and administrators). It is necessary to enable the functionality of user registration, login of registered users to the system with authentication and authorization to perform certain activities. The necessary functionalities include suspending the user under certain conditions and monitoring the history of data about the registered user.

15.2 Problem definition

The following describe the problem of User Authorization and Management:

- For a deeper insight into information about plants and other advanced functionalities, it is necessary to log in to the platform, i.e., it's possible only for registered users (members).
- Log-in is common for all functionalities and in principle must be a single sign on.
- The basic user authentication mechanism is Username and Password.
- Information about the current password is not visible to the administrator or any other user.
- The registration process includes entering basic required user information, such as name: address, email, phone, username and password.
- Optional data are user interests, date of birth, etc.
- After successful registration, a new user automatically receives the default type of Member and the status of Active. Changing the status to Administrator and vice versa, as well as type can only be done by another active administrator.
- Upon registration, the member accepts the terms of use. If he or she violates them, the administrators suspend the membership, which is also described in the terms of use.
- Regular registration and deletion of membership is voluntary and performed by the user.
- Every change of any user data is recorded in the form of a log. The administrator can review the change history of each user, except for the password.
- At the user's request, the system enables a password reset, in such a way that the administrator sends a temporary password to the user via email or SMS
- Optional functionality: In addition to the username and password, authentication is optionally possible using Fb and Google Sign-In, or other services. For example, a web application that

uses OAuth 2.0 to access Google APIs must have authorization credentials that identify the application to Google's OAuth 2.0 server.

15.3 Use cases

15.3.1 Narrative description of the use case

User Authorization and Management are realized through Use Cases (see 15.1). Any person can start Log In. Likewise, the Log In use case is included in those use cases of the GreenDi platform with which only registered users can communicate. If the user is not registered, they are directed to the User Registration use case, which is included in Log In. The basic functionality of this use case involves entering user data, including usernames and passwords. User Registration can be started from Log In also in case the user wants to change their data. User Management is a use case with which only a registered user of the Administrator type can communicate. The functionality of this use case allows the administrator to change the type I status of any other registered user, including suspending it. The user can change the password, that is, the system generates a temporary password on request that lasts a short time, and sends it to the user. External Authentication is an optional function that should enable authentication using Fb and Google Sign-In or other services. For example, a web application that uses OAuth 2.0 to access Google APIs must have authorization credentials that identify the application to Google's OAuth 2.0 server. You can find more information about using OAuth 2.0 to Access Google APIs at https://developers.google.com/identity/protocols/oauth2

15.3.2 Semi-structured description

The use case diagram describes the communication of the Actor and other systems in the environment with the system we are developing. Use cases represent the functionality of authorization and user management, which is described in Figure 15.1.

Table 15.1 presents UC.

15.3.3 Use case diagram

The above story is depicted on use case diagram 15.1.

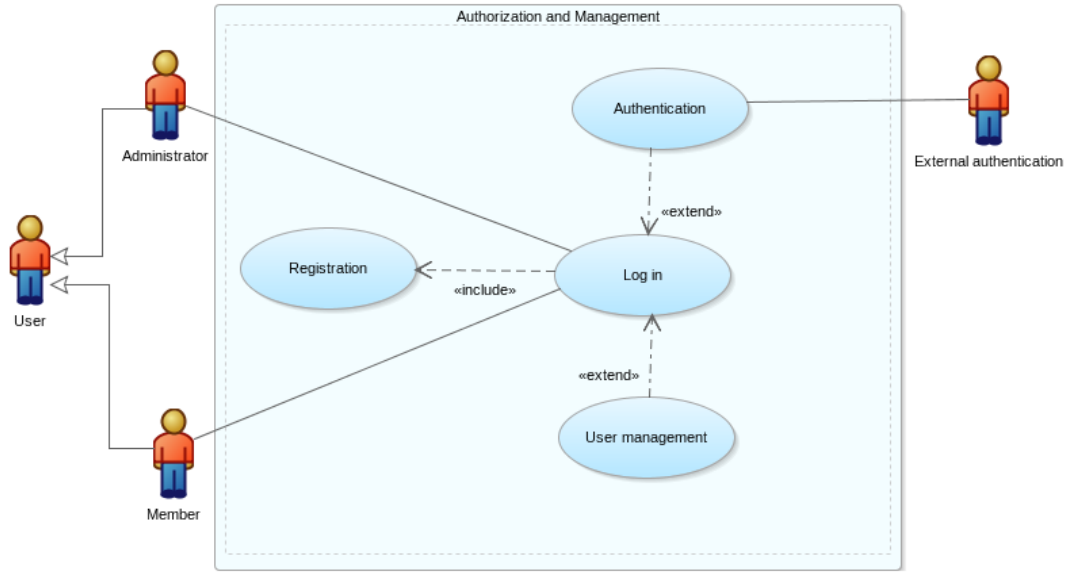


Figure 15.1: Use case diagram - GreenDi User Authorization and Management

Table 15.1: Use case description: user Authorization and Management.

Keyword	**Value**
ID:	*ch15-01*
Title:	*User Authorization and Management*
Description:	*Authorizing and managing users.*
Primary Actor:	*Any person, Member, Administrator*
Preconditions:	*Log In typically starts automatically from other use cases, where the user is set to be a member or administrator.*
Postconditions:	*The resulting situation is that the user is not authorized, or that he is authorized, but his status (pending, active ...) and type (member, administrator) are known.*
Main	-
Success Scenario:	1. Advanced functionalities of GreenDi platform requires logging in to the system (Log In Form of the Log In Use case): a) At the first login, the new user needs to register through the User Registration Form of the User Registration use case, included in the Log In use case. The registration process includes entering mandatory user information, such as name, address, email and phone, as well as optional information, such as user interests and date of birth. b)The user sets the username and the password (they are recorded in a special table in the database, separated from other user data or encrypted). c) The user accepts the terms of use and confirms the entered data. d) The default User type is Member and the default status is Active. 2. User data can be changed by the user himself or herself at each subsequent Log In to the system, through the User Registration Form. 3. Password reset is initiated by the user from the Log In Form, so that a message with a temporary password that lasts 15 minutes is sent to the user's email. 4. Every change of any user data is recorded in the form of a log in the User History data entity. The administrator can review the change history of each user, except for the password. 5. The functionality of the User Management use case allows a user of the Administrator type and in the Active status to change the type and status of another (Administrator to Member, and vice versa, as well as the Active - Suspended - Inactive status). 6. There is a table view of users with the possibility of searching (Table of Users Form). On this form, the administrator can select the details of any user, on a similar form as the User Registration Form (User Details Form), change the status and type of the user, reset the password and open the User History Form, which shows the changes in data about the selected user in tabular form. 7. Each subsequent Log In of the registered user initiates a check of User authorization, which checks the username and password, the status of the user and whether it is an administrator or a member.
Extensions:	*Log In form; User Registration Form; Table of Users Form; User Details Form; User History Form*
Frequency of Use:	*Approx. maximum is 100 per day.*
Status:	*Development status*
Owner:	*public, anonymous user*
Priority:	*high*

15.4 Data model

15.4.1 Narrative description of a data model

The logical data model (Entities-Relationship model) contains 2 types of entities: User and User History. User has the following attributes: User ID (identification attribute); Type (user type: Member, Administrator); Name (name and surname of the user; Address (address of the user); E-mail; Phone; Date of birth (opt.); Interests (opt.); Status (Active, Suspended, Inactive); User name (not visible to the administrator, encrypted); Password (not visible to the administrator, encrypted) Each User has 1 or more User History records. User History has the same attributes as User, it is a copy of the User's record after the change, with the fact that it has a system-generated change time and the ID of the person who made the change (the member himself or the administrator). This means that the changed data is recorded in this log after each change.

15.4.2 Logical data model

The logical data model is presented in Figure 15.2.

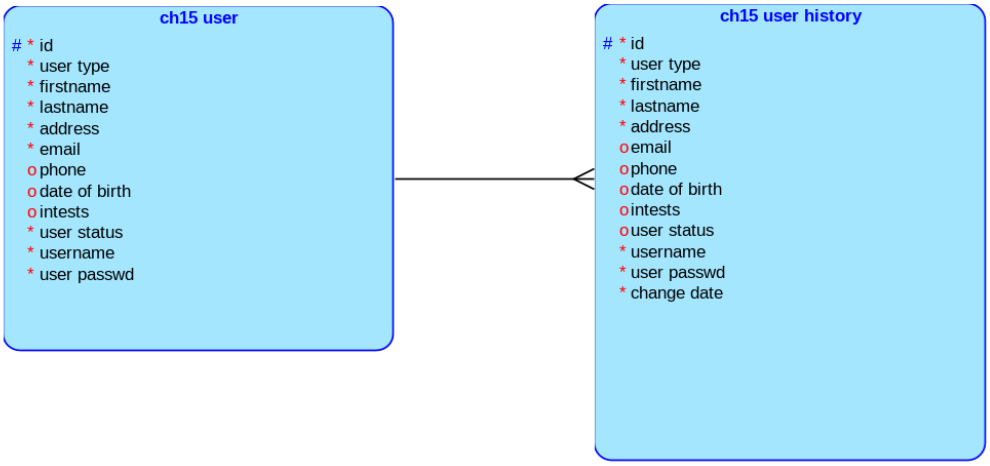


Figure 15.2: Logical data model.

15.4.3 Relational data model

The automatic transformation from the logical data model to the relational data model in Oracle SQL Data Modeler is provided by function *Engineering to relational*. The result is shown in Figure 15.3.

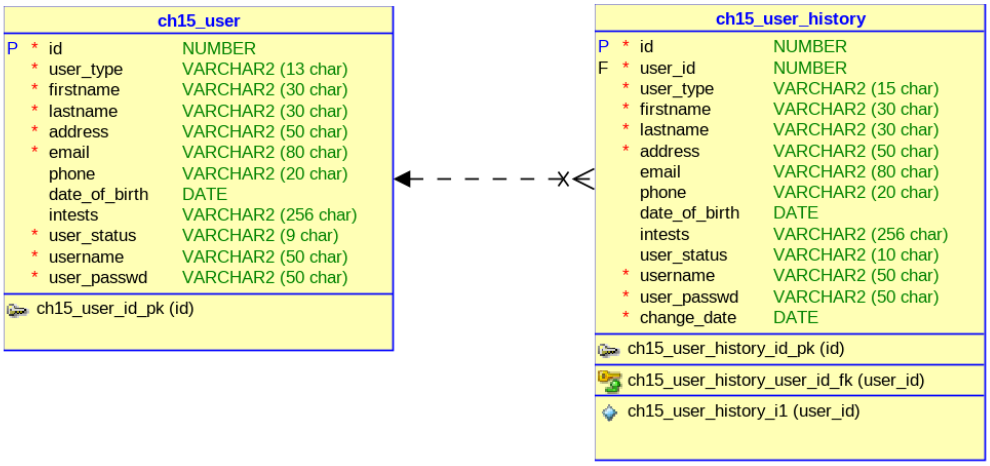


Figure 15.3: Relational data model.

Let now Oracle SQL Data Modeler generates SQL script. Select all tables on the relational model and use the function *File* > *Export* > *DDL File* to get a script. We can import the generated script in APEX and execute it.

15.5 Application interfaces

The User Form is presented in Figure 15.4.

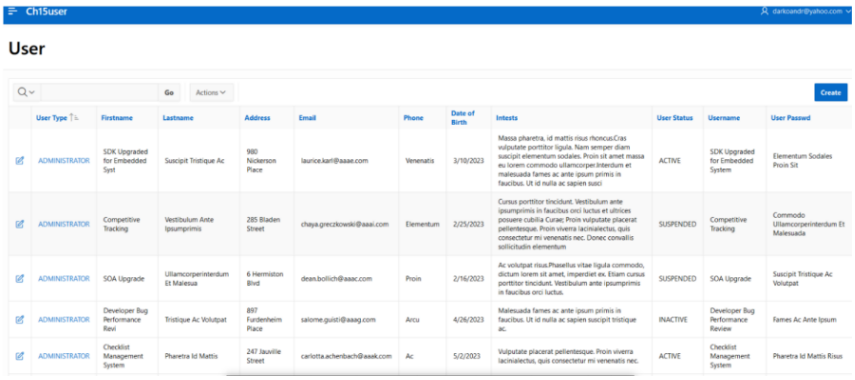


Figure 15.4: User Form.

Insert data into User History Form is presented in Figure 15.5.

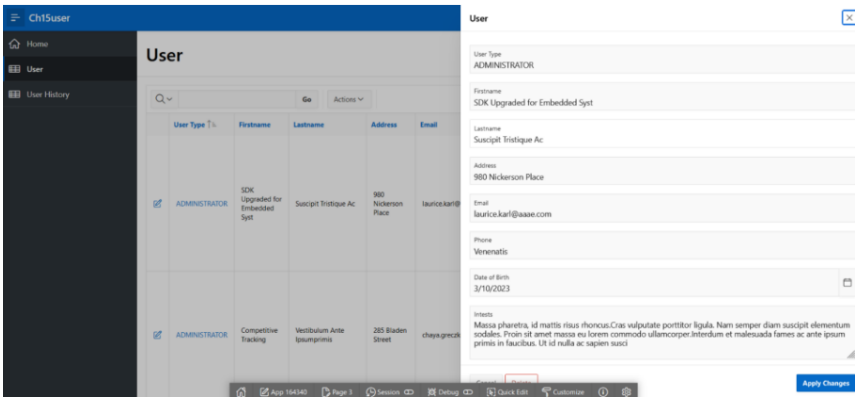


Figure 15.5: User History Form.

Functionalities that require user authorization are described in chapter 18 Webshop - GreenDi.

15.6 Supplementary learning material

You can find the following supplementary learning material:

- exported application
- scripts for creating, dropping and inserting
- video guide

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter15 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

15.6.1 Exported application

Exported application is packaged. Installation creates tables and populates data. De-installation removes all data base objects used in this application.

15.6.2 Video guides

Video guide show every step in application development.

15.7 Questions

1. What are built-in and application level user authentication and authorization functionalities when developing applications using Oracle APEX?
2. How can we use external authentication in APEX?
3. What are the advantages for users when they log into an application with their Facebook, Google or similar accounts?

15.8 Answers

1. a) Built-in User Authentication: APEX provides built-in user authentication functionality, allowing you to define and manage user accounts directly within the APEX environment. Users can have different roles and privileges, and you can control access to specific application pages or components based on user roles. b) Application-Level Authorization: APEX allows you to define fine-grained authorization rules at the application level. You can specify which users or roles have access to specific pages, regions, buttons, or other components within your application.

2. We you can use login options like Facebook, Google, and other platforms in an application or website that you develop ourself. Many popular platforms provide developer APIs (Application Programming Interfaces) that allow you to integrate their login functionality into your own applications or websites. For example, Facebook provides the Facebook Login API, which enables users to log in to your application or website using their Facebook credentials. Similarly, Google provides the Google Sign-In API, which allows users to sign in using their Google accounts. To implement these login options, you typically need to register your application or website with the respective platform's developer portal. This registration process usually involves obtaining API keys or client IDs, which you will then use in your code to authenticate users and manage the login process.
3. By incorporating login options from these platforms, you can offer your users a convenient and familiar way to sign in to your application or website without requiring them to create new accounts or remember additional login credentials. There is no need to register and enter name, address and similar data, no new user names and passwords are created that need to be remembered.



16. Small Innovation System

R. LESKOVAR, U. RAJKOVIČ, A.BAGGIA

16.1 Business view of the case

In a contemporary enterprise, innovative ideas are the key of success and improvement of business performance. Due to their knowledge about the business processes, employees are probably the best source of innovative ideas. These ideas include different viewpoints like process optimization suggestions, customer relationship management improvements, quality assurance tasks upgrades and similar. Therefore, the management in this specific case decided to collect small innovative ideas from employees and reward them.

16.2 Problem definition

Innovative ideas in the enterprise used to be collected via emails and on paper forms. An overview of collected ideas in the enterprise was time consuming. The evaluation criteria for idea assessment was not defined. Due to the high number of potential ideas that could be beneficial to the enterprise, there is a need to have a good overview. The platform should enable the collection of innovative ideas in a small enterprise. Employees should be able to sign-in to the platform and enter their ideas with simple descriptions or attachments. Since ideas do not have the same value (to the management and operations), they will be reviewed and evaluated. When evaluators finish the process the idea is rated as accepted, rejected or a revision is needed. The most innovative ideas will be awarded depending on the grade they receive. The following awards can be achieved for the proposed idea: GOLD (100 EUR), SILVER (80 EUR), BRONZE (60 EUR), CONTRIBUTOR (40 EUR), or THANK YOU. Only managers will have the option to evaluate ideas. Managers will regularly initiate the evaluation and rewarding (i.e. once per month). All signed-in users will be able to review the ideas after they are evaluated.

16.3 Use cases

16.3.1 Narrative description of use case

Each employee has an opportunity to suggest innovation ideas through the application form. All the ideas are collected and evaluated by the management. The ideas are ranked according to the usefulness and realization possibilities and good ideas are rewarded.

16.3.2 Semi-structured description

We can summarize three distinct user stories or use cases: idea submission, overview of ideas and idea evaluation. The final step, reward payment, is not covered by this business case.

Adding an attachment to the idea is actually an extension to the UC described above and we could describe it as a new use case. For the sake of readability, we will proceed with the other two main use cases, however, the application will have a feature to add attachments to the idea.

16.3.3 Use case diagram

The above story is depicted on the use case diagram (see Figure 16.1).

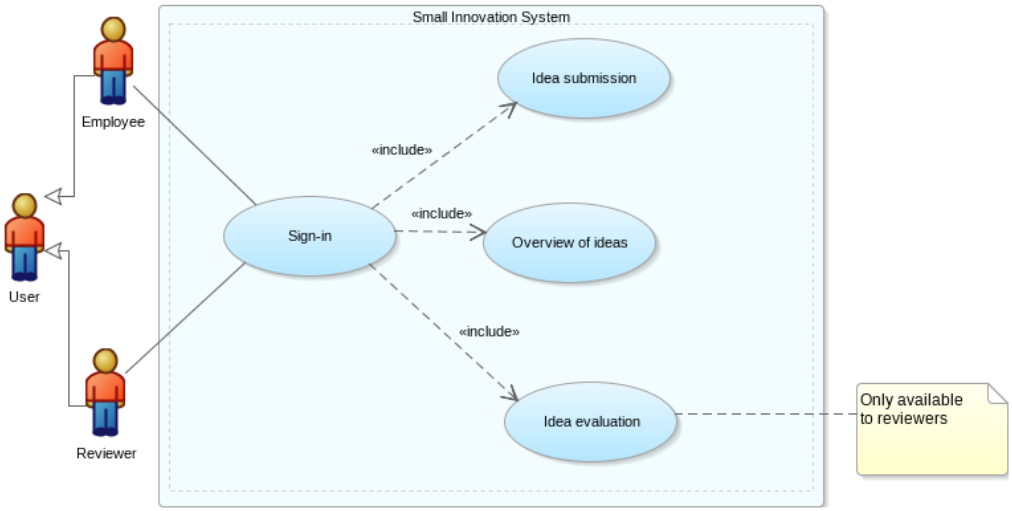


Figure 16.1: Use case diagram.

Again, for the sake of simplicity extension use case (adding an attachment to the idea) is skipped on Figure 16.1.

16.4 Data model

16.4.1 Narrative description of data model

There are three entities in the logical data model. The CH16 IDEA entity has eight attributes: ID, submission date, title, idea description, idea filename, mime type, decision and decision date. Each idea is proposed by one employee. Basic data about employees is stored in the CH16 EMPLOYEE entity: ID, first and last name. An employee can be a manager to other employees. Each employee has only one manager, while manager can manage several employees. Managers evaluate ideas. One manager can evaluate many ideas, whereas one idea can be evaluated only by one manager. Each idea can receive one award. CH16 AWARD entity has three attributes: ID, award description and amount in euros. One type of award can be given to more ideas. Evaluation rules are not included in the data model since that would pose a risk of changing application each time the rules are changed. With the proposed data model we assured: application robustness, fast development and deployment, low development and maintenance costs and flexibility.

16.4.2 Logical data model

The above story is presented on logical data model (see Figure 16.2).

Table 16.1: Use case description: idea submission.

Keyword	**Value**
ID:	*Ch16-01*
Title:	*Idea submission*
Description:	*An employee signs-in to the platform and enters the idea. Each idea has a title and a short description added as an attachment (document file).*
Primary Actor:	*Employee*
Preconditions:	*The employee needs to have the Oracle APEX user account for the Small Innovation System application. Access to the web application has to be enabled.*
Postconditions:	*After successful input of the idea, the idea and the detailed description in the attachment are available to other SIS portal users.*
Main	*Scenarios*
Success Scenario:	1. *Open the web browser and sign-in to the SIS application* 2. *Select Ideas in the menu* 3. *Select Add new idea* 4. *Enter the idea Title and add attachment* 5. *Enter file name and attachment type* 6. *Confirm the insert by clicking the Create button* 7. *Review the inserted data*
Extensions:	• *1a. Sign-in fails* • *1a* Extend:* • *1a1. Show error message* • *1a2. Open sign-in window* • *4a. Datatype error* • *4a* Extend:* • *4a1. Show error message* • *7a. Datatype error* • *7a* Extend:* • *7a1. Show error message*
Frequency of Use:	*On average, the company receives 3 innovative ideas per week.*
Status:	*Finished*
Owner:	*Employee with access to the application*
Priority:	*moderate*

Table 16.2: Use case description: overview of ideas.

Keyword	**Value**
ID:	*Ch16-02*
Title:	*Overview of ideas*
Description:	*An employee signs-in to the platform and checks the status of the idea or other ideas.*
Primary Actor:	*Employee*
Preconditions:	*The employee needs to have the Oracle APEX user account for the Small Innovation System application. Access to the web application has to be enabled.*
Postconditions:	-
Main	*Scenarios*
Success Scenario:	1. *Open the web browser and sign-in to the SIS application* 2. *Select Ideas in the menu* 3. *Browse listed ideas and their status* 4. *Select Overview in the menu* 5. *View the charts of idea submission data*
Extensions:	• *1a. Sign-in fails* • *1a* Extend:* • *1a1. Show error message*
Frequency of Use:	*The option to view ideas is given to employees, reviewers and administrators. Altogether, they access the platform approximately 5 times per day.*
Status:	*Finished*
Owner:	*Employee with sign-in option*
Priority:	*low*

Table 16.3: Use case description: idea evaluation.

Keyword	**Value**
ID:	*Ch16-03*
Title:	*Idea evaluation*
Description:	*A reviewer signs-in to the platform and reviews the idea. The award is also defined in this use-case.*
Primary Actor:	*Reviewer*
Preconditions:	*The employee user has to be added to the CH16_Reviewer role. Access to the web application has to be enabled.*
Postconditions:	*The selected idea is evaluated and the proposed award is defined.*
Main	*Scenarios*
Success Scenario:	1. *Open the web browser and sign-in to the SIS application* 2. *Select Review ideas from the menu* 3. *Identify the idea and add details using the Edit button* 4. *Select decision from the list of values (Accept, Reject, Revise)* 5. *Select the type of award from the list of values (Gold, Silver, Bronze, Contributor, Thank you)* 6. *Confirm the update*
Extensions:	• *1a. Sign-in fails* • *1a* Extend:* • *1a1. Show error message* • *1a2. Open sign-in window* • *2a. Datatype error* • *2a* Extend:* • *2a1. Show error message* • *3a. Datatype error* • *3a* Extend:* • *3a1. Show error message* • *4a. Datatype error* • *4a* Extend:* • *4a1. Show error message* • *5a. Datatype error* • *5a* Extend:* • *5a1. Show error message*
Frequency of Use:	*Managers review the proposed ideas once per month.*
Status:	*Finished*
Owner:	*Reviewer*
Priority:	*high*

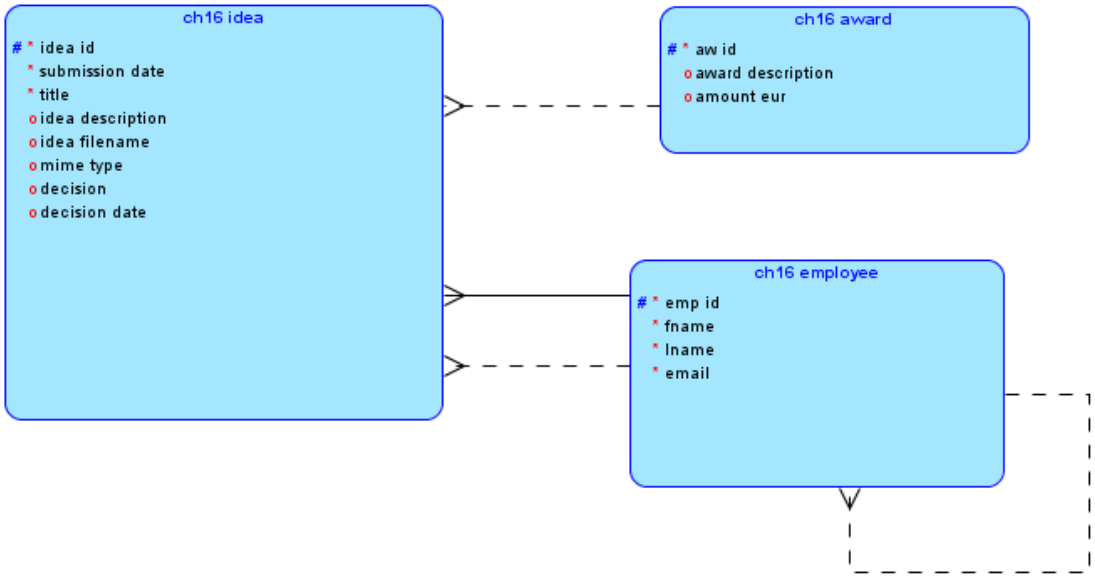


Figure 16.2: Logical data model.

16.4.3 Relational data model

Automatic transformation from logical data model to relational data model in Oracle SQL Data Modeler is provided by function *Engineering to relational*. The result is shown on Figure 16.3.

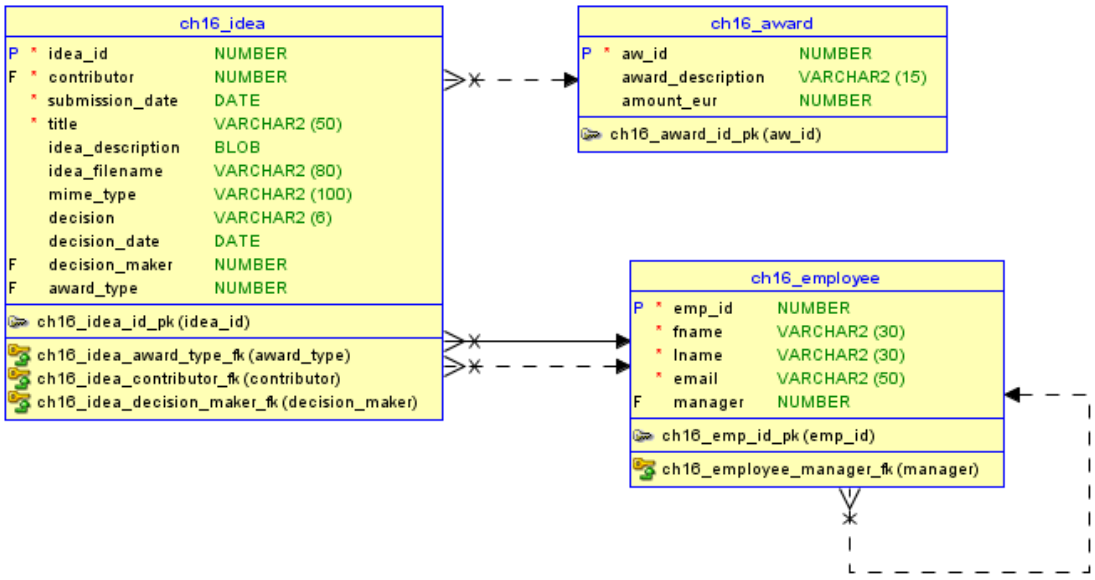


Figure 16.3: Relational data model.

Oracle SQL Data Modeler also generates SQL script for table, sequence and trigger creation. Select all tables on relational model and use function *File* > *Export* > *DDL File* to get a script like this:

```
CREATE TABLE CH16_AWARD (
    AW_ID NUMBER NOT NULL
...
```

```
);

CREATE TABLE CH16_EMPLOYEE (
    EMP_ID NUMBER NOT NULL
...
);
```

Now it is time to generate tables in Oracle database. We can import the generated script and execute it in APEX.

16.5 Application interfaces

In the following figures, application interfaces are presented for both basic roles (employee and reviewer). First, the dashboard for Employee is presented on Figure 16.4.

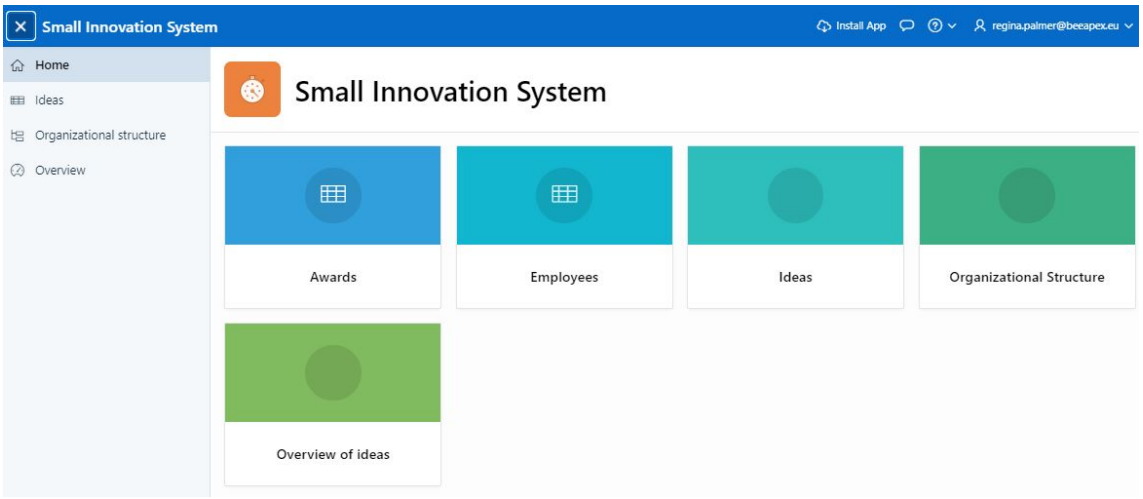


Figure 16.4: The Employee dashboard.

The page for submitting ideas with attachments and its details is presented on Figure 16.5.

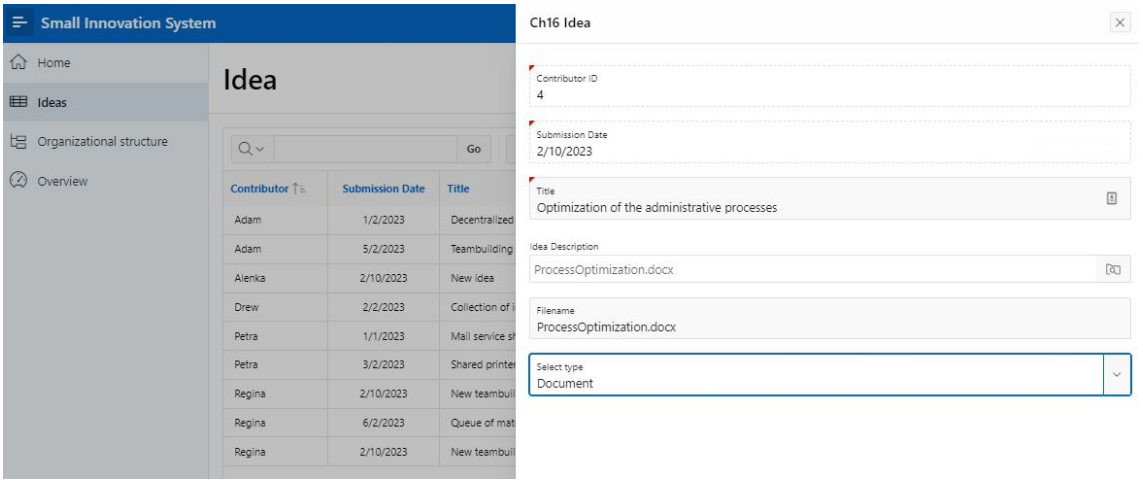


Figure 16.5: Submitting idea with attachments.

Each employee has the option to view the Organizational structure in the company, as presented on Figure 16.6.

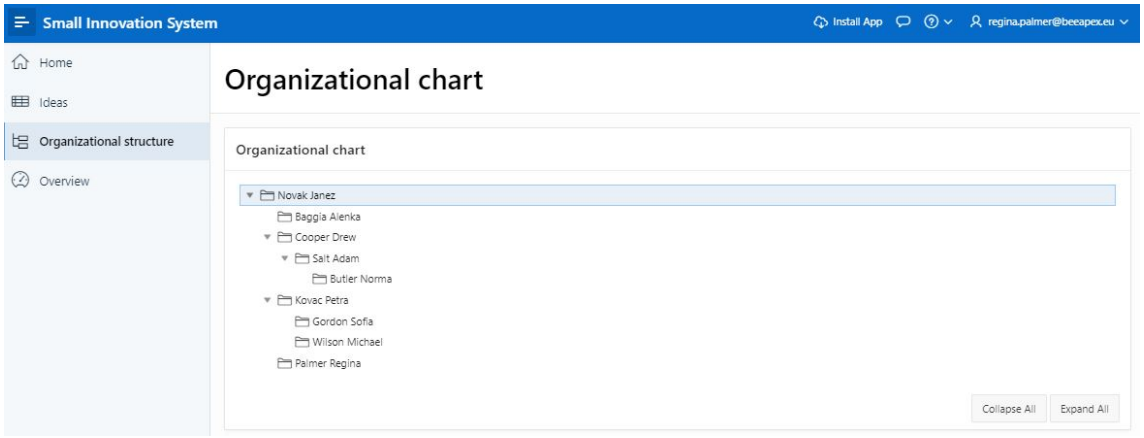


Figure 16.6: Organizational Structure in the company.

Also, each employee can see the dashboard of charts presenting the distribution of ideas over time alongside data demonstrating the breakdown of accepted, rejected, and revised ideas; and employees with the highest number of innovative ideas in the company (see Figure 16.7).

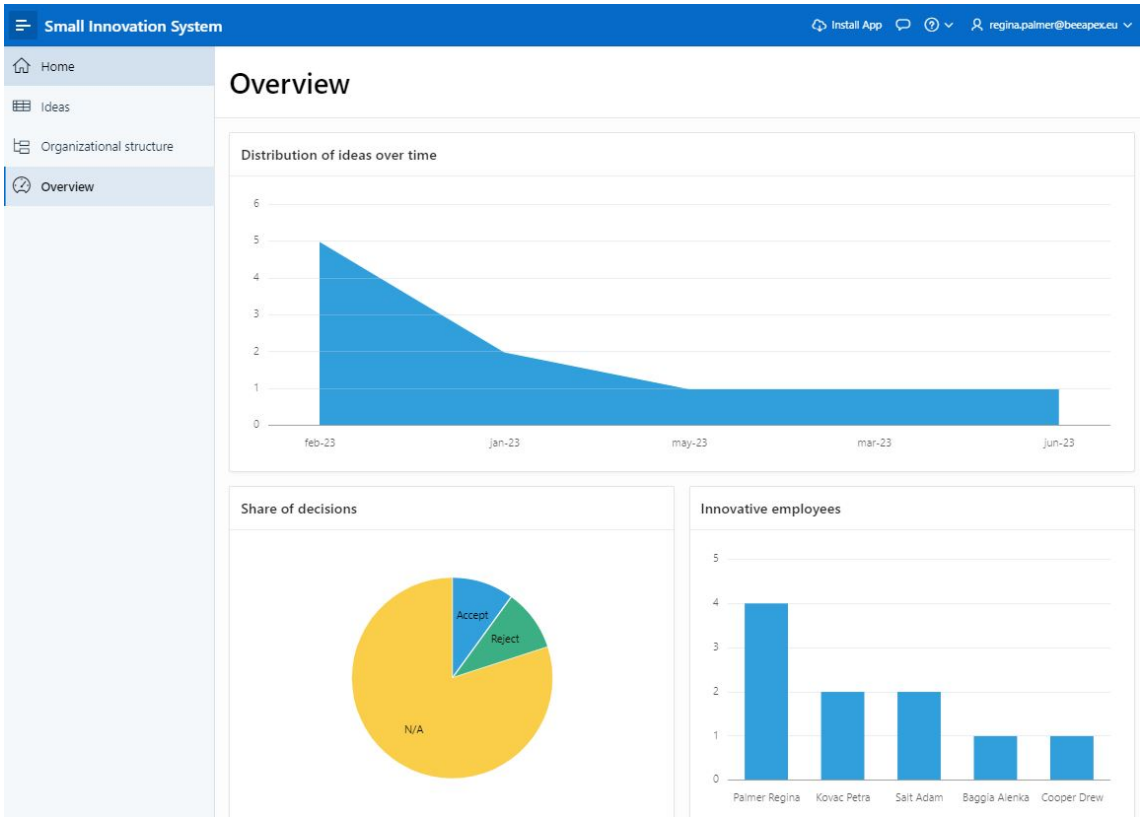


Figure 16.7: Small Innovation Idea overview.

In addition to the described actions, the reviewer has the option to update data about awards and employees, as depicted on Figure 16.8. The administration page is available only to the administrators, not to reviewers.

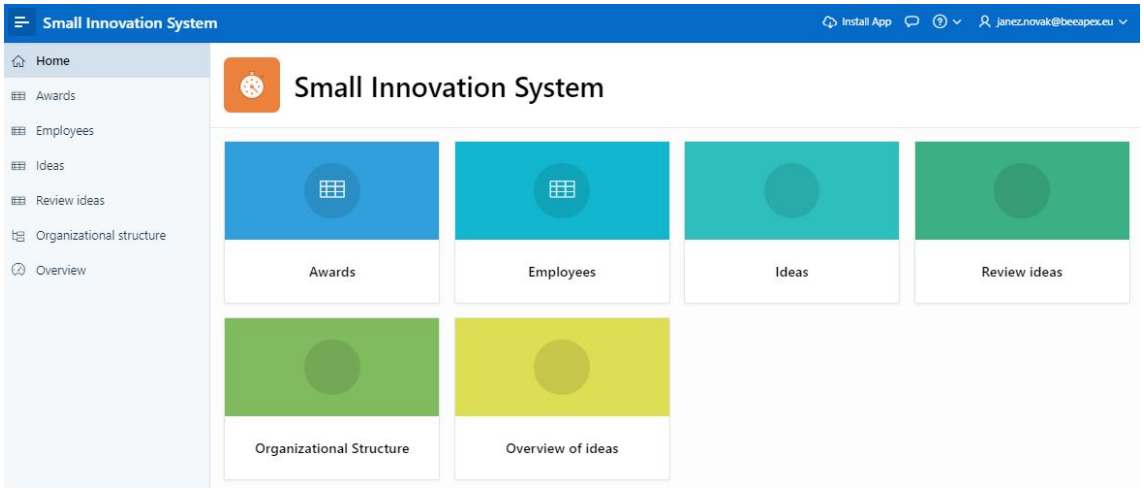


Figure 16.8: Reviewer dashboard.

The page for reviewing the idea is presented on Figure 16.9. This page is only available to the Reviewer. The Reviewer selects the decision and the type of award if applicable.

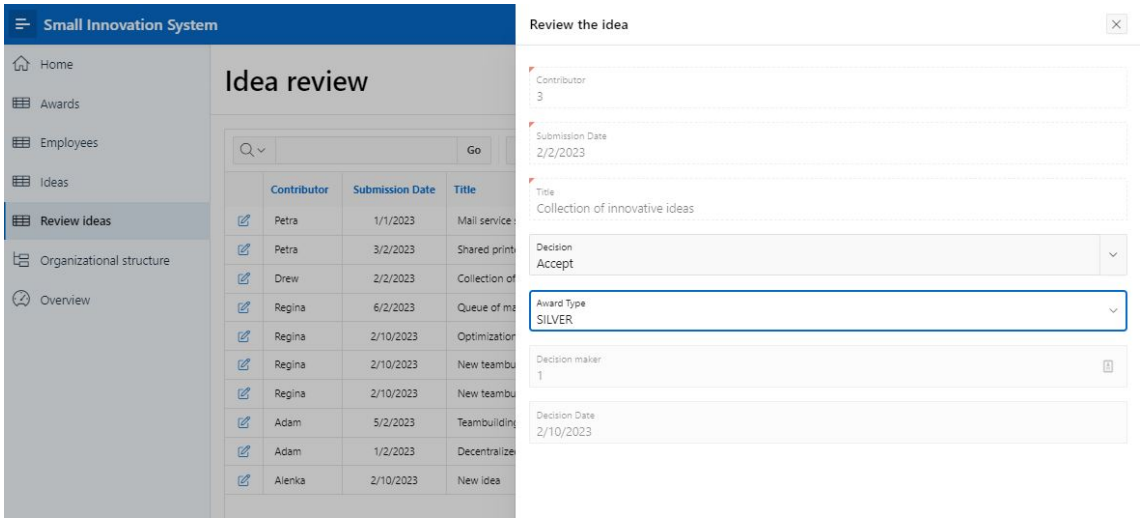


Figure 16.9: Page for reviewing the ideas.

16.6 Supplementary learning material

You can find the following supplementary learning material:

- script for creating
- script for populating tables
- script for dropping tables
- exported packaged application
- video which demonstrate how to generate application out of script

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter16 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

16.6.1 Exported application

Exported application is packaged. Installation creates tables as well it populates data. De-installation removes all data base objects used in this application.

Packaged application is tested and it will run in new workspace if the following requirements are meet:

- add APEX user before running application.

If user is not granted appropriate role than imported application will crash. It is necessary to clear web browser cookie (i.e. Firefox: Settings > Cookies and Site Data > Manage Data) after application crashes due to unmet requirements.

16.6.2 Video guides

Video guide shows all steps in application development.

16.7 Questions

1. Where are user roles in Chapter 16 defined?
2. Which relationship in the logical data model enables the presentation of Organizational structure in Chapter 16?
3. Why are some fields on the page for idea review disabled in Chapter 16?

16.8 Answers

1. No roles are defined in the Chapter 16 data model. They are only used in APEX environment.
2. The relationship connecting the Employee table with itself (recursive relationship) enables the presentation of Organizational structure in Chapter 16.
3. In Chapter 16 some fields are disabled to create a better user experience and assure consistency of the data since the user is not required to add data which are already in the database.



17. Business process management

R. LESKOVAR, U. RAJKOVIČ, A.BAGGIA

17.1 Business view of the case

Most of this book emphasize the importance of data and a data-driven approach to developing web applications with APEX. But the data-driven approach has sibling: process-driven applications. As the name suggests, process-driven applications focuses on processes. Neither can good data-driven application neglect the process aspect nor can process-driven application neglect the data aspect. Implicit or explicit inclusion of both, make any application fit for purpose. It is important for the developer to have profound insight into business processes in organization. Any type of organization, regardless of size, annual earnings, ownership, industry and the mission, conducts business processes. The common classes of business processes that apply for most organizations fall into the following categories:

- sales and marketing include sub processes such as acquiring and retaining customers, generation of leads, customer relationship management, and marketing campaigns.
- supply chain management includes sub processes such as coordination and management of the flow of goods and services, procurement, production, and distribution.
- financial management includes sub processes such as management of an organization's financial resources, accounting, budgeting, and financial reporting.
- human resource management includes sub processes such as management of the organization's workforce, recruitment, training, and employee development.
- customer service includes sub processes such as providing support and assistance to customers, complaint resolution and problem solving.
- operations management includes sub processes such as management of day-to-day business operations, production, logistics, and inventory management.
- information technology includes sub processes such as management of an organization's information technology resources and systems, hardware, software, and data management.

These processes are considered critical as they support the core functions of an organization and enable achieving its goals and objectives. Improving the efficiency and effectiveness of these processes can have a significant impact on the overall performance and success of an organization. In an attempt to make clear understanding and definition of typical business OASIS OPEN developed Universal Business Language which describe processes and business documents (see [5]). Software industry started in late 1990s with integration of processes, invented Business Process Execution Language (BPEL) and later Business Process Modeling Language (BPML). Big players like

Microsoft, IBM, SAP, Oracle as well as myriad of smaller contributed to software platforms which aimed to provide seamless integration of business functions. Apart from software world other industries have sharpened focus on processes through the lens of specific businesses from 1980s with quality management initiatives and standards.

Since the 1990s, the disciplines of business process management (BPM) and business process re-engineering (BPR) have important roles when organizations want to improve their performance. BPM focuses on the representation and documentation of a company's workflows, operations and processes using diagrams, flowcharts, or other visual aids. The goal is to understand and improve existing processes, or design new ones, to increase efficiency and reduce inefficiencies. BPR is sought as the fundamental rethinking and radical redesign of business processes to achieve significant improvements in performance and productivity. It involves the analysis and design of workflows and processes within an organization, with the goal to be more efficient, effective, and capable of adapting to the changing business environment. BPR can involve utilization technology and information systems to automate processes and enhance decision-making.

In BPM and BPR several diagrams can be used:

- Flowcharts: A simple and intuitive way to represent a process as a series of steps, decisions, and loops.
- Swimlane diagrams: A type of flowchart that adds a visual representation of who or what is responsible for each step in a process.
- Process maps: A high-level view of the steps and activities, involved in a process, often including inputs, outputs, and decision points.
- BPMN (Business Process Model and Notation [4]): An industry standard notation used for modeling business processes, standardized by the Object Management Group (OMG).
- EPC (Event-driven Process Chain): A type of flowchart that uses events as the driving force behind a process, instead of activities.
- IDEF (Integration Definition for Function Modeling): A method used in software engineering to model and analyze processes and systems.

The choice of diagram type to present workflow depends on the level of details needed, the purpose of the model, and the target audience. At this point we will focus on BPMN only as: a) BPMN has standard notation and wide acceptance across different industries and b) BPMN is implemented as modeling diagram type in Oracle Application Express (APEX) feature (sometimes referred as extension) called Flows for APEX. Flows for APEX is open source, licensed under very permissive MIT Licence. Flows for APEX allows developers to build and deploy web-based applications using a visual, drag-and-drop interface. Flows for APEX provide a way to automate and simplify complex business processes by breaking them down into a series of steps and tasks, and guiding users through them.

Flows for APEX can include a variety of elements, such as pages, forms, reports, and dialogs, and can be configured to include branching logic, conditional branching, data validation, and error handling. The visual interface of Flows for APEX makes it easy for developers to design, build, and test business processes, without the need for extensive coding. It can be integrated with other Oracle APEX components, such as SQL, PL/SQL, and REST services, to create complete, web-based applications as well as to build custom solutions for specific business needs, such as workflow management, customer on-boarding, and data management. Overall, Flows for APEXs provide a way for organizations to streamline and automate complex business processes, improve productivity, and reduce errors and inefficiencies. Excellent tutorials on Flows for APEX are available on this site. So, how can we link APEX as low-code programming environment and Flows for APEX with embedded standard BPMN presentation?

Activity diagram (see Figure 17.1) depicts the simplified presentation of integrating Flows and APEX application.

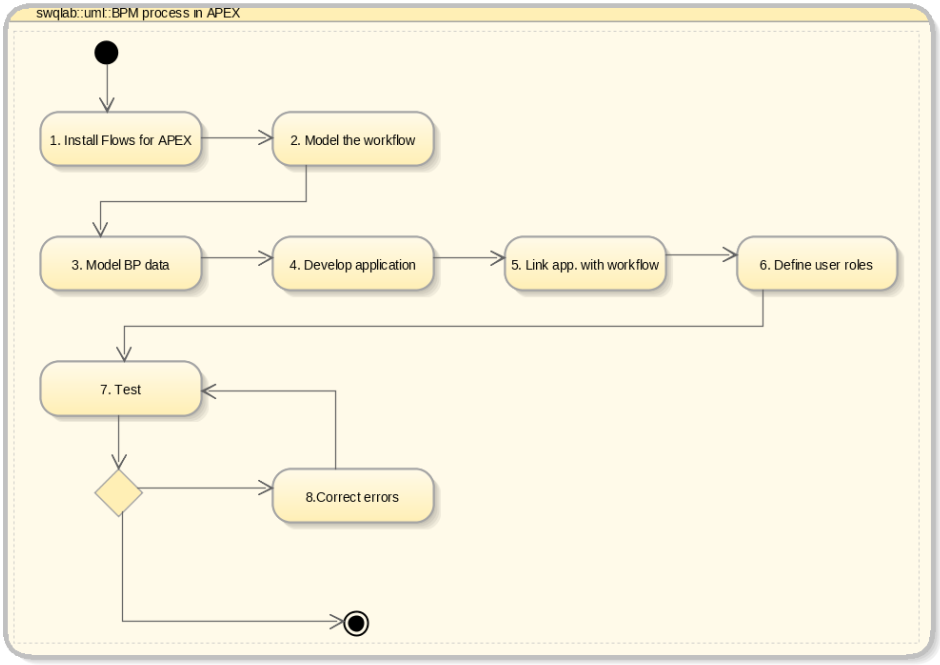


Figure 17.1: Integrating Flows for APEX with APEX application.

Steps on Figure 17.1 *3. modeling data, 4. developing application, 6. defining user roles, 7. testing* and *8. removing errors from application* are common for all APEX applications. Step *1. installing Flows for APEX* is similar to installing any other packaged application. The real novelties are *2. modeling the workflow* and *5. linking APEX application with workflow*.

Readers not acquainted with business process management (BPM) and business process modeling notation (BPMN) diagrams are advised to search for popular textbooks and study already mentioned BPMN spefification [4]. As a brief reminder for all readers, the following elements of BPMN diagrams are the most commonly used:

- Start Event: Indicates the starting point of a process.
- Task: Represents a single unit of work, such as an activity or a step in the process.
- Gateway: Represents a decision point in the process, such as a fork in the flow or a conditional branch.
- Sequence Flow: Connects elements in the diagram and represents the flow of control from one element to another.
- End Event: Indicates the end of a process.
- Pool/Lane: Represents a grouping of related tasks, used to define the roles and responsibilities of different participants in the process.
- Message Flow: Represents the flow of messages between participants in a process, such as a communication between two systems.
- Data Object: Represents a piece of data that is used or produced in a process, such as an invoice or a customer record.
- Data Store: Represents a container for storing data that is used or produced in a process, such as a database or a file system.

These elements can be combined to create detailed and accurate models of business processes, allowing organizations to identify areas for improvement and optimize their workflows.

17.2 Getting started with Flows for APEX

17.2.1 Install Flows for APEX

Download zip file from this URL and unzip all files to your computer. Follow instructions provided by this link. Before installing Flows for APEX plug-in, check requirements. The requirements are met, if you are using your APEX free workspace, free/payable OCI or Oracle Academy account. Two additional post-installation tasks are:

- grant "create job" to workspace. You or APEX instance administrator must establish connection to the database (with OCI interface, SQL Developer, SQL Plus, TOAD) and issue command. You can not grant this privilege within APEX.
- configure workspace, default application and user info within application "Flows for APEX". You can do this step any time later when you develop your own process-driven application.

Though sometimes called feature or plug-in, the installation will create new data base objects (tables, views) and application, called "Flows for APEX". This application is an administrative interface which allows you:

- to monitor instances of workflows on dashboard (Dashboard),
- to visually create new workflows (Flow Management) and
- to execute workflows by creating instances, completing tasks that belong to instances and testing the workflows (Flow Monitor).

17.2.2 Install Sample Process Flow Application

Sample process Flow application is also provided in the same zip file as Flows for APEX. Installation is straight-forward. It deals with expense claims. Employee prepares expense claim which is validated by manager. If the claim is declined, than the employee is informed. Finally, for approved claims, the accounting department prepare payment, execute bank transaction and set status of payment. If the claim is paid, then this instance of the workflow is finished. The application is very similar to real process in any organization - and because it is so well documented it presents great study material.

17.2.3 Read and practice exercises

There are excellent tutorials for Flows for APEX. Start with this site. At the end of these introductory exercises find BPMN tutorial and APEX integration tutorial. Reading and practicing will empower you to develop your own process driven application. Practicing through all exercises will probably take a few hours for the reader to complete. The reader is strongly encouraged to experience learning-by-doing. The rest of this chapter will provide the same template as other chapters in this textbook (from problem definition to application) with additional inclusion of BPMN and application integration with Flows for APEX. However the scope of the application, developed in the Chapter 17 is narrowed to core functionality and skip the details of authorization. It will provide guides to:

- define workflow in Flows for APEX
- start the instance of the defined process within developed application
- monitor the flow of instance within developed application
- complete each task of specific instance of the workflow within developed application

Narrowed application can also present a nice challenge for enthusiastic learner, does it?

17.3 Problem definition

A medium sized manufacturing company receives inquiries for custom made products by customers. Each inquiry may have more related documents with specifications, required standards, schemes, sketches and similar. A customer can have more inquiries. The sales person prepares and classifies

inquiry documents into three categories: manufacturing, financial and business. To evaluate one inquiry sales must prepare multiple documents for decision makers. The production manager and his team evaluate the inquiry manufacturing aspect. They provide an expert opinion on a company's capability to produce the required product item. If the product item can be manufactured, then the team provide estimated duration of manufacturing process in days. Financial manager and his team evaluate the inquiry financial aspect. They provide an expert opinion on the company's capability to provide financing and estimate the expected profit. The final evaluation and the decision is taken by chief executive officer and his team. They consider the feasibility (manufacturing, financial), estimate the importance of the customer and make decision abaout the inquiry. Depending on the company's capability to manufacture and to finance the business opportunity as well as business prospect, the CEO decide to how to reply to customer: a) send business proposal with manufacturer's price and expected delivery duration or b) message to inform the customer that the manufacturer can not send business proposal.

Bad business decisions (send business proposal or reject the business) can have disastrous impacts on manufacturing company such as lost earnings, low profit, bad reputation due to exceeded delivery dates and even bankruptcy. The situation presented cause headache to all involved: sales person, production manager, financial manager and chief executive officer. IT department is responsible to provide user-friendly, reliable and secure software support for described workflow. Their problem is to develop such support with Flows for APEX.

To developed the application, you must have installed Flows for APEX. This chapter will analyze the business case (description, semi-structured description, use case diagram, process model, data model) and develop application. You will learn:

- how to define a process model (workflow) for this business case,
- how to create data model and develop APEX application
- how to link developed application with defined workflow

Please keep in mind that user authorization will not be implemented because this would expand this chapter beyond reasonable student effort.

17.4 Use cases

17.4.1 Narrative description

Four actors are involved in four use cases. Each actor is responsible for one use case: sales person prepares documentation related to inquiry, production manager provides evaluation of manufacturing aspects, financial manager provides evaluation on financial aspects and chief executive officer takes decision. Use case diagram (see Figure 17.2) assumes that workflow designer will decide whether production and financial manager should work in parallel or sequential way. If they work in parallel and one or both decide that inquiry requirements can not be met, than some unnecessary effort (costs) appears, but the workflow can be completed faster. If workflow designer decides for sequential approach than workflow will take more time but less unnecessary effort will appear.

17.4.2 Semi-structured description

UML proposes that each use case has also semi-structured description. Four distinct semi-structured descriptions are provided in Tables 17.1, 17.2, 17.3 and 17.4.

17.4.3 Use case diagram

Use case diagram can also depict generalization. In the given context an employee has four special instances (field managers) and each manager is associated with one use case. All four use cases are depicted in Figure 17.2.

Table 17.1: Use case description: prepare inquiry documentation.

Keyword	**Value**
ID:	*ch17-01*
Title:	*Prepare inquiry documentation*
Description:	*Sales person create new inquiry and add related documents*
Primary Actor:	*Sales person*
Preconditions:	Customer already sent all documents related to inquiry
Postconditions:	-
Main	-
Success Scenario:	1. add new inquiry 2. add related documents to specific inquiry 3. confirm with "Confirm" button or dismiss with "Cancel" button
Extensions:	-
Frequency of Use:	*Approx. 250 per year, average 5 per week.*
Status:	[Development status]
Owner:	Sales person
Priority:	*high*

Table 17.2: Use case description: evaluate manufacturing aspects of inquiry.

Keyword	**Value**
ID:	*ch17-02*
Title:	*Evaluate manufacturing aspects of inquiry*
Description:	*Production manager and his/her team provide expert opinion whether company has capability to manufacture product item and if yes how long (in days) would manufacturing of given quantity take. Justification of opinion is obligatory.*
Primary Actor:	*Production manager*
Preconditions:	Inquiry documentation prepared
Postconditions:	-
Main	-
Success Scenario:	1. enter opinion 2. confirm with "Confirm" button or dismiss with "Cancel" button
Extensions:	-
Frequency of Use:	*Approx. 250 per year, average 5 per week.*
Status:	[Development status]
Owner:	Production manager
Priority:	*high*

Table 17.3: Use case description: evaluate financial aspects of inquiry.

Keyword	**Value**
ID:	*ch17-03*
Title:	*Evaluate finacial aspects of inquiry*
Description:	*Financial manager and his/her team provide expert opinion whether company has capability to finance product item and if yes what would be expected profit. Justification of opinion is obligatory.*
Primary Actor:	*Financial manager*
Preconditions:	Inquiry documentation prepared
Postconditions:	-
Main	-
Success Scenario:	1. enter opinion 2. confirm with "Confirm" button or dismiss with "Cancel" button
Extensions:	-
Frequency of Use:	*Approx. 250 per year, average 5 per week.*
Status:	[Development status]
Owner:	Financial manager
Priority:	*high*

Table 17.4: Use case description: evaluate business aspects of inquiry.

Keyword	**Value**
ID:	*ch17-04*
Title:	*Evaluate busimess aspects of inquiry*
Description:	*Chief executive officer and his/her team make decision on whether to provide business proposal to customer of to skip the offer. Justification of opinion is obligatory and demanded by board of directors.*
Primary Actor:	*Chief executive officer*
Preconditions:	Use cases 17.1, 17.2 and 17.3 finished.
Postconditions:	-
Main	-
Success Scenario:	1. enter opinion 2. confirm with "Confirm" button or dismiss with "Cancel" button
Extensions:	-
Frequency of Use:	*Approx. 250 per year, average 5 per week.*
Status:	[Development status]
Owner:	Chief executive oficer
Priority:	*high*

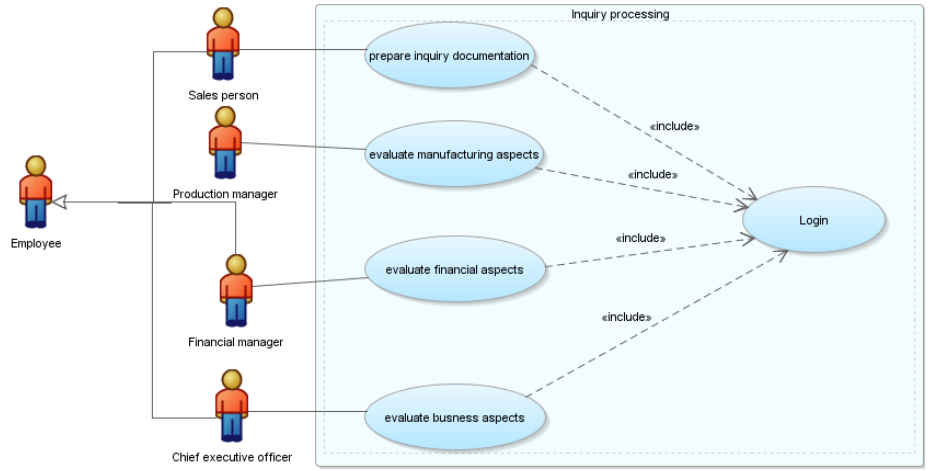


Figure 17.2: Use case diagram.

17.5 Workflow model

Initial workflow model introduces a preparation task which then splits into parallel gateway with two tasks (manufacturing and financial evaluation). Workflow proceeds to the last task (business evaluation) when both of previous tasks are finished (see Figure 17.3).

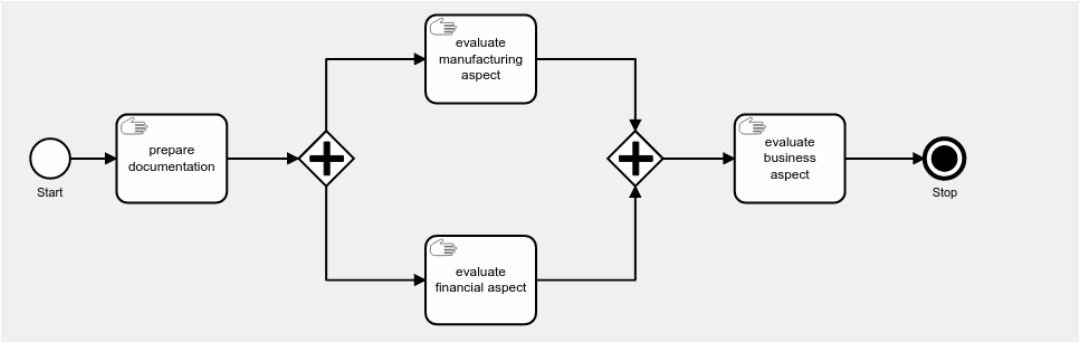


Figure 17.3: Workflow - processing inquiry - BPMN diagram.

All four tasks are defined as "Manual task" and end event (Stop) is defined as "Terminate end event". We can test the workflow immediately in Flow Monitor (in "Flows for APEX" applcation). Save flow as CH17. Then, navigate to Flow Monitor. Click "Create instance" button (see Figure 17.4).

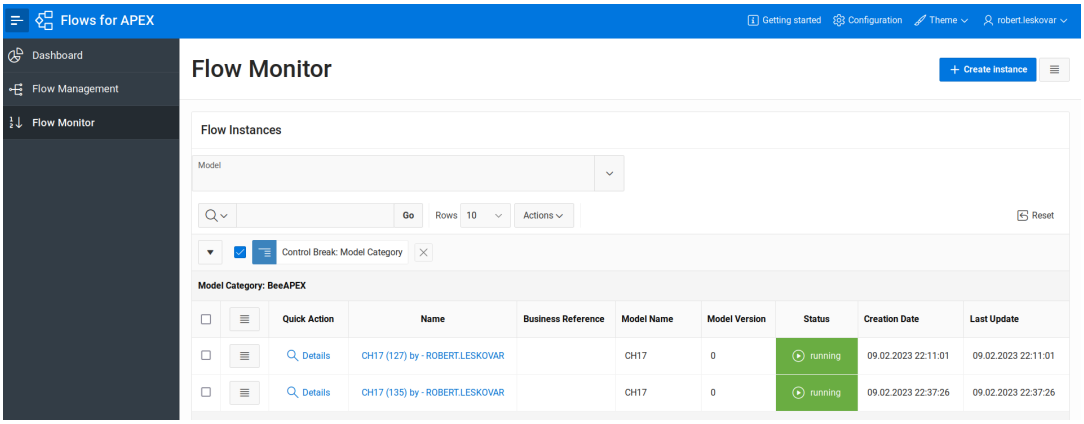


Figure 17.4: Creating instances.

Select model (CH17) and enter distinct name of the instance of the flow CH17 (see Figure 17.5).

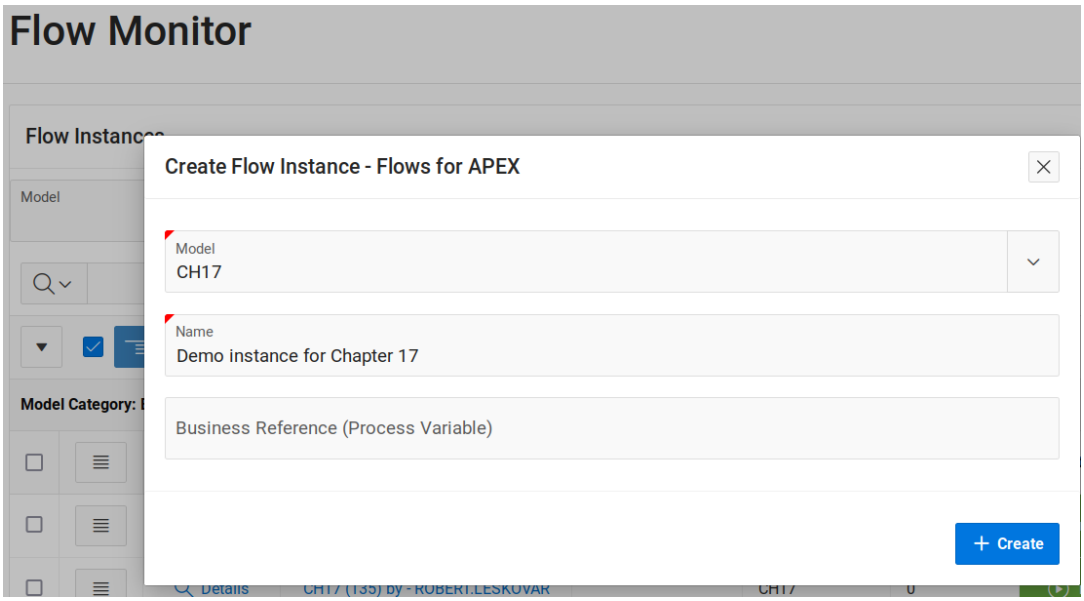


Figure 17.5: Creating instance of the flow CH17.

New instance appears. Click the indicated icon on Figure 17.6 and select "Start".



Figure 17.6: Start demo instance.

The status of the instance will change to "running". Click "Details" for demo instance. Set side-by-side display settings to and click "Complete" in column "Quick Action" (see Figure 17.7).

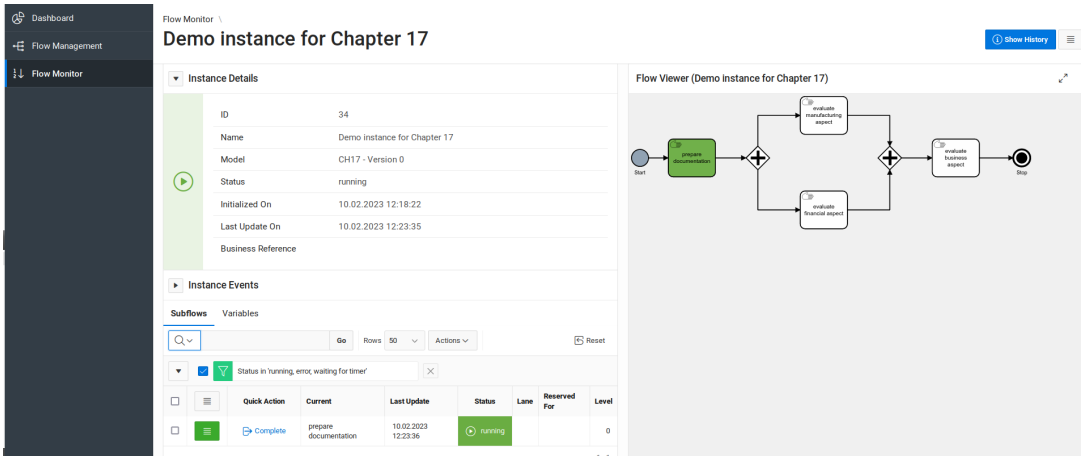


Figure 17.7: Complete first task in demo instance.

Now, the flow proceeds with parallel gate and two tasks (see Figure 17.8).

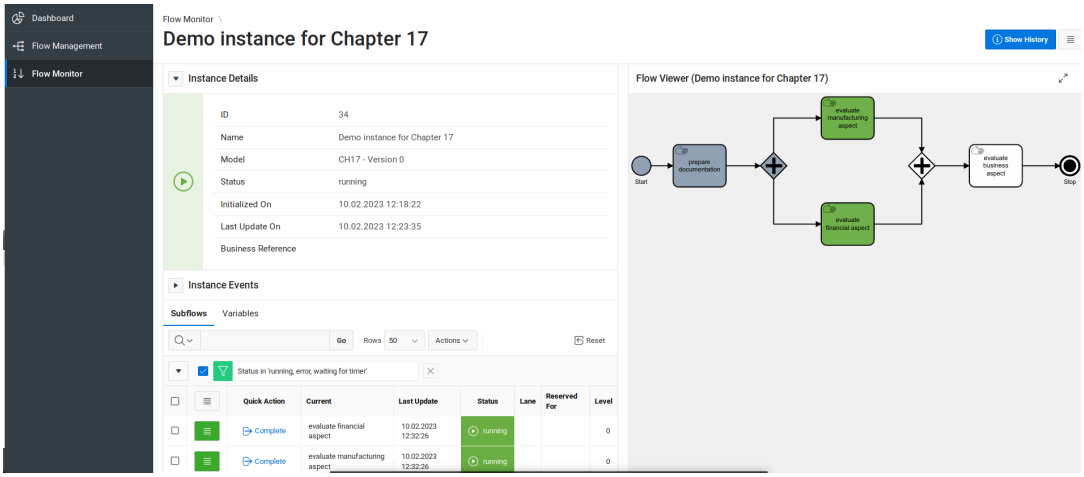


Figure 17.8: Executing tasks in parallel gate.

Click "Complete" in column "Quick Action" for both tasks and proceede with the final task. Repeat completion for the last task and the instance is completed (see Figure 17.9).

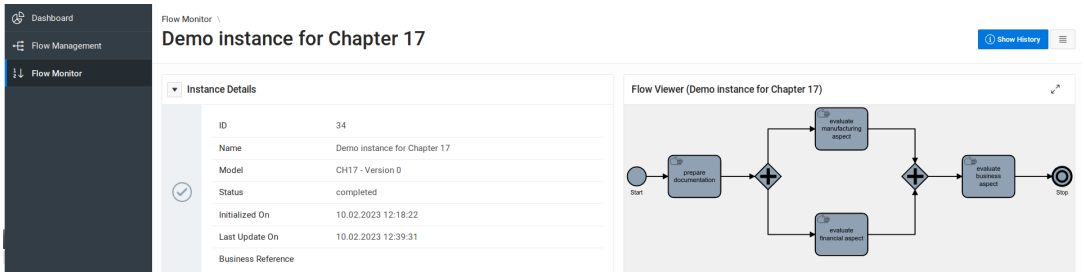


Figure 17.9: Completed demo instance.

Entire history of the instance will be presented by clicking "Show history". We have proved, that the workflow CH17 is executable and we can model data and application interface.

17.6 Data model

17.6.1 Narrative description of data model

The most important entity is **ch17 inquiry**. An instance of **ch17 inquiry** is related to one instance of **ch17 customer** while customer may have multiple inquiries. Each instance of **ch17 inquiry** must have at least three instances **ch17 document**. Each instance of **ch17 document** belongs to one instance of **ch17 document class**.

Attributes for entity **ch17 inquiry** are: ID, customer requirements (price per order, quantity, delivery date), manufacturing aspects (capability, delivery time for requested quantity, justification), financial aspects (financial capability, expected profit for required quantity) and business aspects (decision to take or leave opportunity, justification).

We will define a limited set of attributes for other entities to provide this case more compact.

17.6.2 Logical data model

Logical data model is presented on Figure 17.10.

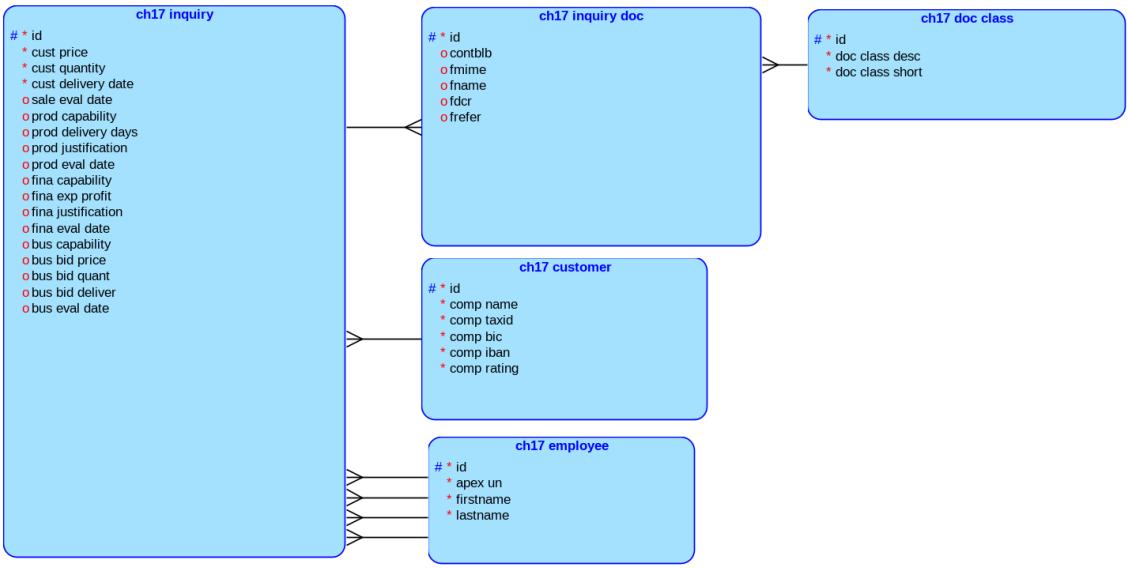


Figure 17.10: Logical data model.

17.6.3 Relational data model

Relational data model is presented on Figure 17.11.

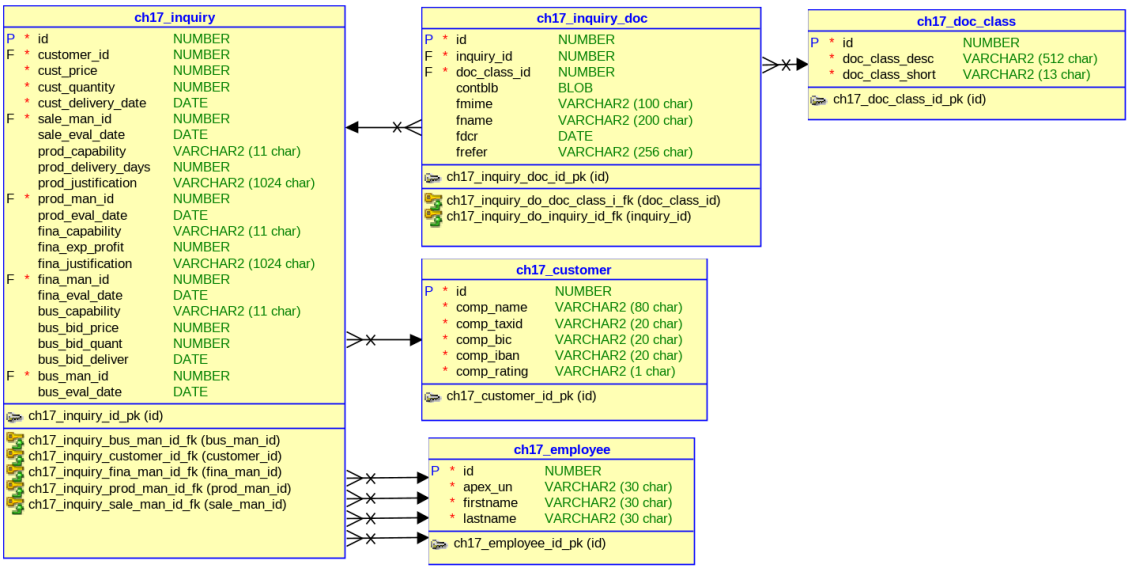


Figure 17.11: Relational data model.

17.6.4 Quick SQL for generating SQL script

Quick SQL is handy tool to accelerate the development of data model. It can be used for prototyping by generating SQL script within APEX (generate application out of script) or to apply reverse engineering to logical data model by importing SQL script into SQL Developer Data Modeler and transform relational to logical model. The data model, presented in this chapter can be described by the following Quick SQL:

```
ch17_employee
apex_un vc30 /nn,
firstname vc30 /nn,
lastname  vc30 /nn

ch17_customer
comp_name vc80 /nn,
comp_taxid vc20 /nn,
comp_bic vc20 /nn,
comp_iban vc20 /nn,
comp_rating vc1 /check 'A','B','C','D' /nn

ch17_doc_class
doc_class_desc vc512 /nn,
doc_class_short vc13 /check 'MANUFACTURING','FINANCIAL','BUSINESS','OTHER' /nn

ch17_inquiry
customer_id num /fk ch17_customer /nn,
cust_price  num /nn,
cust_quantity num /nn,
cust_delivery_date date /nn,
cust_product_item vc50 /nn,
sale_man_id num /fk ch17_employee /nn,
sale_eval_date date,
prod_capability vc11 /check 'YES','NO','CONDITIONAL','N.A.',
```

```
prod_delivery_days num,
prod_justification vc1024,
prod_man_id num /fk ch17_employee /nn,
prod_eval_date  date,
fina_capability vc11 /check 'YES','NO','CONDITIONAL','N.A.',
fina_exp_profit num,
fina_justification vc1024,
fina_man_id num /fk ch17_employee /nn,
fina_eval_date date,
bus_capability vc11 /check 'YES','NO','CONDITIONAL','N.A.',
bus_bid_price num,
bus_bid_quant num,
bus_bid_deliver date,
bus_man_id num /fk ch17_employee /nn,
bus_eval_date date

ch17_inquiry_doc
inquiry_id /fk ch17_inquiry /nn,
doc_class_id /fk ch17_doc_class /nn,
contblb blob,
fmime vc100,
fname vc200,
fdcr date,
frefer vc256
```

One of the benefit of Quick SQL is its compactness and simplicity. Number of lines in Quick SQL is half of the SQL script and number of characters is less than one a third of SQL script. Time and effort for learning Quick SQL is good investment. APEX also provides user-friendly editor with auto-completion, however the advantages of Quick SQL would be more utilized by user who speaks SQL.

17.6.5 Sequence and two stored functions

In this business case, application requirements force us to use some easy-to-understand concepts of Oracle database:

- the mechanism of sequence - it generates integer numbers by defining initial value, maximum value, increment, etc. With this mechanism our application will generate a unique number to identify inquiry and compose the name of the flow instance.
- stored functions in PL/SQL language take input parameters and return a one value. Our application will use two user defined functions: one to get the size of the document stored in BLOB column and one to count the number of documents, associated with specific inquiry.

We can define sequence and stored functions in SQL Workshop > SQL Command. The sequence has name, minimum value, maximum value, increment, start value and some other properties which are at the moment not important for us:

```
CREATE SEQUENCE   "CH17_SEQ_INQUIRY"  MINVALUE 1 MAXVALUE 999999999999
  INCREMENT BY 1 START WITH 100 CACHE 1
  NOORDER  NOCYCLE  NOKEEP  NOSCALE  GLOBAL;
```

We will also define two stored functions which are explained as comments within the code - two dashes at the beginning of the lines are comments. The first function is called ch17_doc_bytesize:

```
create or replace function ch17_doc_bytesize(xdocid in number) return number
```

```
-- Function ch17_doc_bytesize returns the size of the BLOB, stored in contblb
-- column in the table ch17_inquiry_doc for particular row.
-- Function parameter 1 is the ID of of ch17_inquiry_doc table.
-- The length of contblb column is provided by another system level PL/SQL
-- function called dbms_lob.getlength.
is
wdoc_size number := 0;
wblob blob;
Begin
  select contblb into wblob from ch17_inquiry_doc where id = xdocid;
  wdoc_size := dbms_lob.getlength(wblob);
  RETURN  wdoc_size;
END ch17_doc_bytesize;
```

The second function is called ch17_count_docs:

```
create or replace function ch17_count_docs(xinqid in number) return number
-- Function ch17_count_docs returns the number of documents associated with
-- particular inquiry. Function parameter 1 is the ID of inquiry.
-- If the length of the BLOB in the table ch17_inquiry_doc is greater than 0,
-- then we assume that the content is stored. So, function adds 1 to variable
-- wnumdocs, which is used as a counter. When all matching rows in the
-- table ch17_inquiry_doc are examined, than counter function returns counter.
is
wnumdocs number := 0;
CURSOR c_blob is
  select contblb from ch17_inquiry_doc where inquiry_id = xinqid;
Begin
  FOR r_blobs IN c_blob
    LOOP
      if dbms_lob.getlength(r_blobs.contblb) > 0 then
        wnumdocs := wnumdocs + 1;
      end if;
    END LOOP;
  RETURN  wnumdocs;
END ch17_count_docs;
```

17.7 Application interfaces

We suggest the reader to install packaged application for this chapter prior to design of application. Please find link in Supplementary learning material for this chapter. Page numbers quoted, relate to prepared packaged application. Reader can of course develop and use other page numbers to provide the same functionality as described in this section.

Application interface provide functionalities for four actors: sales manager, production manager, financial manager and chief executive manager. It was first generated with wizard and all features included (progressive, web app, about page, access control, activity reporting, configuration options, feedback and theme style selection, total 21 pages). By default a login page is also generated. Home page contains pure HTML to describe a business situation. It provides links to required plug-in and tutorials (see Figure 17.12, Page 1 in packaged application).

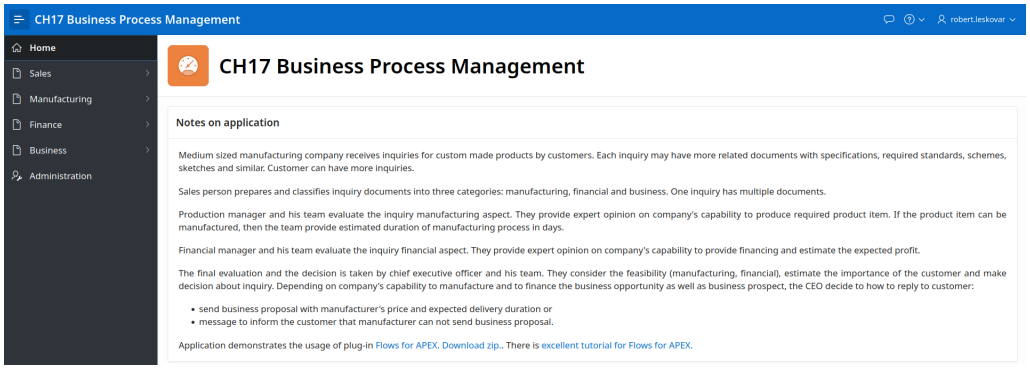


Figure 17.12: Home page of "CH17 Business Process Management" application.

Some LOVs and Plug-ins were set up or included prior to application development in a narrower sense.

17.7.1 List of values in Shared Components

In this application the following Lists of value are defined:

- CH17_LOV_CAPABILITY: type is static, display values are (YES, NO, CONDITIONAL, Not Available), return values are (YES, NO, CONDITIONAL, N.A.). This LOV is used to fill three distinct capabilities (manufacturing, financial and business) .
- CH17_LOV_CRATING: type is static, display values are (Excellent rating, Good rating, Edge acceptable rating, Not acceptable rating), return values are (A, B, C, D). This LOV is used to enter customer rating.
- CH17_LOV_CUSTOMER: type is dynamic, based on table CH17_CUSTOMER, display value is DOC_CLASS_DESC and return value is ID. This LOV is used on forms and reports to contribute to readability and to prevent entering wrong customer ID.
- CH17_LOV_DOC_CLASS: type is dynamic, based on table CH17_DOC_CLASS, display value is COMP_NAME and return value is ID. This LOV is used on forms and reports to contribute to readability and to prevent entering wrong class of the document related to inquiry.
- CH17_LOV_EMPLOYEE: type is dynamic, based on SQL query on table CH17_EMPLOYEE:

  ```
  SELECT FIRSTNAME || ' ' || LASTNAME as d,
  id as r
  FROM CH17_EMPLOYEE order by LASTNAME, FIRSTNAME
  ```

  This LOV is used on forms and reports to contribute to readability and to prevent entering the wrong person who prepares documentation and evaluates inquiry. Concatenate operator is used to form display string with employee first name, space and last name.
- CH17_LOV_INQUIRY: type is dynamic, based on SQL query on table CH17_INQUIRY:

  ```
  SELECT
  'Instance:' || ch17_inquiry.id || ' (Customer: ' || comp_name || ',
     product:' || cust_product_item  as d,
  ch17_inquiry.id as r
  FROM ch17_inquiry, ch17_customer
  WHERE ch17_inquiry.customer_id = ch17_customer.id;
  ```

  This LOV is used on forms and reports to contribute to readability and to prevent entering wrong inquiry ID. Concatenate operator is used to form display long string with a lot of information.

17.7.2 Plug-ins in Shared Components

The following plug-ins in Flows for APEX are imported: Manage Flow Instance, Manage Flow Instance Step, Manage Flow Instance Variables and Viewer. These plug-ins are copied from installed Flows for APEX plug-in.

17.7.3 Sales manager

Application interfaces allow sales manager to:

- initiate workflow instance (see Figure 17.13, Page 2 in application)
- monitor workflow (see Figure 17.14, Page 5 in application)
- confirm that all documents for further evaluations are store in the database (see Figure 17.15, Page 4 in application)
- get insight into all inquiries (see Figure 17.17, Page 6 in application)
- get insight into all documents related to inquiries (see Figure 17.17, Page 8 in application)
- upload document into related column of inquiry document (see Figures 17.18 and 17.20, Page 8 and 9 in application)

Comments on Figure 17.13 (numbers and arrows) present the sequence of actions: 1 (user enters data), 2 (user confirms entered data) and 3 (success or fail message is displayed, data fields are cleared for new entry).

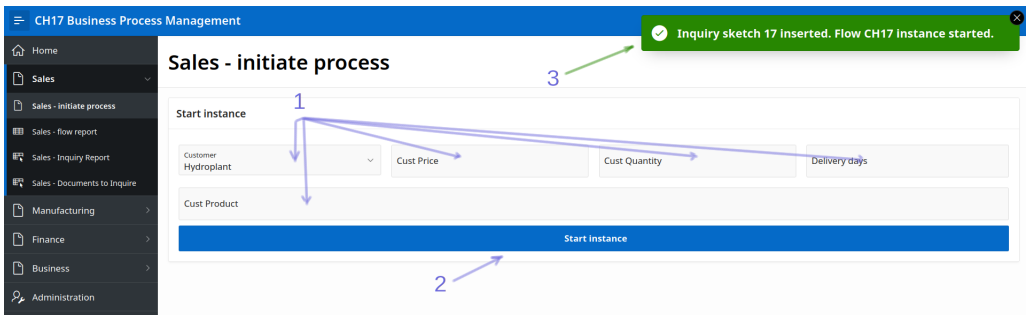


Figure 17.13: Sales - initiated process.

Besides the visible components there is one hidden page item (P2_WID) which simply gets next numerical value from sequence CH17_SEQ_INQUIRY. This was the reason that we define sequence CH17_SEQ_INQUIRY before page design. Button "Start" executes two processes: "Add inquiry" adds one row to table CH17_INQUIRY and four rows in table CH17_INQUIRY_DOC, while "Create and start" process executes action from Flows for APEX plug-in (see details in packaged application, Page 2, Processes). Message of successful completion is returned as seen on Figure 17.13.

Flow report for sales in presented on Figure 17.14.

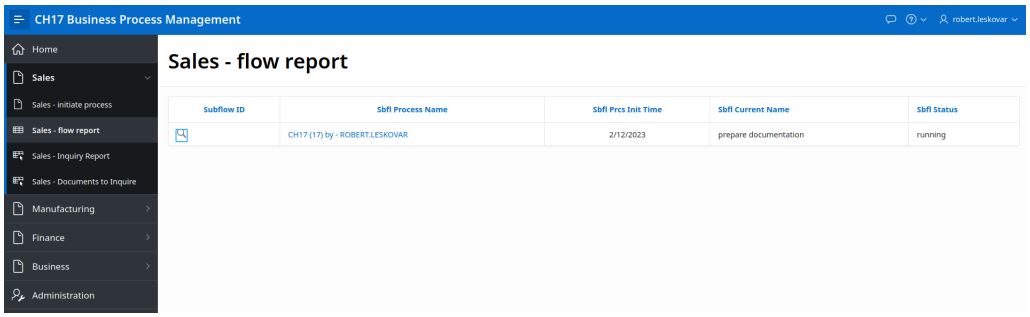


Figure 17.14: Sales - flow report.

Flow report for sales is classic report based on SQL query:

```
select sbfl_id, sbfl_prcs_id, sbfl_process_name, sbfl_prcs_init_ts,
          sbfl_current_name, sbfl_step_key, sbfl_status
    from flow_task_inbox_vw
    where sbfl_dgrm_name = 'CH17' and
      sbfl_current_name = 'prepare documentation'
```

It shows only instances of process "CH17" which have current name "prepare documentation". Two links are provided on Page 4 in application (see Figure 17.14):

1. lens icon in "Subflow ID" column shows the state of the instance in workflow diagram (see Figure 17.15, Page 5 in application).

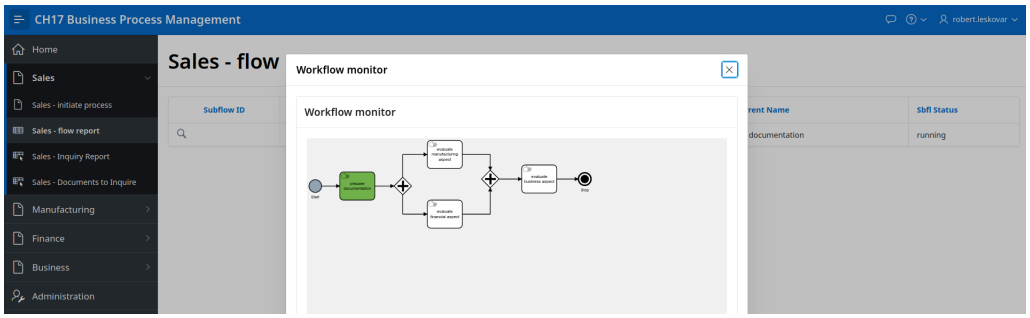


Figure 17.15: Sales - flow diagram for selected instance.

2. "Sbfl Process Name" column shows modal dialog in which sales can confirm that documentation is ready (see Figure 17.16, Page 7 in application).

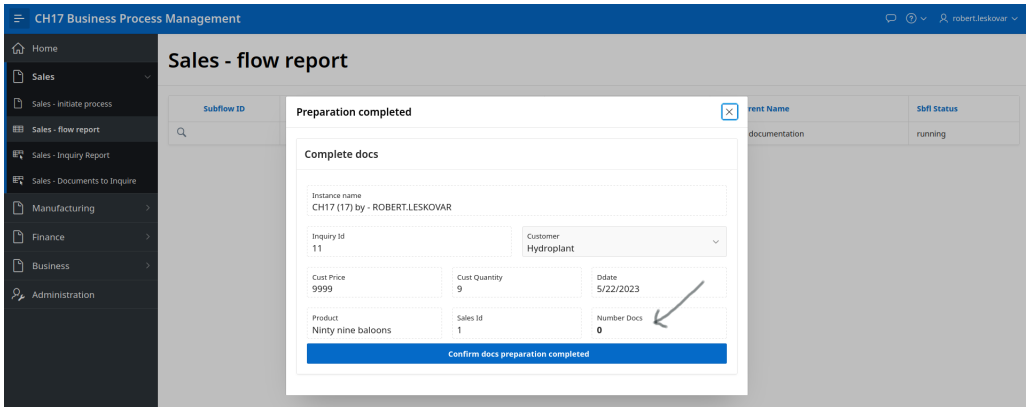


Figure 17.16: Sales - flow report for selected instance.

At this point, documentation is not prepared yet (0 documents uploaded). The counter is a result of PL/SQL block as source:

```
BEGIN
RETURN CH17_COUNT_DOCS(:P7_INQUIRY_ID);
END;
```

Function CH17_COUNT_DOCS (see details in packaged application Supporting objects or after installation in Object Browser > Functions) counts all large binary objects (BLOB) stored in table CH17_INQUIRY_DOC which are associated with the specific inquiry. If the BLOB size is greater than zero, then content is stored, otherwise only empty blob exists.

According to the business rules at least three documents must be prepared (for manufacturing, finance and business).

To get insight in all inquiries, interactive report for sales is prepared (see Figure 17.17, Page 6 in application).

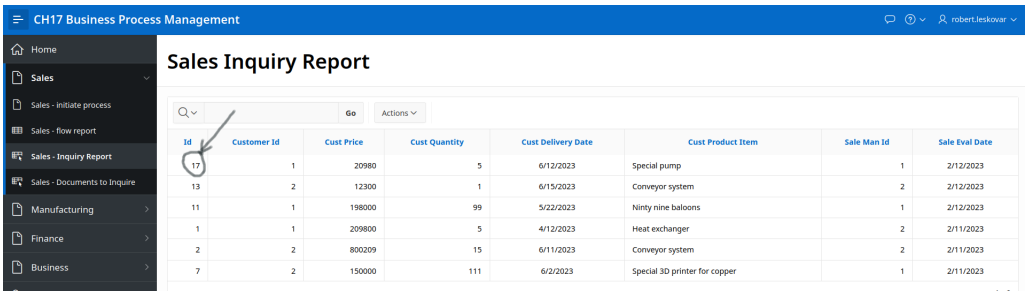


Figure 17.17: Sales - inquiry report.

Sales inquiry report offers searching, and actions (selection of columns, rearranging, filtering, formatting, charting, reporting, exporting etc.). The aim of this report is to provide the sales manager an ID of inquiry which can be used in the process of documentation preparation (i.e. 17 as indicated on Figure 17.17). Next step for the sales manager would be preparation of technical, financial, business and other specification (see Figure 17.18, Page 8 in application).

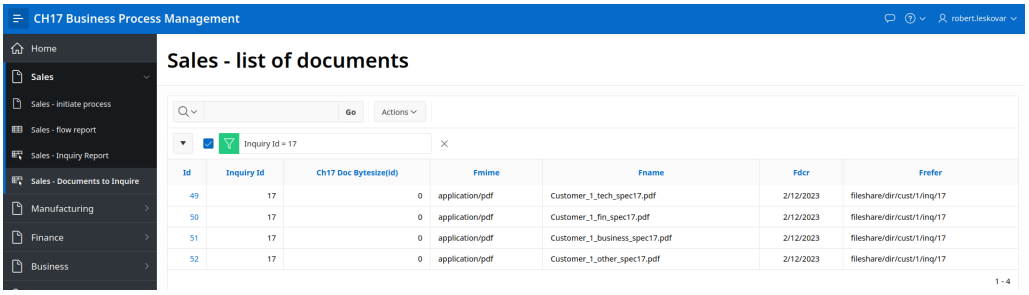


Figure 17.18: Sales - list of documents.

Figure 17.18 shows filtered results (inquiry ID is 17). No BLOB is uploaded - column "Ch17 Doc Bytesize(id)" shows all zero. Page 8 is an interactive report, based on the following SQL query:

```
select ID, INQUIRY_ID,DOC_CLASS_ID, CH17_DOC_BYTESIZE(ID), CONTBLB,
       FMIME,FNAME, FDCR, FREFER
from CH17_INQUIRY_DOC
```

PL/SQL function CH17_DOC_BYTESIZE returns the size of the BLOB. Function takes the ID of the CH17_INQUIRY_DOC as an input parameter and is defined as follows:

```
create or replace function CH17_DOC_BYTESIZE(xdocid in number) return number
is
  wdoc_size number := 0;
  wblob blob;
Begin
  select contblb into wblob from ch17_inquiry_doc where id = xdocid;
  wdoc_size := dbms_lob.getlength(wblob);
  RETURN  wdoc_size;
END;
```

ID of the CH17_INQUIRY_DOC table is used as a link to open a modal page (Page 9) in application (see Figure 17.19).

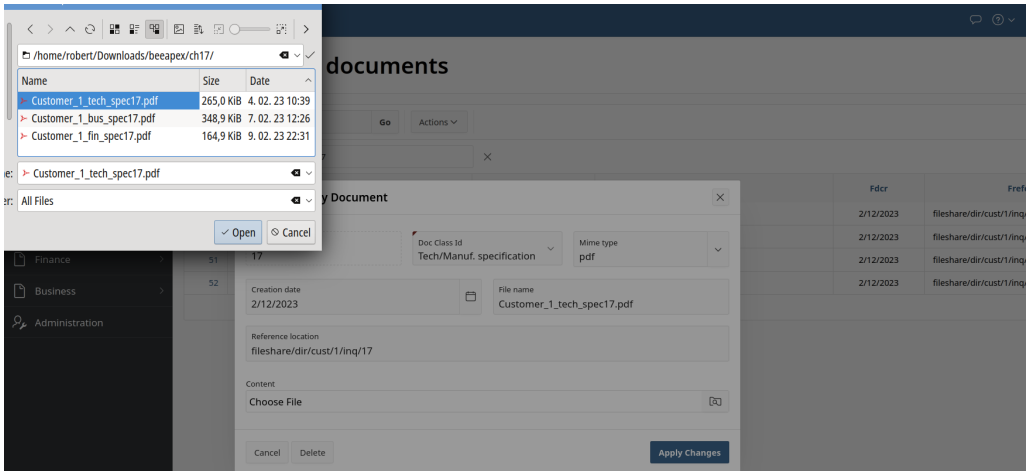


Figure 17.19: Sales - uploading document for inquiry.

By refreshing Page 8, user can be assured that documents are uploaded (see Figure 17.20, Page 8 in application).

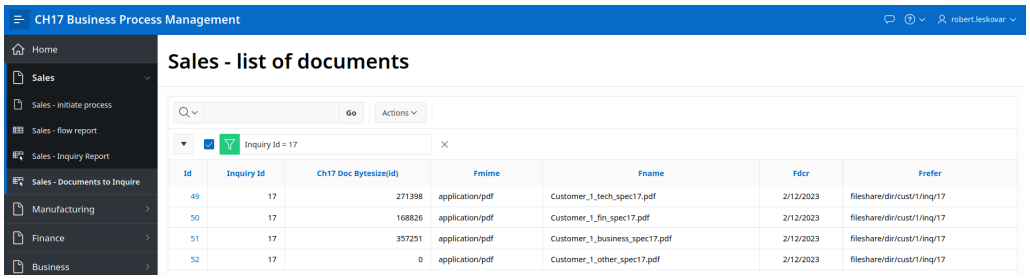


Figure 17.20: Sales - list of documents after uploading and refreshing.

Sales would now confirm that the preparation task is finished. By opening the Sales flow report and clicking the link on process name, the modal page opens (see Figure 17.21, Page 8 in application).

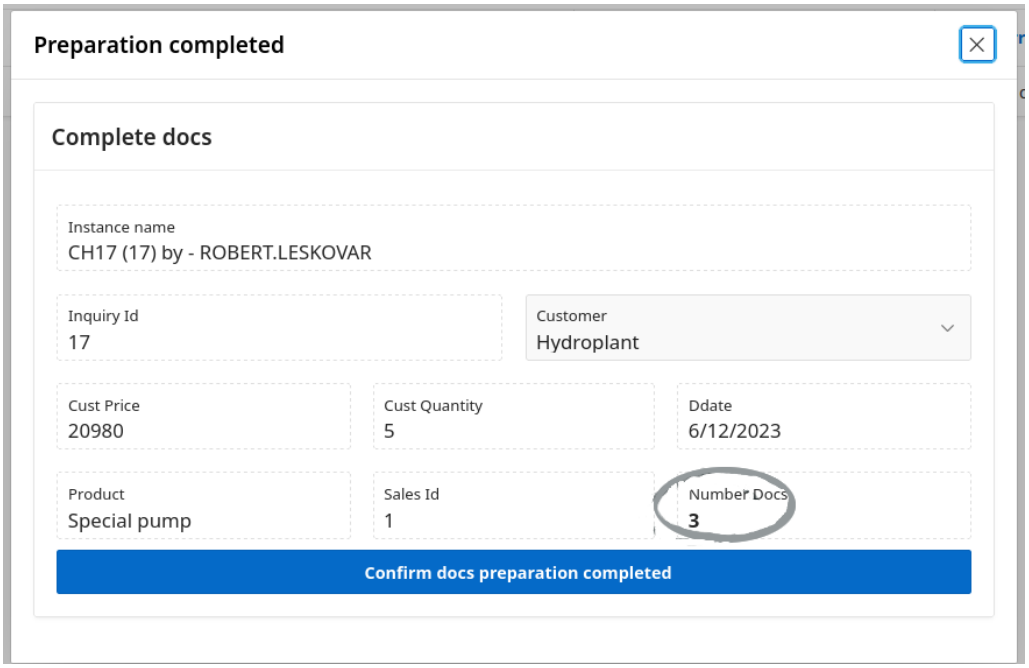


Figure 17.21: Sales - report for selected instance after uploading three documents.

By clicking "Subflow ID", sales manager gets visual confirmation, that document preparation task is completed (see Figure 17.22, Page 5 in application).

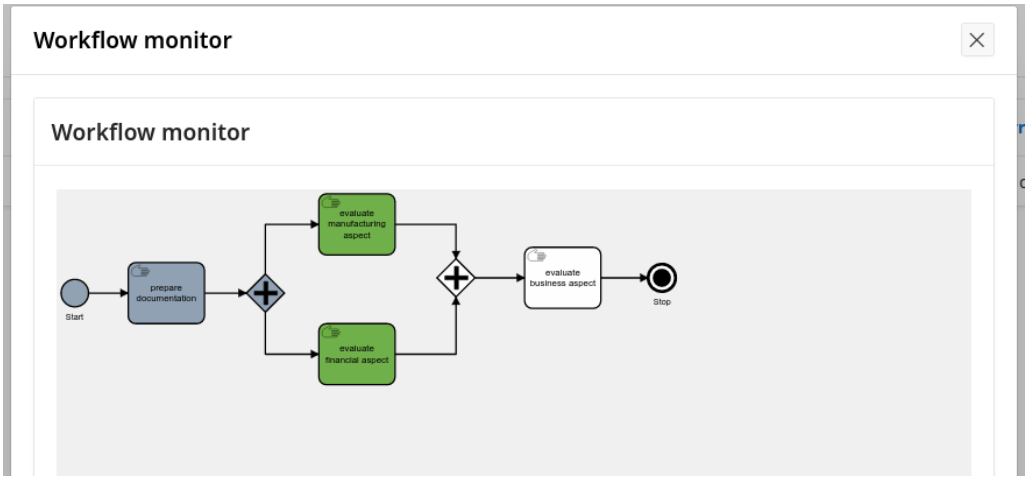


Figure 17.22: Sales - flow report for selected instance after uploading three documents.

17.7.4 Production manager

Production manager can:

- monitor workflow (report on Figure 17.23, Page 20 in application and diagram on Figure 17.24, Page 5 in application)
- enter manufacturing evaluation (Figure 17.25, Page 21 in application)

By clicking "Manufacturing flow report" production manager can track unevaluated inquiries (i.e. report on Figure 17.23, Page 20 in application).

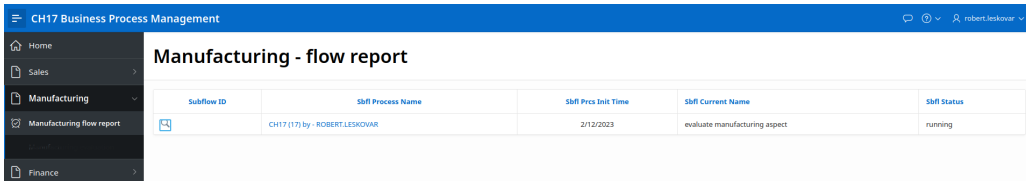


Figure 17.23: Manufacturing - flow report.

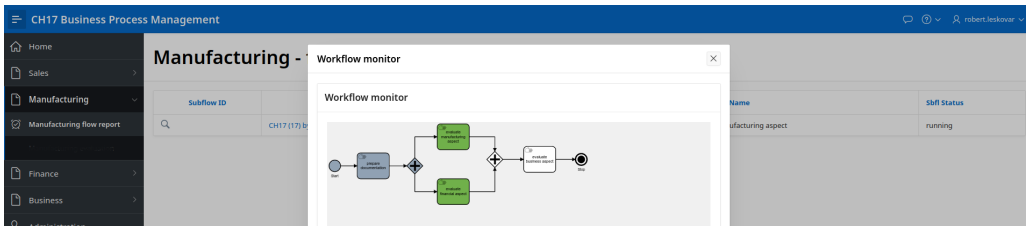


Figure 17.24: Manufacturing - BPMN diagram - state of instance.

Selecting lens icon under "Subflow ID" opens Page 5 (see Figure 17.24) which shows the instance of the flow CH17. Instance is in the parallel gate which means that both tasks (manufacturing and financial evaluation) must be finished before the next task (business evaluation) starts. Manufacturing evaluation can be entered by clicking "Sbfl Process Name" link on Page 20 (see Figure 17.25, Page 21 in application).

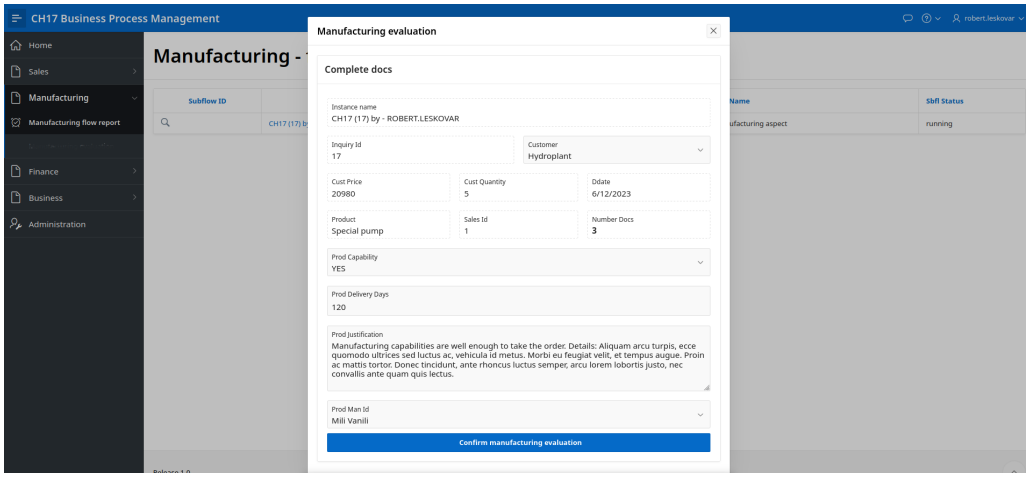


Figure 17.25: Manufacturing evaluation.

Pressing the button "Confirm manufacturing evaluation" will move the instance flow in next state, depicted on Figure 17.26.

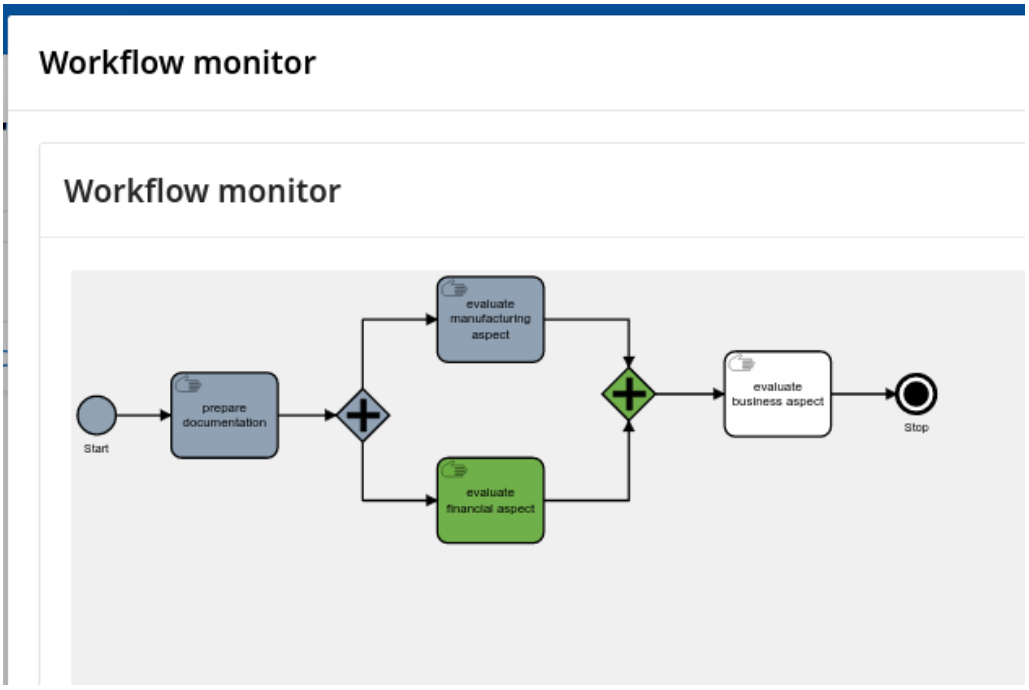


Figure 17.26: Manufacturing evaluation finished, instance waiting to financial evaluation.

By refreshing Page 20, report displays "Manufacturing has no flow instances".

17.7.5 Financial manager

Financial manager can:

- monitor workflow (report on Figure 17.27, Page 30 in application and diagram on Figure 17.28, Page 5 in application)
- enter financial evaluation (Figure 17.29, Page 31 in application)

By clicking "Financial flow report" financial manager can track unevaluated inquiries (i.e report on Figure 17.27, Page 30 in application).

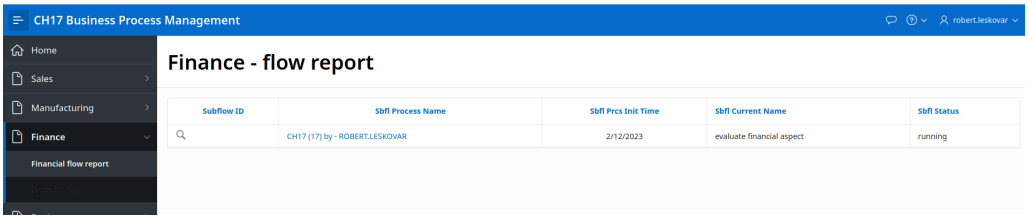


Figure 17.27: Finance - flow report.

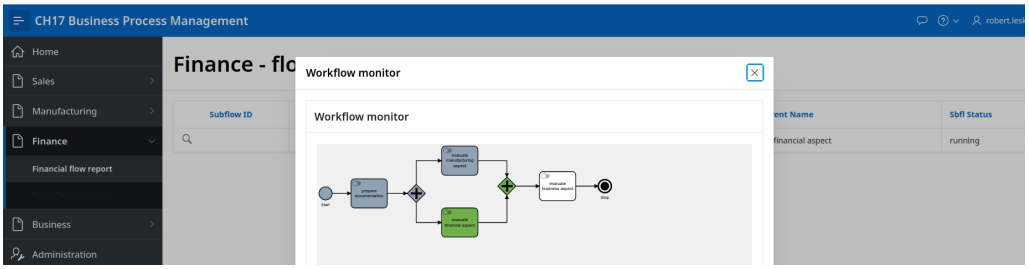


Figure 17.28: Finance - BPMN diagram - state of instance.

By clicking lens icon under "Subflow ID" application opens Page 5 (see Figure 17.28) which shows the instance of the flow CH17. Financial evaluation can be entered by clicking "Sbfl Process Name" link on Page 30 (see Figure 17.29, Page 31 in application).

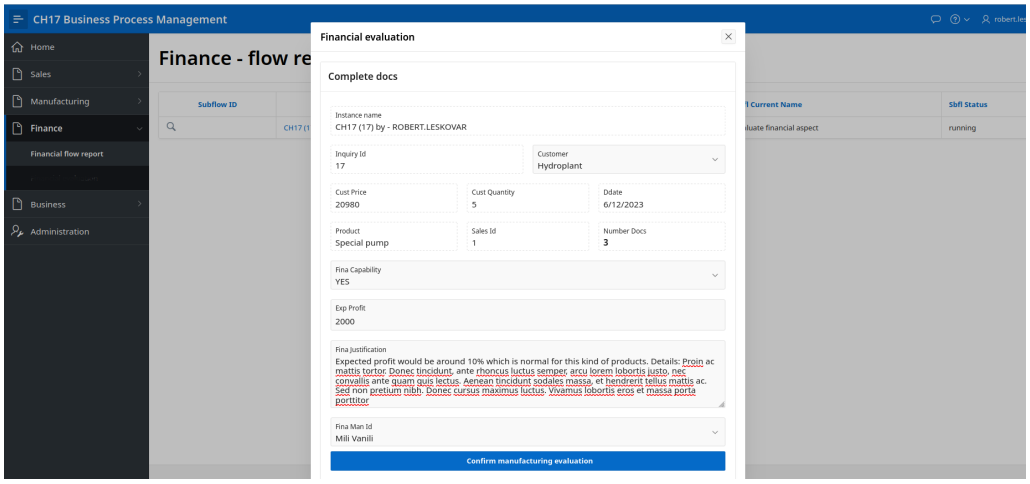


Figure 17.29: Financial evaluation.

Pressing the button "Confirm financial evaluation" will move the instance flow in next state, depicted on Figure 17.30.

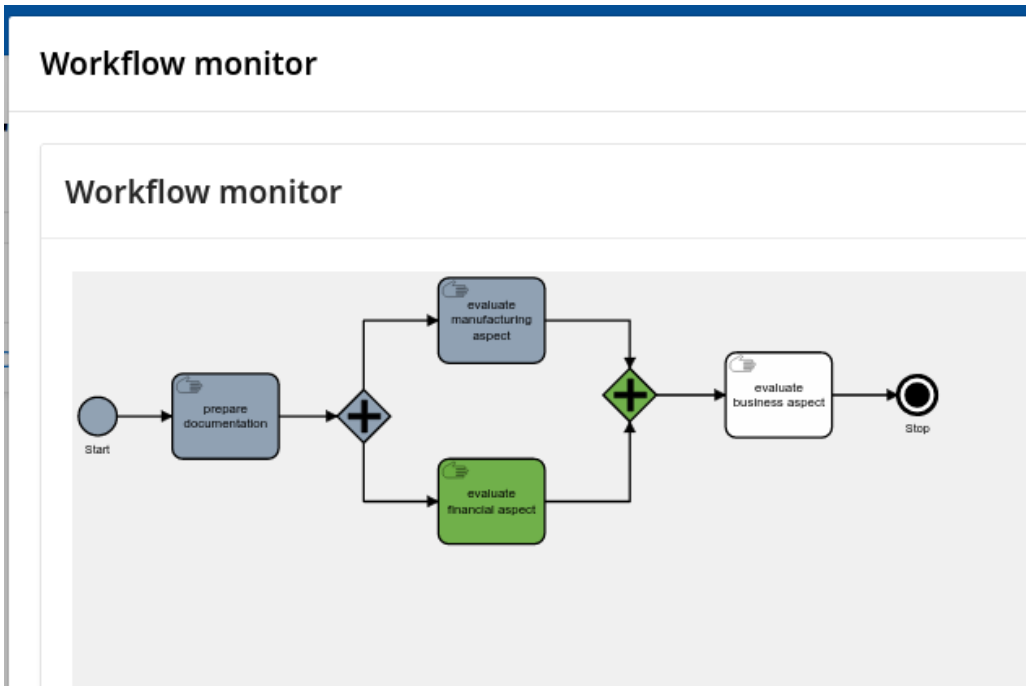


Figure 17.30: Financial evaluation finished, instance waiting to business evaluation.

By refreshing Page 30, report displays "Finance has no flow instances".

17.7.6 Chief executive officer - business manager

Business manager can:

- monitor workflow (report on Figure 17.32, Page 40 in application and diagram on Figure 17.31, Page 5 in application)

- enter business evaluation (Figure 17.33, Page 41 in application)

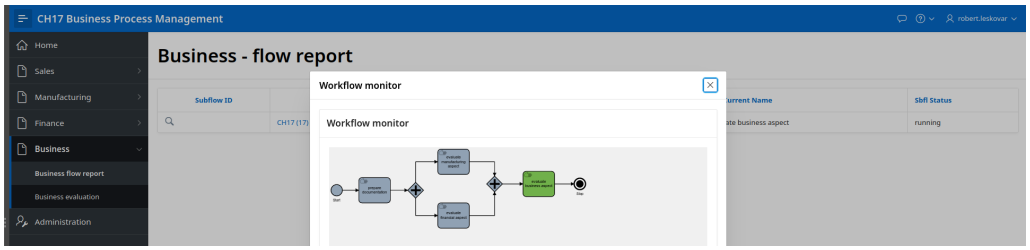


Figure 17.31: Finance - BPMN diagram - state of instance.

By clicking "Business flow report" CEO can track unevaluated inquiries (i.e report on Figure 17.32, Page 40 in application).

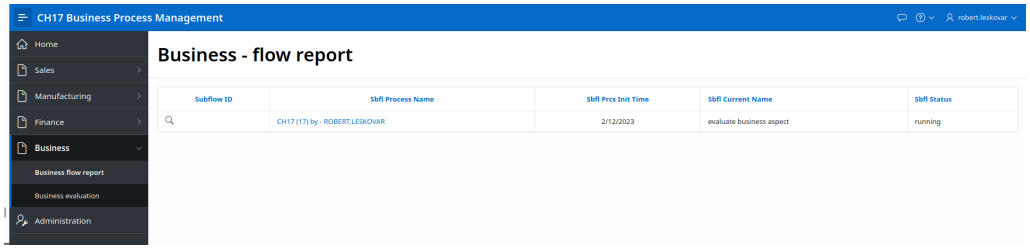


Figure 17.32: Business - flow report.

Click on the lens icon under "Subflow ID" will open Page 5 (see Figure 17.31) which shows the instance of the flow CH17. Business evaluation can be entered by clicking "Sbfl Process Name" link on Page 40 (see Figure 17.33, Page 41 in application).

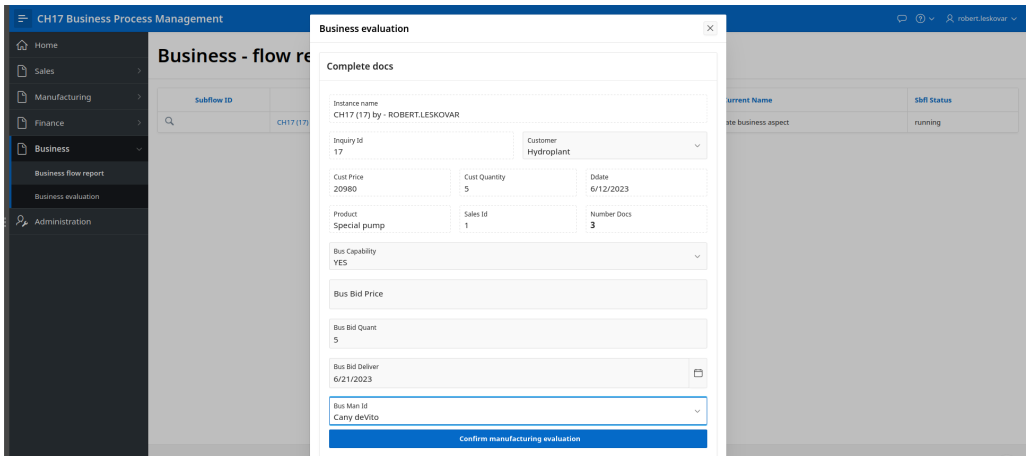


Figure 17.33: Business evaluation.

Pressing the button "Confirm business evaluation" will move the instance to the end of workflow, depicted on Figure 17.34.

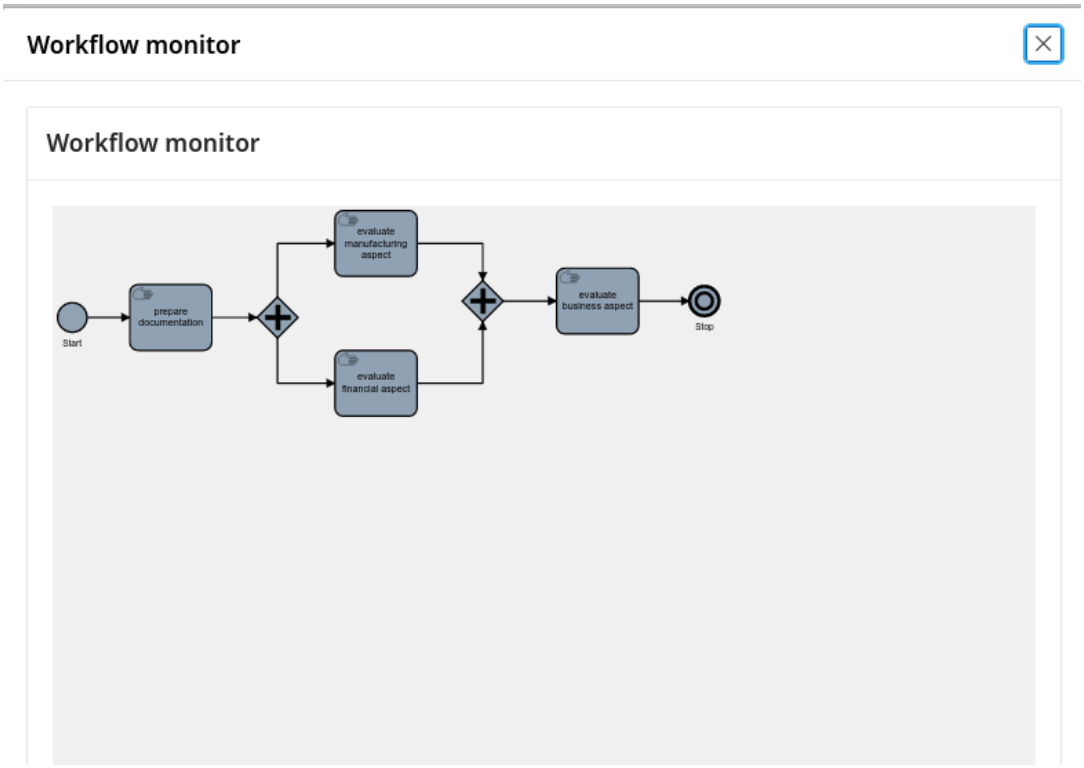


Figure 17.34: Business evaluation finished, instance terminated.

By refreshing Page 40, report displays "Business has no flow instances".

17.8 Linking application with Flows for APEX

The application, developed in chapter 17 applied two functions provided by Flows for APEX:

- start an instance on the Page 2
- show the state of the instance on Page 5
- push forward instance to the next task on Pages 7, 21, 31, 41

Starting an instance requires a process on the page. In identification area of the process type: "Flows for APEX - Managing Flow Instance [Plug-In]". Action should be "Create and Start". We set Flow instance Name to:

```
CH17 (&P2_WID.) by - &APP_USER.
```

Remember that page item P2_WID contains unique number, provided by sequence. So the name is concatenated string, composed of fixed string "CH17", space, open parenthesis, unique sequence number, close parenthesis, space, fixed string "by - " and current user. Select Flow is using "Static text" and the Static text is "CH17". Success Message is "Flow CH17 instance started." and Error Message is "Flow CH17 instance NOT started.".

To present the state of the instance on Page 5 the page wizard for plug-in page is used. The page contains viewer plug-in and hidden item - P5_PRCS_ID. See settings on Figure 17.35.

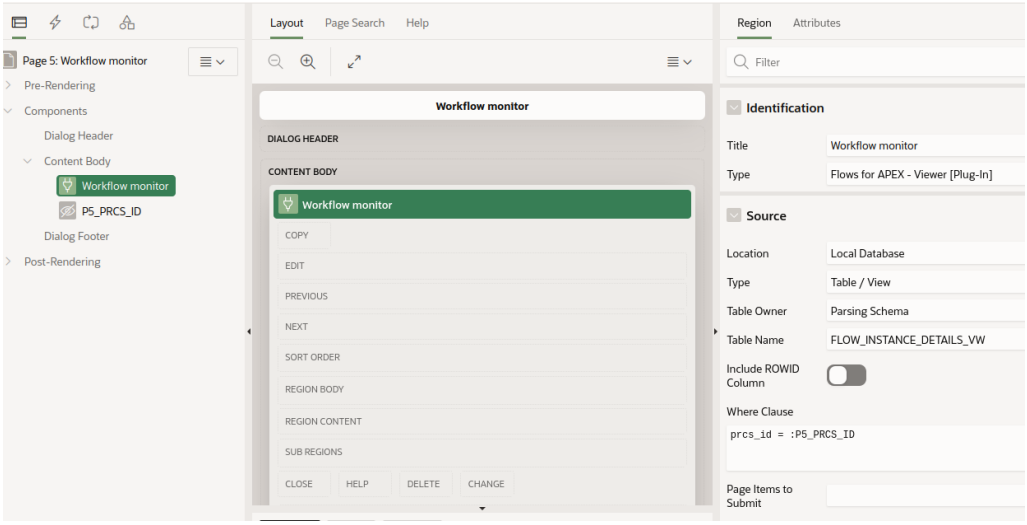


Figure 17.35: Showing the state of the instance on BPMN diagram.

To push instance forward (i.e. Pages 7, 21, 31, 41) define page items as shown on Figure 17.36.

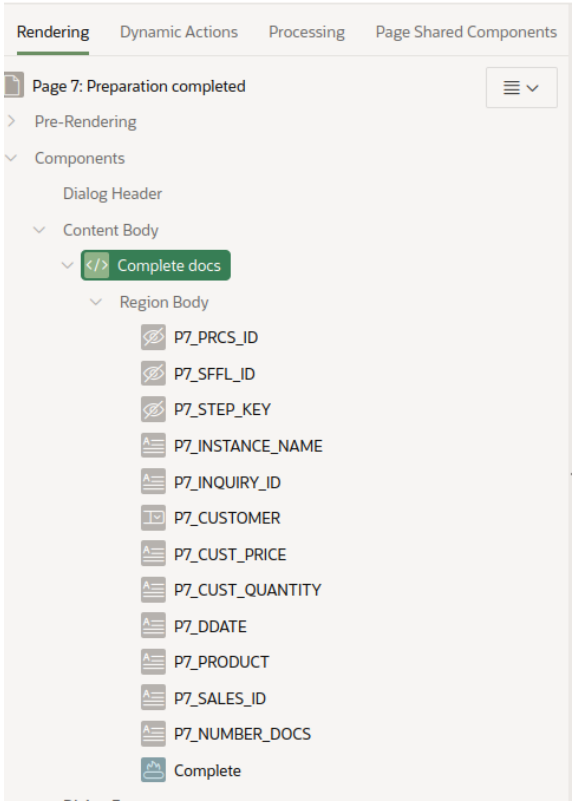


Figure 17.36: Setting page items.

Only first four page items (from P7_PRCS_ID to P7_INSTANCE_NAME) are relevant to Flows for APEX integration. Create a new process which is triggered by a button. Process is defined as "Manage Flow Instance Step" plug-in. Set Action, Process ID Item, Subflow ID item and Step Key as shown on Figure 17.37.

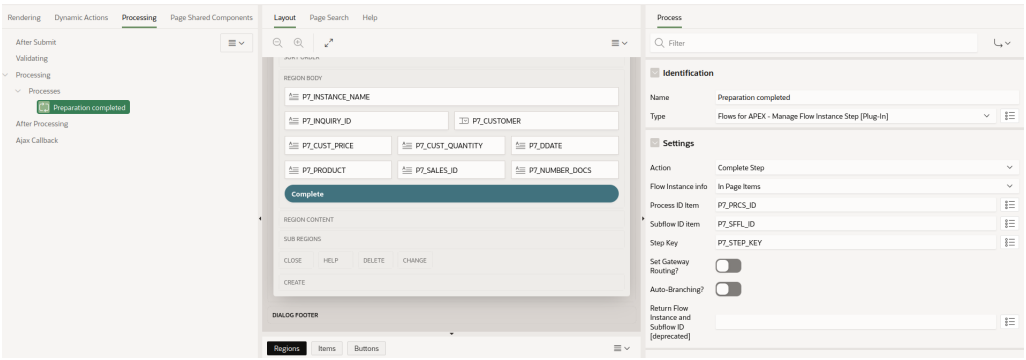


Figure 17.37: Completing step in Flows for Apex.

17.9 Define user roles

User roles can be defined in Application Access Control in Shared Components (i.e. Figure 17.38).

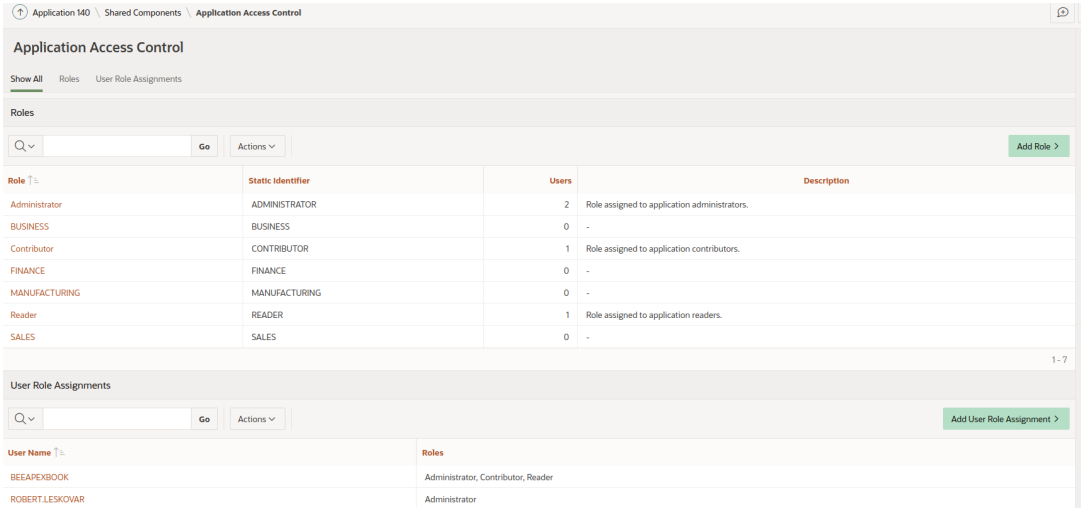


Figure 17.38: Define roles and user roles in Application Access Control menu.

At this time we will skip the details and implementation for this specific case. For fast overview see Chapter 13 or for thorough insight study APEX documentation on authorization.

17.10 Testing and correcting errors

For beginner, the testing process would be very simplified as you put your hands on the keyboard and mouse to develop with APEX, you are on a certain path to make errors and to correct it. Do not be afraid to make errors. Take the lesson from each one that you solved by yourself. Share the knowledge in open APEX community. The discipline of software testing is a huge one and you can learn it by doing it. There is no other way. Developers are the first line of defense against software errors, faults, and failures.

Surprisingly, failed software development projects in most cases went the wrong way before a single line of code was written. The application, developed for this chapter, has software errors too. Some of them will be discovered and debugged by clever readers. Some errors are also deep inside APEX itself. We, the writers and developers of this chapter hope that testing and debugging will present a pleasant challenge to readers.

Removing errors from software is sometimes fun and sometimes a curse. Just do not give up too soon!

17.11 Supplementary learning material

You can find the following supplementary learning material:

- exported packaged application and scripts
- workflow model called CH17
- video guides

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find the textbook in Books section, scripts in folder Part 2 > Chapter17 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

17.11.1 Exported application

Exported application is packaged. Installation creates tables, sequence, functions as well as populate data. De-installation removes all data base objects used in this application.

Packaged application is tested and it will run in new workspace if the following requirements are meet:

- privilege "create job" is granted to schema which hosts new workspace. Database administrator can grant this privilege by command "grant create job to <schema>". APEX workspace is associated with database object called schema. If you are using OCI (free, paid or provided by Oracle Academy) than use web OCI interface (Autonomous Database > specific instance > Database action) or enter directly web database administrator interface. You can execute queries and scripts. Figure 17.39 presents the outlook of the interface with entered command (1) and database response (2).

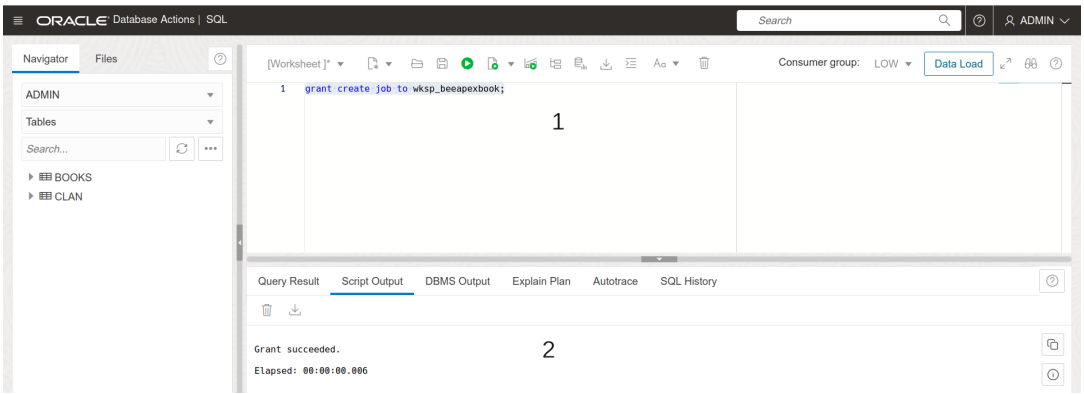


Figure 17.39: Granting "create job privilege" to workspace.

If you use other tools (SQL Developer, TOAD) the procedure is pretty much the same, except the outlook of the interface.

- Flows for APEX must be installed in new workspace (see prior instructions).
- Workflow model called CH17 is imported into Flows for APEX. File *CH17.bpmn* is stored in directory Flows4APEX in packaged application as static file. Go to Shared Components > Static Application Files and download it to your local computer. The same file is also available in learning materials scripts. Than import file *CH17.bpmn* with Flows for APEX. Log in into application Flows for APEX. Select Flow management and import the file (see Figures 17.40 - step 1 and Figure 17.41 - step 2).

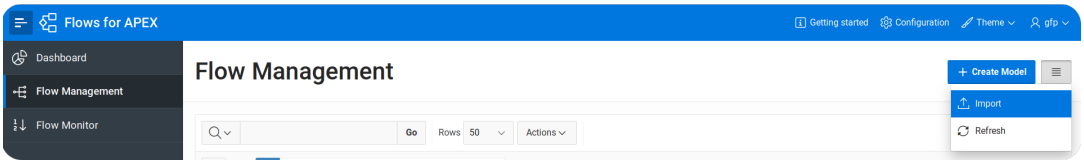


Figure 17.40: Import workflow called CH17 into Flows for APEX - step 1.

The meaning of the numbers on Figure 17.41 are: 1 (arbitrary category), 2 (name of the model must be exactly CH17), 3 (navigate your file explorer to CH17.bpmn) and 4 (confirm by pressing Import)

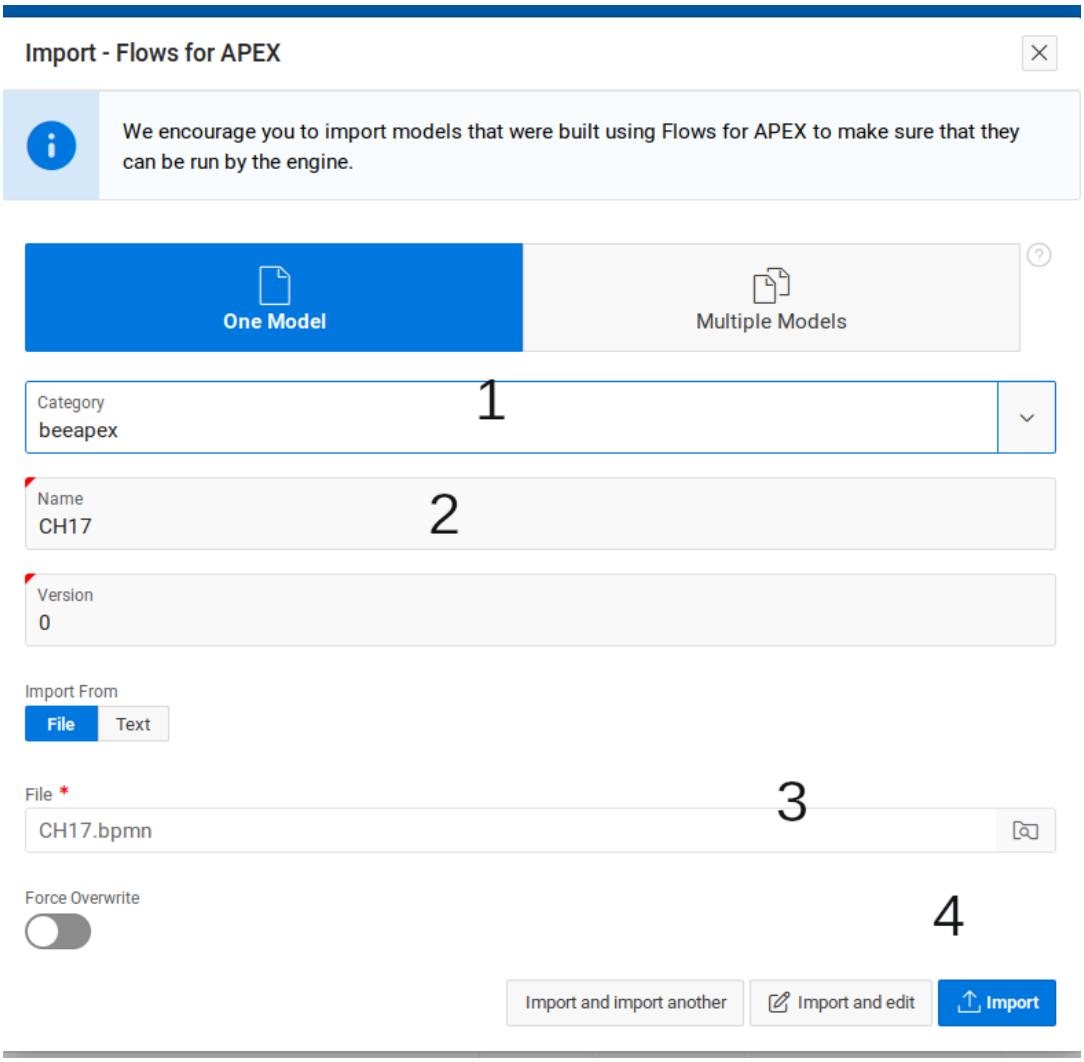


Figure 17.41: Import workflow called CH17 into Flows for APEX - step 2.

- After installation of the packaged application add a user role assignment (Shared components > Application Access Control > Add User Role Assignment).

If any of the above requirements is not met then the imported application will crash. It is necessary to clear the web browser cookie (i.e. Firefox: Settings > Cookies and Site Data > Manage Data) after application crashes due to unmet requirements.

17.11.2 Video guides

Video guide shows initial steps in application development.

17.12 Questions

1. Explain what is Flows for APEX?
2. What are the artifacts of business process model presented with BPMN diagram?
3. Which functions by Flows for APEX were used in this application?

17.13 Answers

1. Flows for APEX is called a feature, a plug-in and an application. All three meanings are correct. Feature is meant when we discuss in the context of process modeling. Plug-in relates to development of custom application. Application "Flows for APEX" is aimed to design workflows and administer the instances in the case of issues.
2. Artifacts of business process model presented with BPMN diagram are:
   - Start Event: Indicates the starting point of a process.
   - Task: Represents a single unit of work, such as an activity or a step in the process.
   - Gateway: Represents a decision point in the process, such as a fork in the flow or a conditional branch.
   - Sequence Flow: Connects elements in the diagram and represents the flow of control from one element to another.
   - End Event: Indicates the end of a process.
   - Pool/Lane: Represents a grouping of related tasks, used to define the roles and responsibilities of different participants in the process.
   - Message Flow: Represents the flow of messages between participants in a process, such as a communication between two systems.
   - Data Object: Represents a piece of data that is used or produced in a process, such as an invoice or a customer record.
   - Data Store: Represents a container for storing data that is used or produced in a process, such as a database or a file system.
3. To start instance, action "Create and Start" was used. To present instance on BPMN diagram, plug-in page wizard used Viewer component. To push forward to next step the component "manage Flow Instance Step" was used in application.



18. GreenDi – Exchange of Plants and Seeds

VJERAN STRAHONJA, DIJANA OREŠKI, DARKO ANDROČEC, ANA KUTNJAK

18.1 Business view of the case

A short business overview of the GreenDi platform is described in chapters 14 and 15. Chapter 18 refers to the part of the platform dedicated to the exchange of plants and seeds. All registered users of the platform can participate in this exchange, presenting their own offer, or responding to someone else's offer. At the same time, the system should enable searching for offers by different filters, sorting columns and similar functionalities. In addition, communication between the bidder and other users related to a specific offer should be enabled. Communication is in the form of chat.

18.2 Problem definition

The following sentences describe the problem:

- The basic functionalities of exchanging seeds and plants are submitting one's own offers (bid and ask) and responding to other people's offers (bid and ask), which is reserved for members.
- Members can search the catalog by different criteria and open a table with open offers for each type of plant from the catalog.
- The platform is not commercial, ie plants are not sold or bought on the platform, but exchanged and donated. Therefore, plant transactions are not subject to taxation.
- The offer includes the plant that is offered or requested, quantity, place, bidder, comments, bid date, bid validity date, instructions for picking up or sending, status, etc.
- Offers (bid or ask) are presented in tabular form, with the possibility of sorting by columns and searching and filtering by various criteria (type of plant, date of offer, member, place, status, etc ...).
- Each member can respond to any open offer.
- The conversation on any bid takes the form of a public, or private chat.
- The basic unit for conversation (chat) is a message.
- The message contains information about the member who sends it (automatically generated), the text of the message and visibility (public, private).
- The system should support all phases of lifecycle management of offers (create, change, close, delete and archive). The bid is created, modified, closed, deleted and archived by the bid owner. The administrator can delete an offer if it violates some rules and can archive

Table 18.1: Use case description: exchange of Plants and Seeds.

Keyword	**Value**
ID:	*ch18-01*
Title:	*Exchange of Plants and Seeds*
Description:	*Plants Exchange.*
Primary Actor:	*Any person*
Preconditions:	*Basic information about the plant is open for search and viewing without any registration.*
Postconditions:	*If user enters new offer than new data is stored.*
Main	-
Success Scenario:	1. user scrolls up and down the catalog until plant is found or applies filter search 2. user opens the list of offers for selected plant or all plants 3. user scrolls up and down the list of offers or applies filter search 4. user opens new offer for selected plant, enters details and exposes the new offer 5. user clicks on selected offer, opens the pop-up window with detail of selected offer 6. user can respond to any active offer with a counteroffer or acceptance, or start a chat. 7. chat is in the form of messages and replies to messages 8. link is visible to the offerer and the other participating users
Extensions:	• *List of offers (report)* • *Details of the offer (new or existing)* • *Chat (list of messages)*
Frequency of Use:	*Approx. maximum is 100 per day.*
Status:	*Development status*
Owner:	*public, anonymous user*
Priority:	*high*

inactive offers after a certain time.

18.3 Use cases

18.3.1 Narrative description of use case

Communication with UC Offer management is available to every registered user or member of the GreenDi platform. A prerequisite for communication is a successful Log-In (included UC). The UC Browse catalog of plants functionality is an extension of the UC Offer management. The administrator is also an external user of UC Offer management. At the same time, he can view all offers and chats without restrictions and can change the status of each offer.

18.3.2 Semi-structured description

Table 18.1 presents UC.

18.3.3 Use case diagram

The above story is depicted on use case diagram 18.1.

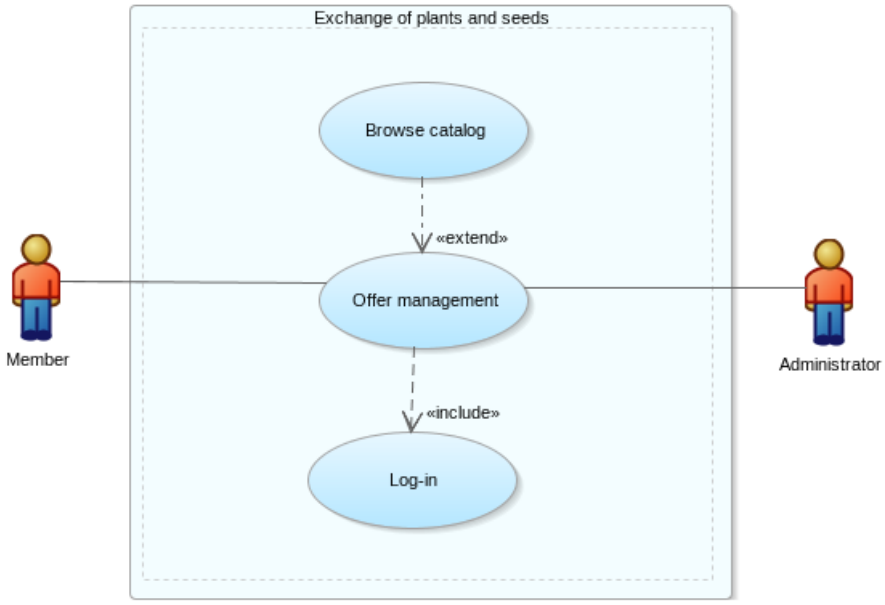


Figure 18.1: Use case diagram - Exchange of Plants and Seeds

18.4 Data model

This section explains data model.

18.4.1 Narrative description of data model

The logical data model (Entity-Relationship model) is actually an upgrade of the data model from Chapters 14 and 15). The Plant and Thematic unit entity types were taken from Chapter 14, and User from Chapter 15. The basic entity types for Exchange of Plants and Seeds are Offer and Message. Offer is described by the following attributes: IDoffer (identification attribute); Bid or ask (describes if plant or seed is offered or requested), Quantity (of plant or seed), Location (where plant or seed can be picked-up, or delivered), Bidder (the user who bids or asks), Comments (comments on quality, instructions for delivery etc.), Date (date of the offer), Valid to (validity date), Status (active, withdrawn, realized). Offer is an original offer, or a response to an original offer. This means that each offer is a response to 0, or one offer, AND has 0, 1, or more responses. Each Offer is offered by one and only one User. 0, 1, or more Messages are related to the Offer. Each Message is related to 1 and only 1 Offer, sent by 1 and only 1 User (sender) and intended for 1 and only 1 user (recipient).

18.4.2 Logical data model

Logical data model is presented on Figure 18.2.

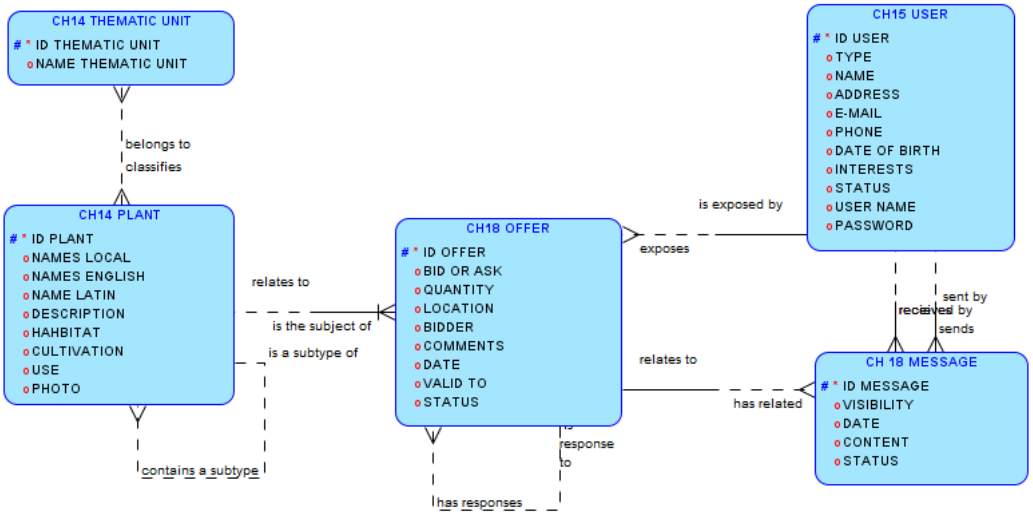


Figure 18.2: Logical data model.

18.4.3 Relational data model

Automatic transformation from logical data model to relational data model in Oracle SQL Data Modeler is provided by function *Engineering to relational*. The result is shown on Figure 18.3.

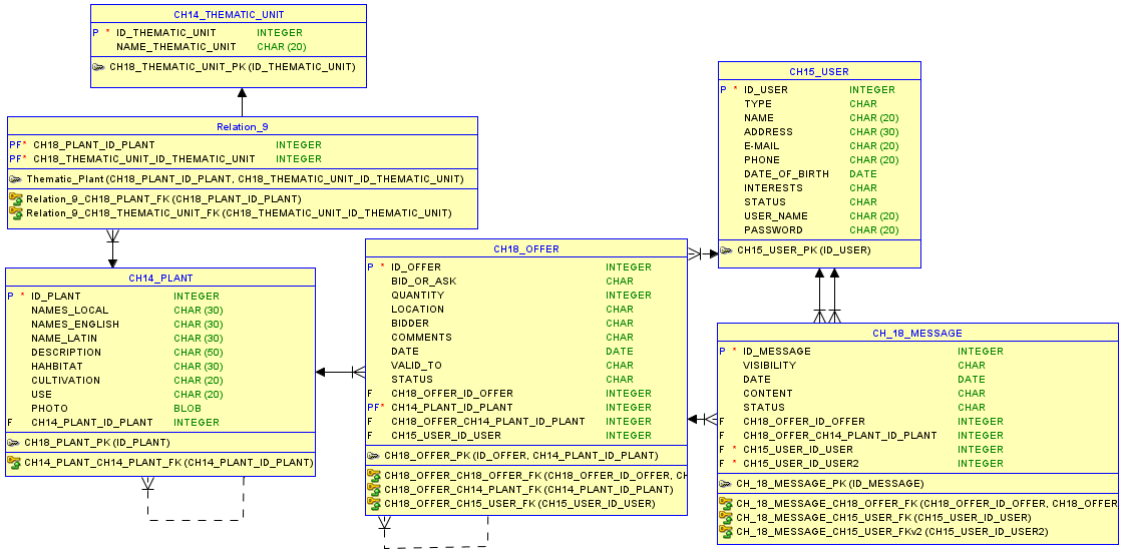


Figure 18.3: Relational data model.

Let now Oracle SQL Data Modeler generates SQL script. Select all tables on relational model and use function *File* > *Export* > *DDL File* to get a script. We can import the generated script in APEX and execute it.

18.5 Application interfaces

Offers are presented as interactive report (see Figure 18.4).

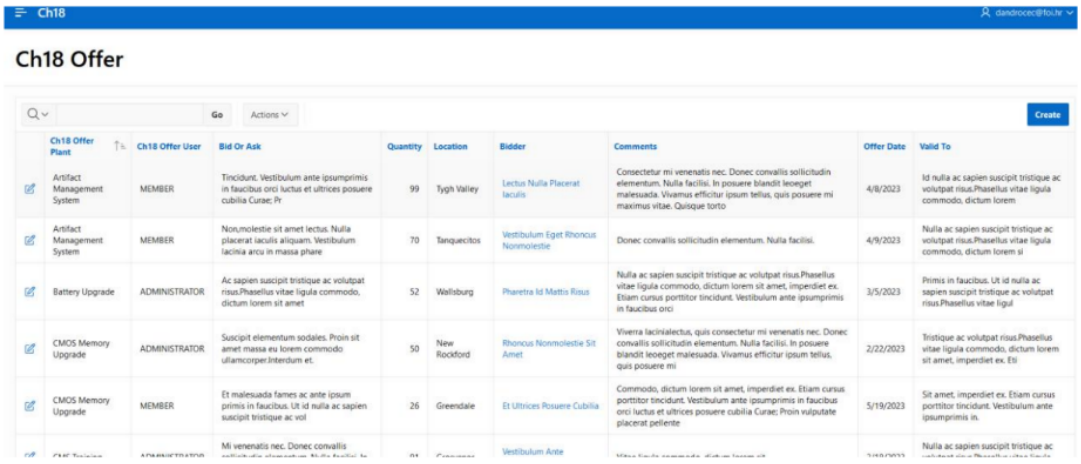


Figure 18.4: Offers - interactive report.

Message form is presented on Figure 18.5.

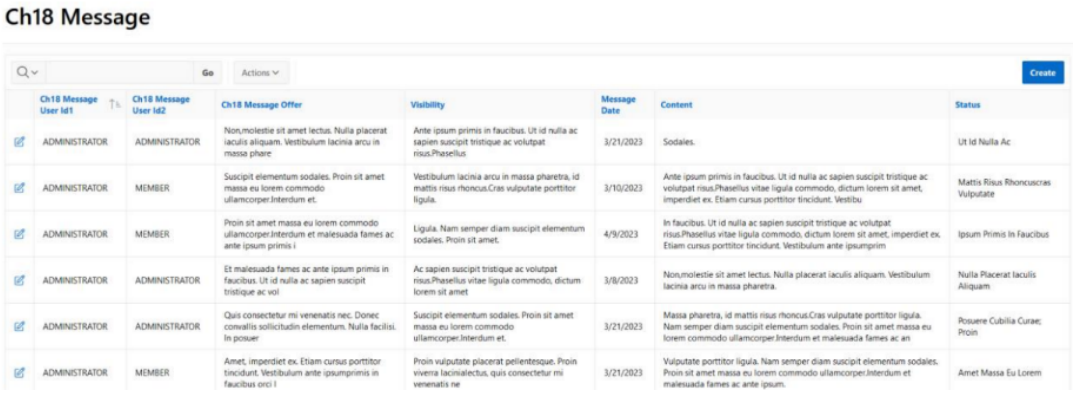


Figure 18.5: Message form.

18.6 Supplementary learning material

You can find the following supplementary learning material:

- exported application
- scripts for creating, dropping and inserting
- video guide

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter18 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

18.6.1 Exported application

Exported application is packaged. Installation create tables and populate data. De-installation removes all data base objects used in this application.

18.6.2 Video guides

Video guide show every step in application development.

18.7 Questions

1. How could we make the offer visible with a delay, from a certain moment, when the bidder wants it?
2. Can this app be used to chat about a topic other than actually exchanging plants and seeds? How?
3. In table CH18_OFFER, the column VALID_TO has CHAR datatype. What are the potential problems if a date is defined as CHAR type?

18.8 Answers

1. First, we would add a column in CH18_OFFER table. Than we would correct the form which enables entering new offers. Finally, we would correct the query on offers report by adding condition (where new_column <= SYSDATE).
2. Theoretically, it is possible. The moderator of the chat should open a dummy offer, and all users who come in with their dummy counteroffer could participate in the chat.
3. Storing dates as CHAR can introduce data integrity issues, sorting problems, limited date operations, formatting difficulties and some other problems. It is generally recommended to use the appropriate date data types provided by Oracle, such as DATE or TIMESTAMP, for accurate and efficient date storage and manipulation.



19. Book review management system

A. KUTNJAK, L. HRUSTEK, A. BAGGIA AND R.LESKOVAR

19.1 Business view of the case

The book review management system was designed with the aim of managing reviews of books available for sale on the arbitrary on-line platform. The company (i. e. Amazon) is thinking about managing the catalog of available books in such a way that, based on user reviews, it decides on the books offer by adding or removing it. Using the review management system, the company collects user reviews, creates a unique database and decides on further action based on the collected data. The basic stakeholders of the database are the administrator and the user, the book reader. The use case and idea are inspired and offered by Kaggle - online community platform for data scientists and machine learning enthusiasts. Original data is available www.kaggle.com. Use cases and data model are profoundly modified and supplemented. The aim of this business case is to demonstrate development capabilities of APEX relying only on wizards. With the exception of Quick SQL (total 31 short lines) no other code will be written by developer.

19.2 Problem definition

Review management system has the following functionalities:

- Given the wide range of books in the global offer, data on their reviews and rating score are collected.
- The common view of the platform includes a list of books with information about the title, authors, content description, publisher, published date, categories and reviews.
- The administrator's options include adding books to the platform's database, as well as editing and selecting them by category.
- Since the database allows users to add reviews, the administrator can view reviews and review comments. Based on that, the administrator makes decisions about actions per book (adding or removing).
- The user has the option of registering to the platform, which enables him to view book details, but also to review them and comment reviews.
- Reviews on the user side have two benefits. First, based on the analysis of the book reviews, the potential user decides to buy book. Second, based on the existing reviews, company makes actions regarding books, i.e., adding or removing them from the catalog.

19.3 Use cases

19.3.1 Narrative description of the use case

The administrator is in charge of entering or changing data about books within the company review management system and is enabled to view various statistical indicators related to books. The Amazon review management system is based on user registration on the platform. By registering, users can view the catalog of books available on the platform. The book catalog contains detailed information about available books and is the basis for further actions within the platform. The books are classified into categories, which makes searching easier and allows introduction of user personal preferences. Registered user can add review and add comment on review.

19.3.2 Semi-structured description

Activities to enter new book data are assigned to the administrator. It complements the current database with new books, which results in the addition of the offer in the book catalog. The frequency of use of the platform by the administrator is on a daily basis. The user makes registration on the book review management system platform. After successful registration, user reviews the catalog of books, their reviews, and finally adds his own review and comments on reviews. The frequency of using the platform by users is several times a month.

Description of use cases are provided in Table 19.1.

19.3.3 Use case diagram

The above story is depicted on use case diagram (see Figure 19.1).

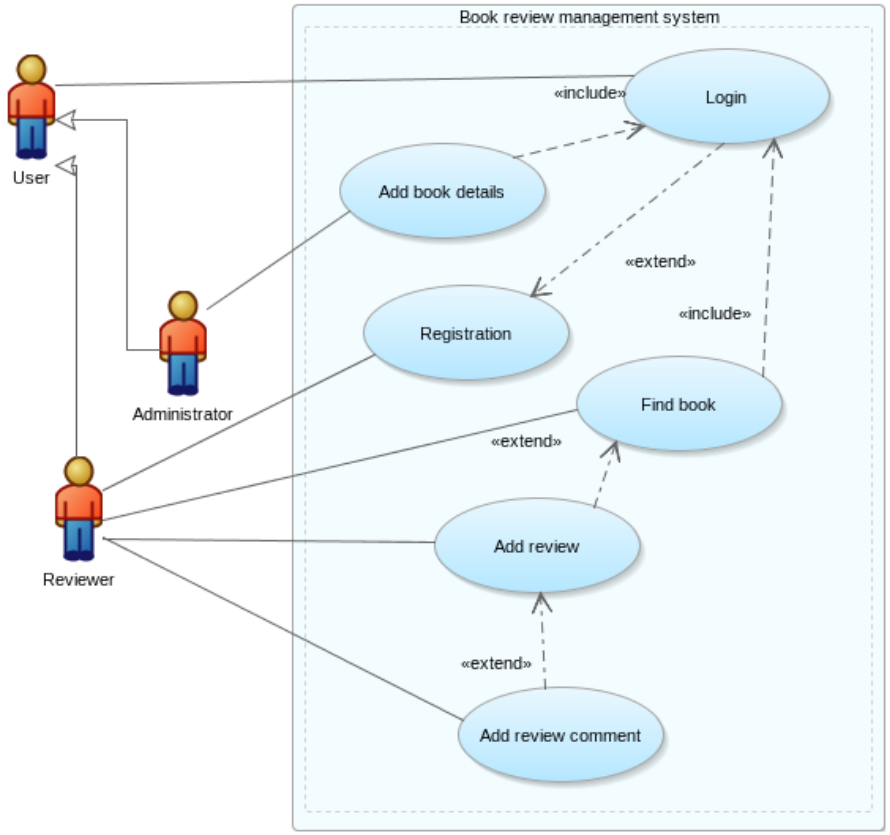


Figure 19.1: Use case diagrams.

Table 19.1: Use case description: book reviews management system

Keyword	**Value**
ID:	*ch19-10*
Title:	*Book reviews analysis*
Description:	*The book review management system is available based on registration on the platform. The platform allows browsing catalogs, filtering them, and classifying them according to categories.*
Primary Actor:	*User*
Preconditions:	browser on PC or smart phone, user has credentials, application is accesible.
Postconditions:	if book detail, review or comment on review is added than database store it.
Main	Scenarios
Success Scenario:	First scenario (administrator add new book): 1. the administrator logs into application. 2. the administrator selects the form for entering a new book. 3. the entry form includes title, authors, description, publisher, publish date and category selection. 4. the administrator completes the procedure by creating a new entry. 5. the administrator logs out of the platform. Second scenario (user, book review): 1. the user registers on the platform. 2. the user browses the book catalog and selects the book he wants to review. 3. the review form includes title, price, profile name, review helpfulness, review score, review summary, and review text. 4. the user completes the procedure by entering a new review. 5. the user logs out of the platform. Third scenario (user, commenting book review): 1. the user registers on the platform. 2. the user browses the book reviews. 3. the review includes title, price, profile name, review helpfulness, review score, review summary, and review text. 4. the user adds a new review comment. 5. the user logs out of the platform.
Extensions:	• *Book insert* • *Interactive report per book* • *Book reviews* • *Comments on reviews*
Frequency of Use:	*Daily basis, several time on month.*
Status:	[Development status]
Owner:	Potential users, based on registration.
Priority:	*high*

19.4 Data model

19.4.1 Narrative description of data model

Logical data model (Entity-Relationship model) consists of several entities: ch19_category, ch19_book_data, ch19_book_review, ch19_book_user in ch19_book_review_comment. Entity book_data consist of several attributes:, title, authors, about, publisher, publication_date and rating_count. Entity book_user include profile_name, firstname and lastname. Entity book_review have attributes: review and review_date and review_score. Entity review_comment contain responses on reviews: rev_comment, rev_helpfulness and rev_comm_score. Each book belong to one and only one category. Category can have more books. There may be more reviews related to one book. User can comment more books and more reviews by other users. We add primary UID called ID in each entity. These attributes will become primary keys in corresponding tables. After we generate SQL script for table creation we add two check constraints - we will allow review score and comment on review score to have numeric values between 1 and 5 only.

Developer has several possibilities to create tables. One is to use SQL Developer Data Modeler and design logical model from scratch, transform to relational and generate SQL script. Another is to create tables in APEX, export scripts which creates tables, apply reverse engineering in SQL Develoepr Data Modeler to generate relational model and than transform it to logical model. Using only APEX Quick SQL functionality is the most straight forward approach. After Quick SQL is writen, it can be transformed into SQL script with APEX built-in generator.

19.4.2 Logical data model

Logical data model is presented on Figure 19.2.

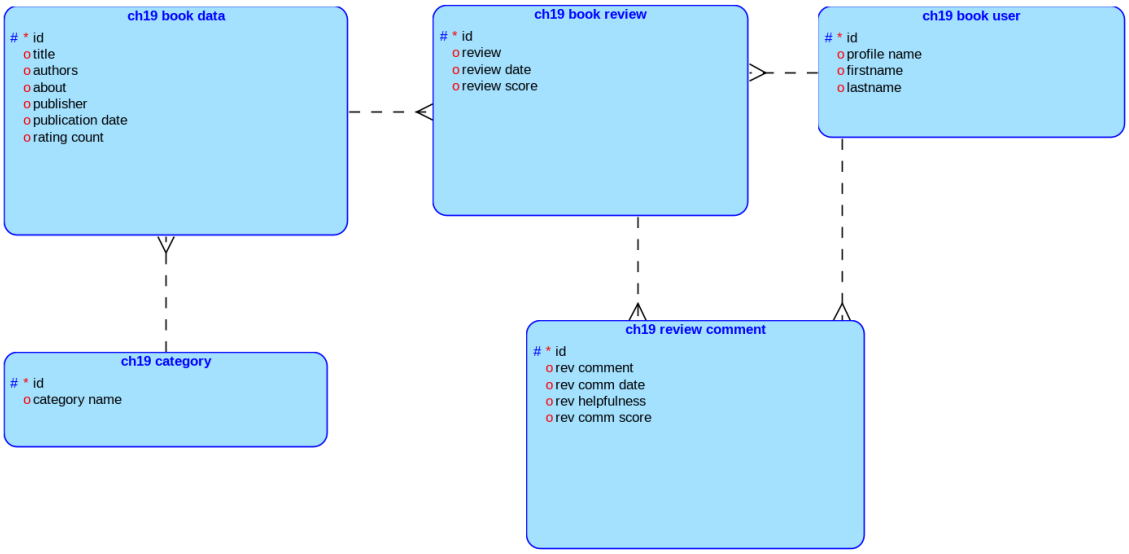


Figure 19.2: Logical data model.

19.4.3 Relational data model

Automatic transformation from second logical data model to relational data model in Oracle SQL Data Modeler is provided by function Engineer to Relational Model. The result is shown on Figure 19.3.

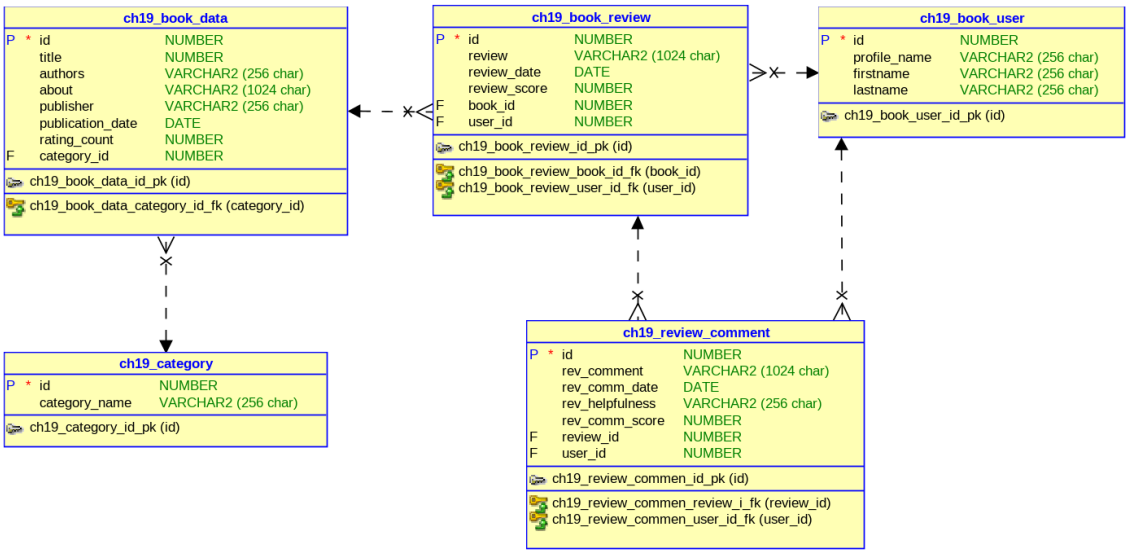


Figure 19.3: Relational data model.

Let now Oracle SQL Data Modeler generates SQL script. Select all tables on relational model and use function File > Export > DDL File to get the script. We can import the generated script in APEX and execute it.

19.4.4 SQL script

The script which corresponds to relational model includes the following SQL commands:

```
1  create table ch19_category (
2          id number generated by default on null as identity
3          constraint ch19_category_id_pk primary key,
4          category_name varchar2(256 char));
5
6  create table ch19_book_user (
7          id number generated by default on null as identity
8          constraint ch19_book_user_id_pk primary key,
9          profile_name varchar2(256 char),
10         firstname varchar2(256 char),
11         lastname varchar2(256 char));
12
13 create table ch19_book_data (
14         id number generated by default on null as identity
15                 constraint ch19_book_data_id_pk primary key,
16         title varchar2(256 char),
17         authors varchar2(256 char),
18         about varchar2(1024 char),
19         publisher varchar2(256 char),
20         publication_date date,
21         rating_count number,
22         category_id      number
23         constraint ch19_book_data_category_id_fk
24         references ch19_category on delete cascade);
25
```

```
create table ch19_book_review (
        id number generated by default on null as identity
        constraint ch19_book_review_id_pk primary key,
        review varchar2(1024 char),
        review_date date,
        review_score  number
                constraint ch19_book_revie_review_scor_ck
                check (review_score in (1,2,3,4,5)),
        book_id   number
        constraint ch19_book_review_book_id_fk
        references ch19_book_data on delete cascade,
        user_id   number
                constraint ch19_book_review_user_id_fk
references ch19_book_user on delete cascade);

create table ch19_review_comment (
        id  number generated by default on null as identity
                constraint ch19_review_commen_id_pk primary key,
        rev_comment varchar2(1024 char),
        rev_comm_date date,
        rev_helpfulness varchar2(256 char),
        rev_comm_score number
                constraint ch19_review_c_rev_comm_scor_ck
                check (rev_comm_score in (1,2,3,4,5)),
        review_id number
                constraint ch19_review_commen_review_i_fk
                references ch19_book_review on delete cascade,
        user_id   number
                constraint ch19_review_commen_user_id_fk
                references ch19_book_user on delete cascade);
```

19.4.5 Quick SQL

The most straight forward procedure to generate SQL script is usage of Quick SQL function in APEX. The following lines in Quick SQL enable generation of SQL script, presented above:

```
ch19_category
        category_name vc256

ch19_book_user
        profile_name vc256
        firstname vc256
        lastname vc256

ch19_book_data
        title vc256
        authors vc256
        about   vc1024
        publisher  vc256
        publication_date date
        rating_count num
        category_id /fk ch19_category
```

```
17
18 ch19_book_review
19         review vc1024
20         review_date date
21         review_score  num /check 1,2,3,4,5
22         book_id /fk ch19_book_data
23         user_id /fk ch19_book_user
24
25 ch19_review_comment
26         rev_comment vc1024
27         rev_comm_date date
28         rev_helpfulness vc256
29         rev_comm_score  num /check 1,2,3,4,5
30         review_id /fk ch19_book_review
31         user_id /fk ch19_book_user
```

The code is very dense. However it requires the basic knowledge of data modeling and syntax of Quick SQL.

19.5 Application interfaces

19.5.1 Administrator

In this case we will generate application from SQL script which creates tables and populates sample data. Find file CH19CREATEINSERT.sql in learning materials and import it into your workspace. Once imported open it and click Create App (see Figure 19.4).

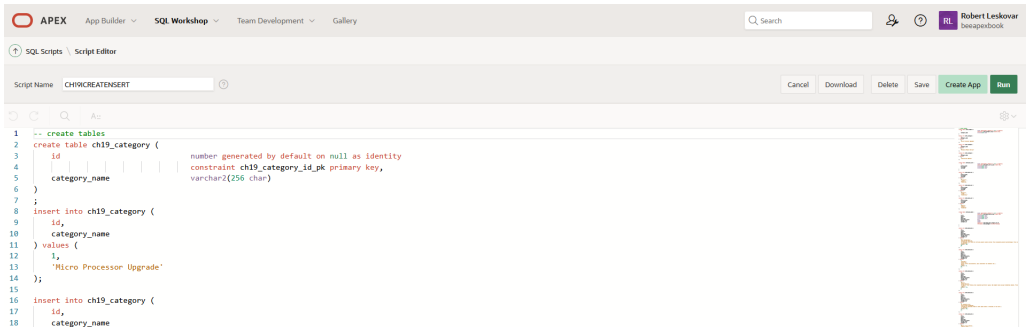


Figure 19.4: Generating application out of script CH19CREATEINSERT - part 1.

Name the application, select all Features and click Generate Application (see Figure 19.5).

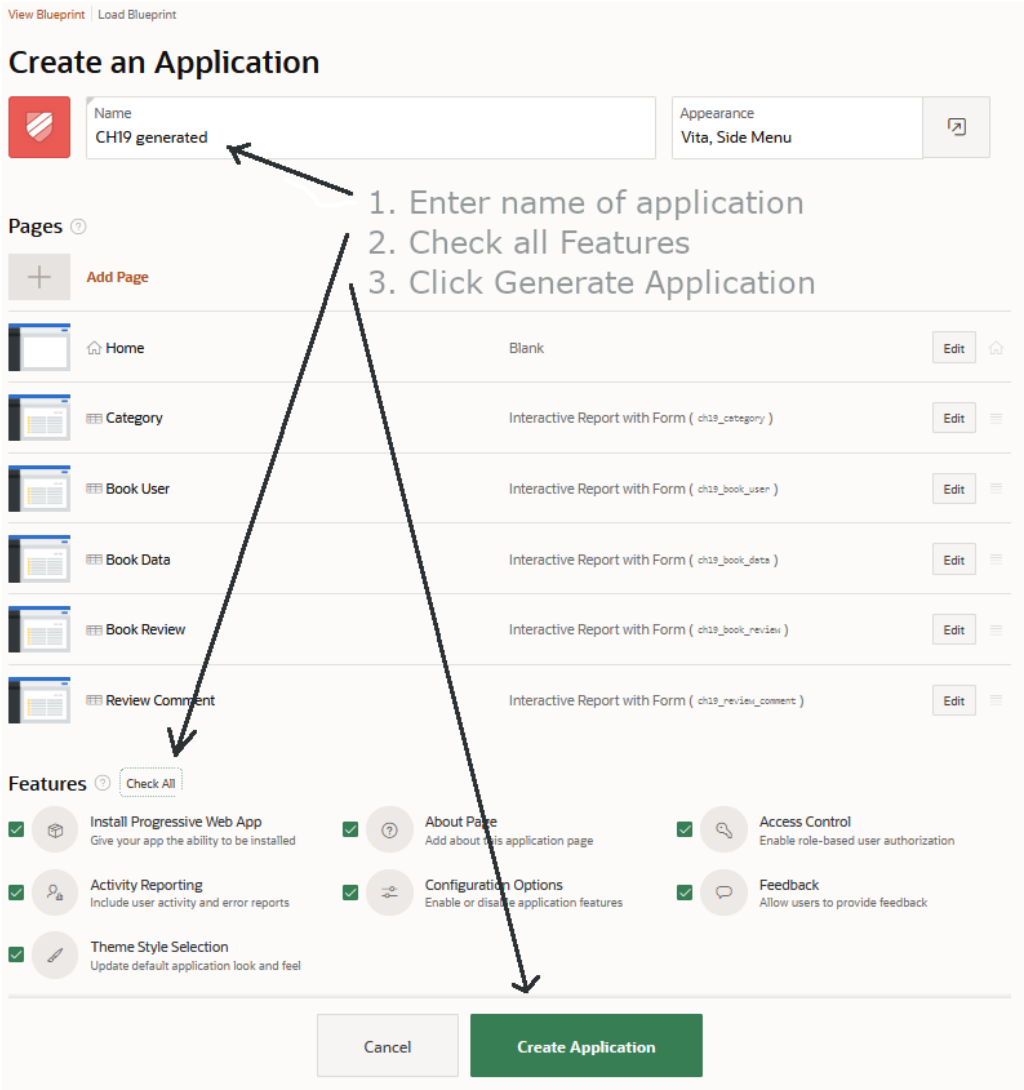


Figure 19.5: Generating application out of script CH19CREATEINSERT - part 2.

Working prototype of the application is now prepared. From now on functions for administrator and end user will be presented. So far not a single line of code was written by developer (with the exception of Quick SQL) and no application interface was tailored by developer. The actual code behind the application was assembled entirely by APEX wizards.

Figure 19.6 shows the form for entering a new book by the administrator.

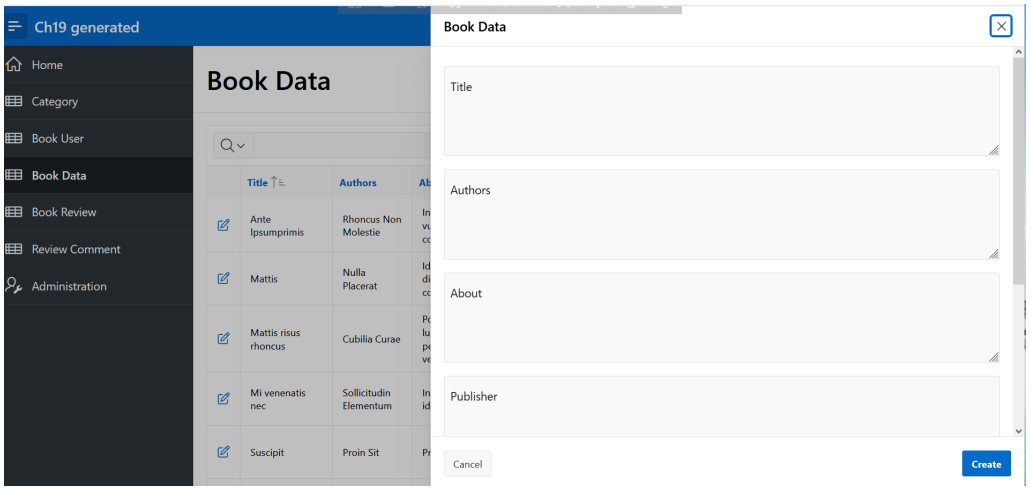


Figure 19.6: Adding book by administrator.

Figure 19.7 shows the form for inserting the book category by the administrator.

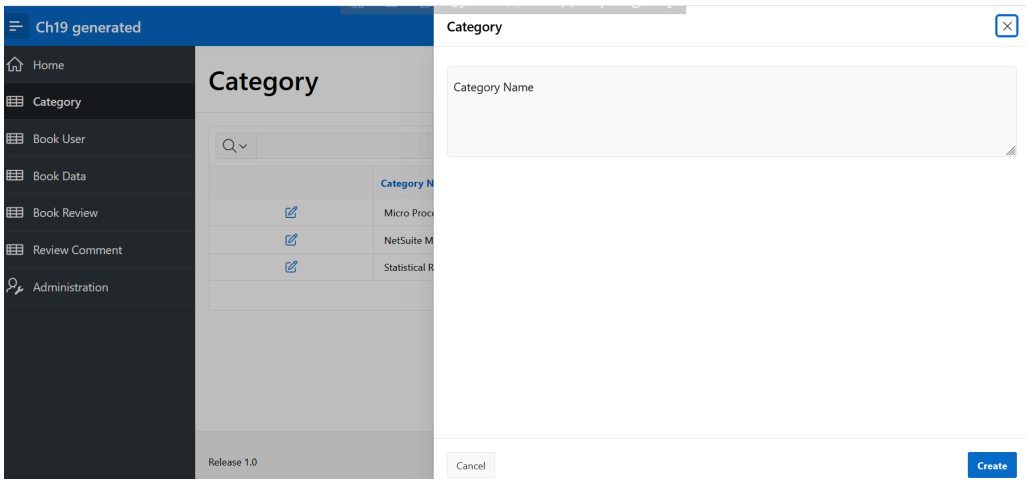


Figure 19.7: Adding category by administrator.

19.5.2 User

Figure 19.8 shows the user registration form.

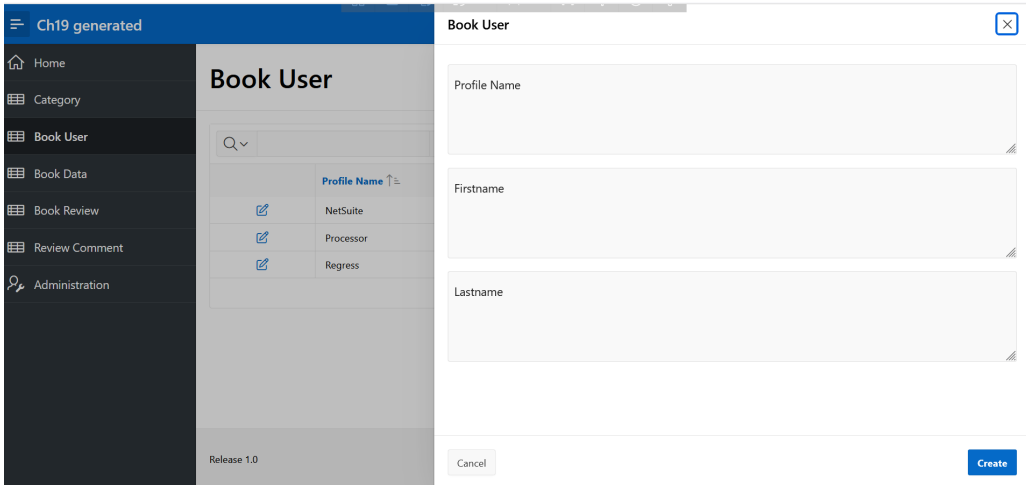


Figure 19.8: User registration.

Figure 19.9 shows the browsing and adding reviews.

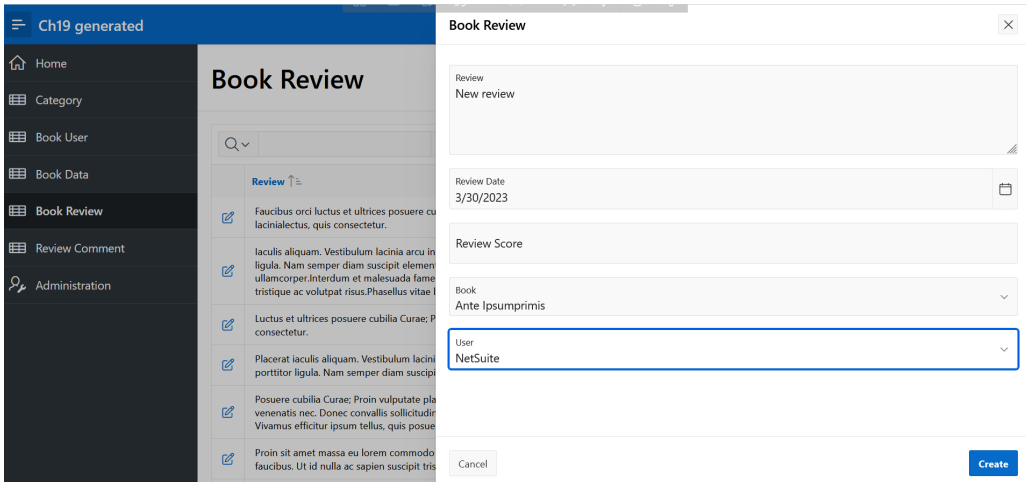


Figure 19.9: Browsing and adding reviews.

Figure 19.10 shows form to comment a review.

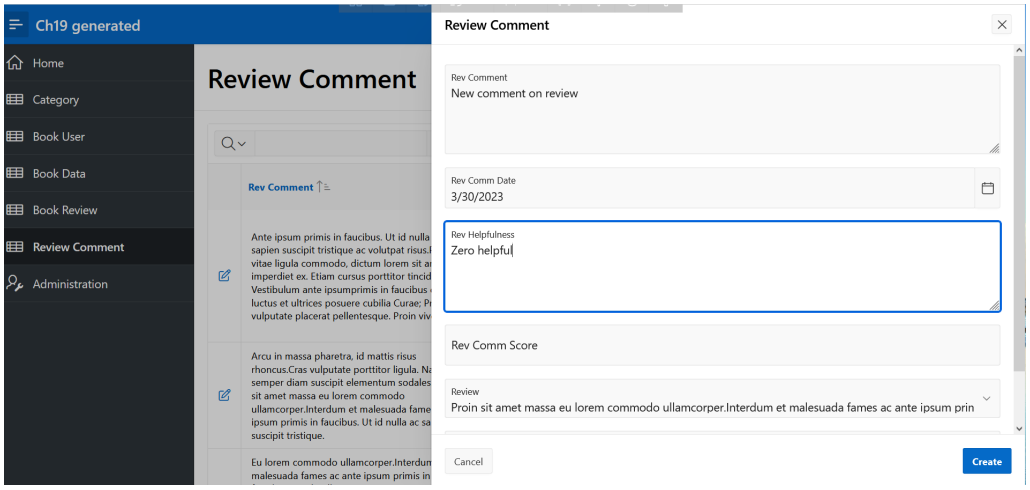


Figure 19.10: Form to comment a review

Reports are by default tabular. End user can generate graphs without programming, using just built in wizard. See Chapter 8 for transforming reports to graphs.

19.6 Define user roles

User roles can be defined in Application Access Control in Shared Components. At this time we will skip the details and implementation for this specific case. For fast overview see Chapter 13 or for thorough insight study APEX documentation on authorization.

19.7 Supplementary learning material

You can find the following supplementary learning material:

- script for creating and populating tables
- script for dropping tables
- exported packaged application
- video which demonstrates how to generate application out of script.

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter19 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

19.7.1 Exported application

Exported application is packaged. Installation creates tables as well it populates data. De-installation removes all data base objects used in this application.

Packaged application is tested and it will run in new workspace if the following requirements are meet:

- add APEX user before running application. Only in developement and testing workspace navigate to Shared Components > Application Access Control > Add User Role Assignment; enter APEX user and set this user roles Administrator, Contributer and Reader. In production consultation with skilled personel before deployment in a must.

If user is not granted appropriate role than imported application will crash. It is necessary to clear web browser cookie (i.e. Firefox: Settings > Cookies and Site Data > Manage Data) after application crashes due to unmet requirements.

19.7.2 Video guides

Video guide shows all steps in application development.

19.8 Questions

1. Why was in this case used Quick SQL script?
2. How can you generate multiple forms and reports in one shot?
3. How can you authorize APEX application user within application itself?

19.9 Answers

1. Quick SQL script was introduced to speed-up development of the data model. Quick SQL script generates SQL script upon request. SQL script is executable i.e. it creates database objects like tables and views.
2. It is possible to generate multiple forms and reports in one shot by writing the script which contain definition of several tables.
3. To authorize APEX application user within application itself, the application must be generated with Access Control Feature. Than authorized user (Administrator) can add other users with Administrator, Contributor and Reader roles.



20. Bill-of-material and cost calculation

R. LESKOVAR, U. RAJKOVIČ, A.BAGGIA

20.1 Business view of the case

The observed small company, which will be called **OSC** (10-20 employees) manufactures cable assemblies, cable bundles, conductors and signal lamps. They also offer assembly of electromechanical parts and mechanical semi-products. Their customers are bigger companies which provide house appliances, measuring equipment, control systems, electricity meters, medical devices and similar. Customer orders are accepted by phone and e-mail. For simple products and operations, as well as for repetitive orders, **OSC** send confirmation to the customer very quickly. Recently, orders for multi-level assembled products and more complex semi-products have been increasing. Also, there are dozens of items in one order and each item can include product composed od several semi-products and materials. Semi-products may be composed of semi-products and materials. The customers wants to receive an offer quickly. Delivering a calculation of the price of the material and semi-products (bill-of-material calculation or short BOM calculation) is demanding, error prune and time consuming task for multi-level assembled products and semi-products. It may take several days to respond to the customer with the offer. The calculation of costs of work (operations) is another calculation which for brevity of the chapter is omitted. The scope of the chapter will therefore be BOM calculation only. The company owner expressed the following needs:

- the report on products, semi-products, material and possibly customer inquiries
- the report on bill-of-material
- form for editing products, semi-products, material and possibly customer inquiries
- form for editing bill-of-material
- calculation and tree-like presentation of bill-of-material

An application should be usable on phone, tablet or desktop computer.

20.2 Problem definition

The company manager faces several issues related to agile response:

- calculation is time consuming, error prone and demanding, however it must be prepared very quickly,
- over-dues and calculation mistakes can affect business in several ways such as loosing potential customers, lower financial results, decreasing business reputation, lowering sales and stuffing and similar.

Table 20.1: Use case description: report and maintain basic data.

Keyword	**Value**
ID:	*Ch20-01*
Title:	*Report and maintain basic data*
Description:	*The manager uses the APEX application to report and maintain (select, update) basic data whether it is a product, semi-product, material or inquiry. Materials and aggregates are described with same set of attributes. Updating prices of materials will not trigger re-calculation of the prices of the aggregates in which material is used. Customer inquiry can be composed of several products and semi-products. The unit of measure for quantity in this case will be piece.*
Primary Actor:	*Manager*
Preconditions:	*Manager has account for APEX application.*
Postconditions:	*After reporting or maintaining data another action can be taken including BOM calculation.*
Main	*Scenarios*
Success Scenario:	1. *Open the web browser and sign-in to the application.* 2. *Select menu item or page navigation "Prod./Semi/Material".* 3. *To edit scroll down the page or filter data to find specific product, semi-product or material.* 4. *Click on pencil icon for specific product, semi-product or material and update data.* 5. *Click "Apply changes".*
Extensions:	*none*
Frequency of Use:	*average 5 per day*
Status:	*Finished*
Owner:	*Manager*
Priority:	*high*

The above risks should be mitigated effectively by web application.

20.3 Use cases

20.3.1 Narrative description

We can determine basic tasks to perform:

- report and maintain data about products, semi-products and materials. We treat consuming as reporting in tabular and tree forms and maintaining as adding, updating and deleting (see Table 20.1).
- calculate BOM for specific inquiry, product or semi-product (see Table 20.2).

The titles of use cases are therefore "report and maintain basic data", "report and maintain structure data" and "calculate BOM".

20.3.2 Semi-structured description

The above story is depicted on use case diagram.

20.3.3 Use case diagram

The above story is depicted on use case diagram (see Figure 20.1).

Table 20.2: Use case description: report and maintain structure data.

Keyword	**Value**
ID:	*Ch20-02*
Title:	*Report and maintain structure data*
Description:	*Manager uses APEX application to report (view selected data) and maintain (insert and delete) the structure of products, semi-products and material. Customer inquiry can be composed of several products and semi-products. Inserting new items or deleting existing items will not trigger (re)calculation of the prices of the aggregates.*
Primary Actor:	*Manager*
Preconditions:	*Manager has account for APEX instance.*
Postconditions:	*After reporting or maintaining data, another action can be taken including BOM calculation.*
Main	*Scenarios*
Success Scenario:	1. *Open the web browser and sign-in to the application.* 2. *Select menu item or page navigation "Bill-of-material".* 3. *To add a component click "Create" button.* 4. *To delete an item in the structure click "Delete".*
Extensions:	*none*
Frequency of Use:	*average 30 per month*
Status:	*Finished*
Owner:	*Manager*
Priority:	*high*

Table 20.3: Use case description: calculate BOM.

Keyword	**Value**
ID:	*Ch20-03*
Title:	*Calculate BOM*
Description:	*The manager uses the APEX application to calculate the price of the structure described by BOM for a specific item, whether it is a product, semi-product or inquiry. The resulting prices are updated for all items, aggregated in the structure, regardless of their complexity and depth, including the top aggregate.*
Primary Actor:	*Manager*
Preconditions:	*Manager has account for APEX instance.*
Postconditions:	*After calculation, any another action can be taken including another BOM calculation.*
Main	*Scenarios*
Success Scenario:	1. *Open the web browser and sign-in to the application.* 2. *Select menu item or page navigation "Tree view and calculation".* 3. *Select the aggregate to calculate.* 4. *Click "Calculate" button. Message appears showing the status of calculation and the calculated price.* 5. *Expand or collapse the tree structure of selected item to show sub parts data.*
Extensions:	*none*
Frequency of Use:	*average 30 per month*
Status:	*Finished*
Owner:	*Manager*
Priority:	*high*

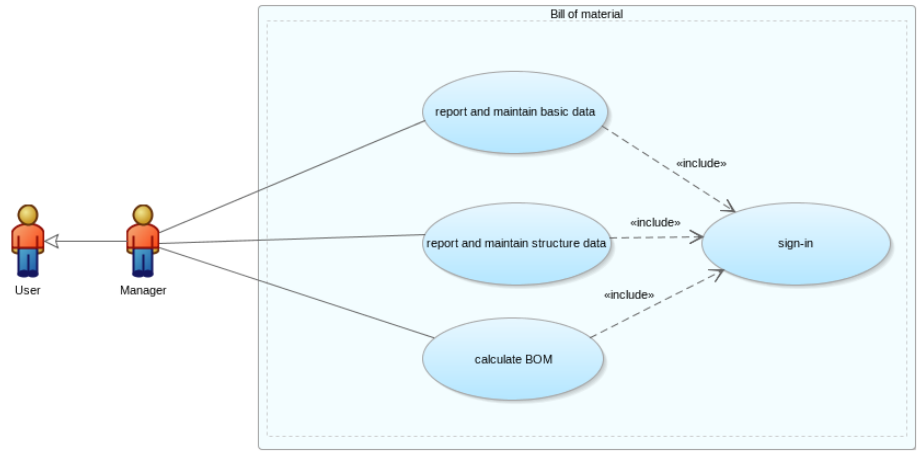


Figure 20.1: Use case diagram.

20.4 Data model

20.4.1 Narrative description of data model

There are only two entities in the data model for this case. Entity **ch20 psm** (later referred as a table) describes products, semi-products, materials and customer inquiries with only three attributes common to all: ID, description, unit of measure and price per unit of measure. Values for all attributes are required. Why? It is meaningless to store an instance without knowing how it is called, how it is measured and what is the price per unit. The ID is unique number without any decimal part. The Description must contain at least one printable character. The domain of units of measure is limited to millimeter, centimeter, meter and piece. Due to the small number of distinct values of units of measure no additional entity will be introduced to present the fact that one unit of measure has many occurrences in **ch20 psm** while one **ch20 psm** instance relates to only one instance of unit of measure only.

To present arbitrary hierarchical structure (bill of material of any width and any depth) the entity **ch20 bom** has three attributes only: identification of sub part, identification of aggregate and quantity of sub part contained in aggregate. Values for all attributes must be present. The pair (**identification of sub part, identification of aggregate**) is unique however we will add a primary key generated as a sequence of integer values to take benefits of low code APEX environment. One instance of **ch20 bom** can has double relation with **ch20 psm**: sub part relates to one instance of **ch20 psm** and aggregate relates to one instance of **ch20 psm**. In the opposite direction we have: one **ch20 psm** instance may point to many sub parts and many aggregates in **ch20 bom** entity. Therefore we have two 1-to-many relations between **ch20 psm** and **ch20 bom**. One more thing more about logical data model must be pointed: the uniqueness of pair **(identification of sub part, identification of aggregate)**. Logical model in Oracle Data modeler has no "syntax" to express this uniqueness as special object in data base called index. But on relational data model it is possible to introduce uniqueness of specified pair of values as index. In relational data model the unique index is composed of two fields in the table. This unique index guarantees that no duplicate pair values will be stored in data base - data base engine will protect developer and end user to disrupt data integrity.

20.4.2 Implementation of business rules in data base

Logical and relational data model is simple. In mathematics bill of material is special type of graph where each node is connected with only one parent node except the root node which has no parent node. It is very easy to break a hierarchy by introducing additional connection between nodes or

deleting a connection. In the context of data base changes in connections affects data integrity. Storing cyclical structure in **ch20 bom** would cause that calculation of bill of material never ends. In the context of business rules it is impossible and forbidden for a car (product) to be composed of wheels (semi-product) and at the same time that wheel to be composed of cars. Also car cannot be composed of itself. To prevent violation of business rules we will introduce a trigger (another data base object) mechanism which is usually represented by a few lines of code in PL/SQL language. The trigger will be executed before writing to or updating **ch20 bom**. Such code would prevent new insert into **ch20 bom** if pair in reverse order already exists or both new sub part and aggregate has the same value. Implementation of such kind of rules cannot be presented on logical or relational data model.

It is also impossible to present how calculation of BOM will be implemented in data models. This chapter will provide insight into code which turns hierarchy upside down (from bottom to top) and start summing total cost from the bottom - first calculate the cost of material in semi-products and them cost of semi-products in product. Remember that data model with two entities or two tables is capable to present any complexity (width and depth of hierarchy) of BOM structure.

So start with data models first.

20.4.3 Logical data model

Logical data model is presented on Figure 20.2.

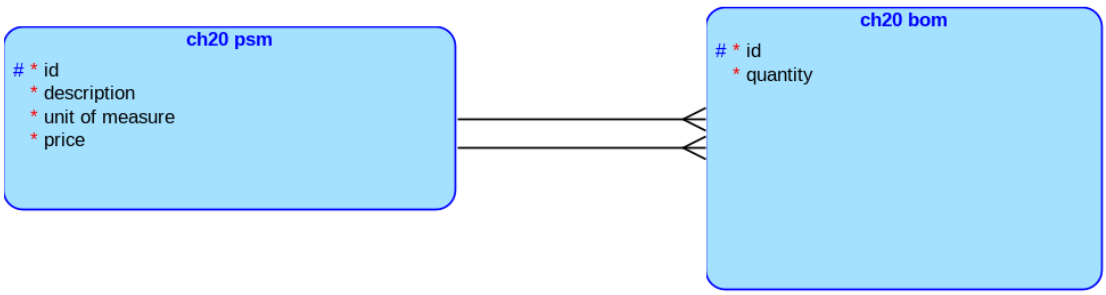


Figure 20.2: Logical data model.

20.4.4 Relational data model

Automatic transformation from logical data model to relational data model in Oracle SQL Data Modeler is provided by function *Engineering to relational*. Than we introduce unique index *ch20_bom_ui* composed of two fields *aggreg_id* and *subprt_id* (see Figure 20.3). The result, relational data model ready to be exported as SQL script is shown on Figure 20.4.

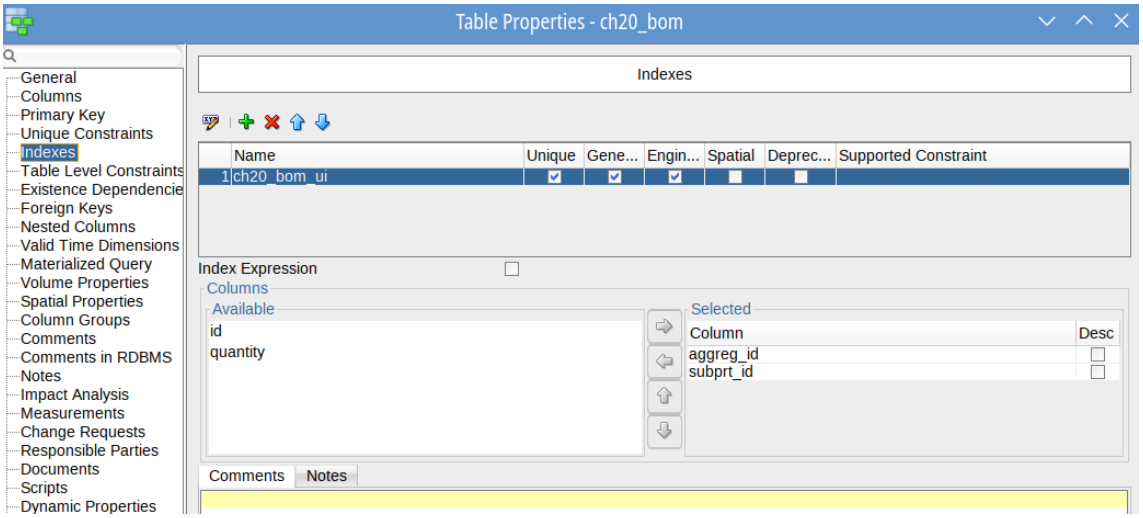


Figure 20.3: Definition of unique index in Oracle SQL Data Modeler.

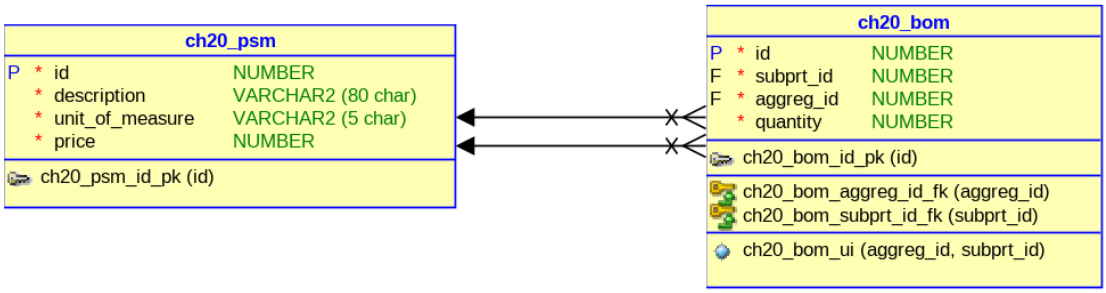


Figure 20.4: Relational data model.

Oracle SQL Data Modeler also generates SQL script for table, sequence and trigger creation. Select all tables on relational model and use function *File* > *Export* > *DDL File* to get a script. Save the script and check the order of table definitions. Table *ch20_psm* must be defined first and then *ch20_bom*. The following code define table *ch20_psm* and set constraints:

```
1          CREATE TABLE ch20_psm (
2                  id NUMBER GENERATED BY DEFAULT ON NULL
3                  AS IDENTITY ( START WITH 1 NOCACHE ) NOT NULL,
4                  description      VARCHAR2(80 CHAR) NOT NULL,
5                  unit_of_measure  VARCHAR2(5 CHAR) NOT NULL,
6                  price            NUMBER NOT NULL
7          ) LOGGING;
8
9          ALTER TABLE ch20_psm
10                 ADD CONSTRAINT ch20_psm_id_pk PRIMARY KEY ( id );
11
12         ALTER TABLE ch20_psm
13                 ADD CONSTRAINT ch20_psm_unit_of_measure_ck
14         CHECK ( unit_of_measure IN ( 'CM', 'M', 'MM', 'PIECE' ) );
```

The following code define table *ch20_bom*, set constraints and define unique index:

```
1          CREATE TABLE ch20_bom (
```

```
                id          NUMBER GENERATED BY DEFAULT ON NULL
                AS IDENTITY ( START WITH 1 NOCACHE ) NOT NULL,
                subprt_id NUMBER NOT NULL,
                aggreg_id NUMBER NOT NULL,
                quantity  NUMBER NOT NULL
        ) LOGGING;

        ALTER TABLE ch20_bom
                ADD CONSTRAINT ch20_bom_id_pk PRIMARY KEY ( id );

        ALTER TABLE ch20_bom ADD CONSTRAINT
        ch20_bom_subprt_id_fk FOREIGN KEY ( subprt_id )
        REFERENCES ch20_psm ( id )
                ON DELETE CASCADE NOT DEFERRABLE;

        ALTER TABLE ch20_bom ADD CONSTRAINT
        ch20_bom_aggreg_id_fk FOREIGN KEY ( aggreg_id )
        REFERENCES ch20_psm ( id )
                ON DELETE CASCADE NOT DEFERRABLE;

        CREATE UNIQUE INDEX ch20_bom_ui
                ON ch20_bom (aggreg_id, subprt_id );
```

Save script with the instructions which creates tables with primary and foreign keys, check constraints and unique index. Now it is time to generate tables in Oracle database. We can import the generated script and execute it in APEX.

20.4.5 Objects in APEX

Navigate your browser to your APEX workspace. In this section we will:

- create tables and indexes
- create function and procedure
- create trigger

There are two options to create tables and index. If you used Oracle Data Modeler to generate script first imported it (SQL Workshop > SQL Scripts > Upload) and then run it. Another option is to generate the above script by Quick SQL in the case you want to use only APEX to generate the such script. Navigate to SQL Workshop > Utilities > Quick SQL. Put the following text in Quick SQL text area (left part of the window):

```
        ch20_psm
          description vc80 /nn,
          unit_of_measure vc4  /check 'mm','cm','m', 'piece' /nn,
          price num /nn

        ch20_bom /unique
          subprt_id num  /fk ch20_psm /nn,
          aggreg_id num  /fk ch20_psm /nn,
          quantity num /nn
```

Click generate SQL. Tables have the same properties as the tables generated in Oracle Data Modeler (see Figure 20.5).

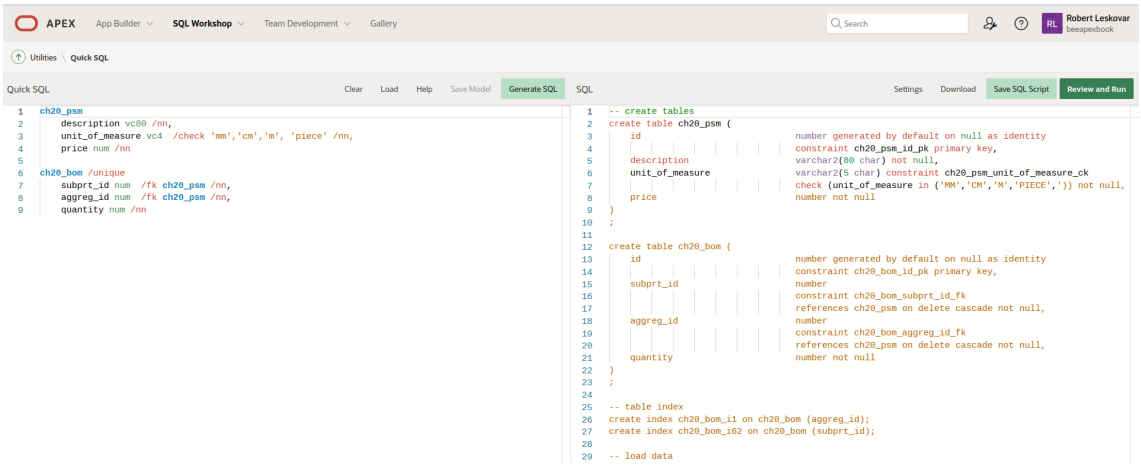


Figure 20.5: Generating SQL script by using Quick SQL tool.

Take a look at the lines 26 and 27. These two lines, although correct will be edited later. Click Save SQL Script and Review and Run. Replace lines 26 and 27 (create two indexes) and replace with one statement:

```
create unique index ch20_bom_ui on ch20_bom (aggreg_id, subprt_id);
```

Now run the script. Two tables and one unique index will be generated. Note that running this script will not be successful if you already run script generated with Oracle Data Modeler. The reason is that data base objects already exist.

To create function enter the following text in SQL Workshop > SQL Commands:

```
1  create or replace FUNCTION  CH20_GET_PRICE (
2    xpsm IN   NUMBER
3    ) RETURN NUMBER
4  IS
5    y_price number;
6  BEGIN
7    SELECT price INTO y_price FROM ch20_psm  WHERE ID = xpsm;
8    return y_price;
9  EXCEPTION
10   WHEN OTHERS THEN
11     return(-505);
12 END CH20_GET_PRICE;
```

This function will return the price (product, semi-product, material, inquiry) for given ID. If something unexpected happens with the data base (lost connection, server down, hardware error, no data for specified ID) then negative value -505 will be returned. Select all text and run. You will receive feedback that function is created.

To create procedure to calculate costs of bill of material enter the following text in SQL Workshop > SQL Commands:

```
1  create or replace PROCEDURE  CH20_CALCULATE_BOM
2    (xpsm IN NUMBER,
3    ystatus OUT INTEGER,
4    yprice OUT NUMBER)
5  -- creation date: 20.1.2023
6  -- author: Robert Leskovar
7  -- input parameter: xpsm, id of product or semi-product
```

```
-- output parameters 1: status of calculation:
                0=not calculated, 1=succesfully calculated
-- output parameters 2: price calculated
-- description: calculation of the total costs of given product
--                              and all underlaying levels
-- exceptions: system and user defined
-- date of testing: 20.1.2023
-- author of test: swqlab
-- status: APPROVED
is
  v_rezult   NUMBER(38, 2);
  v_psm        NUMBER;
  v_price    NUMBER(38, 2);
  e_negative_price EXCEPTION;
  CURSOR c_ds IS
   WITH STRUK(hier, subprt_id, aggreg_id, quantity) AS
    (SELECT 1 AS hier, subprt_id, aggreg_id, quantity
     FROM CH20_BOM JOIN CH20_PSM ON subprt_id = CH20_PSM.ID
        WHERE aggreg_id = xpsm
   UNION ALL
    SELECT hier+1, C1.subprt_id, C1.aggreg_id, C1.quantity
     FROM CH20_BOM C1 JOIN CH20_PSM P ON C1.subprt_id = P.ID
      JOIN STRUK C2 ON C1.aggreg_id = C2.subprt_id)
   SELECT hier, subprt_id, aggreg_id, quantity
    FROM STRUK ORDER BY hier DESC, aggreg_id, subprt_id;
BEGIN
  ystatus := 0;
  v_rezult := 0.0;
  v_psm := 0;
  SAVEPOINT old_state;
  FOR r_ds IN c_ds LOOP
   v_price:= CH20_GET_PRICE(r_ds.subprt_id);
   IF ( v_price <= 0 ) THEN
    RAISE e_negative_price;
   END IF;
   IF ( r_ds.aggreg_id = v_psm ) THEN
    v_rezult := v_rezult + v_price * r_ds.quantity;
    UPDATE ch20_psm SET price = v_rezult WHERE ID = r_ds.aggreg_id;
   ELSE
    v_psm := r_ds.aggreg_id;
    v_rezult := v_price * r_ds.quantity;
    UPDATE ch20_psm SET price = v_rezult WHERE ID = r_ds.aggreg_id;
   END IF;
  END LOOP;
  COMMIT;
  ystatus := 1;
  yprice := v_rezult;
EXCEPTION
  WHEN e_negative_price THEN
   ROLLBACK TO old_state;
```

```
58    ystatus := 0;
59 yprice := -505;  WHEN OTHERS THEN
60    ROLLBACK TO old_state;
61    ystatus := 0;
62    yprice := -100;
63 end CH20_CALCULATE_BOM;
```

This procedure will return the status of calculation and the price (product, semi-product, material, inquiry) for given ID unless any component has negative value. If something unexpected happens with the data base (lost connection, server down, hardware error, no data for specified ID) then negative value will be returned. If the reader is not familiar with PL/SQL it would be enough to read comments in procedure. Otherwise rather complex cursor is created with common table expression (WITH statement). This cursor returns union of two sets. It provides complete structure of aggregate which is ordered from bottom to top. The prices of intermediate aggregates are stored in corresponding rows and cumulated till all rows in cursor are processed.

The last object to be created is trigger. The trigger will ensure that no aggregate and sub part in bill of material cannot have the same value (car is composed of car) and traversal relation is created (if car is composed of wheels than wheel cannot be composed of cars). To create the trigger which implements stated rules enter the following text in SQL Workshop > SQL Commands:

```
1  create or replace trigger CH20_TRG_BOM_RULES
2    before insert or update on ch20_bom
3    for each row
4
5    declare
6     xsubprt numeric :=:new.subprt_id;
7     xaggreg numeric :=:new.aggreg_id;
8     xcount numeric;
9    begin
10    select count(subprt_id) into xcount from ch20_bom
11     where subprt_id = xaggreg and aggreg_id = xsubprt;
12    if xsubprt = xaggreg or xcount > 0 then
13     :new.subprt_id :=null;
14    end if;
15 end;
```

With this trigger we will prevent inserting by forcing NULL value in a table field which must have value. The precedence of NOT NULL definition in the table column over the trigger mechanism will cause that no data inserted if this business rule is violated. However we protected the hierarchy to become cycle in a graph (term from a graph theory). If we skip the creation of this trigger we should train end user not to fall into cycle trap. It is much better to prevent than to heal, right?

20.5 Application interfaces

In the following figures, application interfaces are presented. First we present the sketch of entry window. User will navigate to managing basic data, managing structure data and calculation either with left side menu or with page navigation (see Figure 20.6).

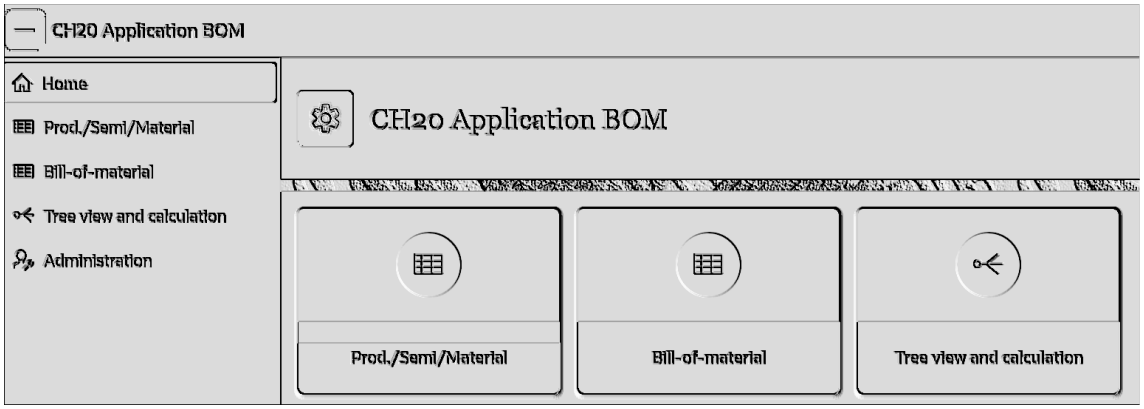


Figure 20.6: Application home page.

Next two figures (20.7 and 20.8) present the sketches of report and form for products, semi-products, materials and inquiries. Inserting, updating and deleting is available.

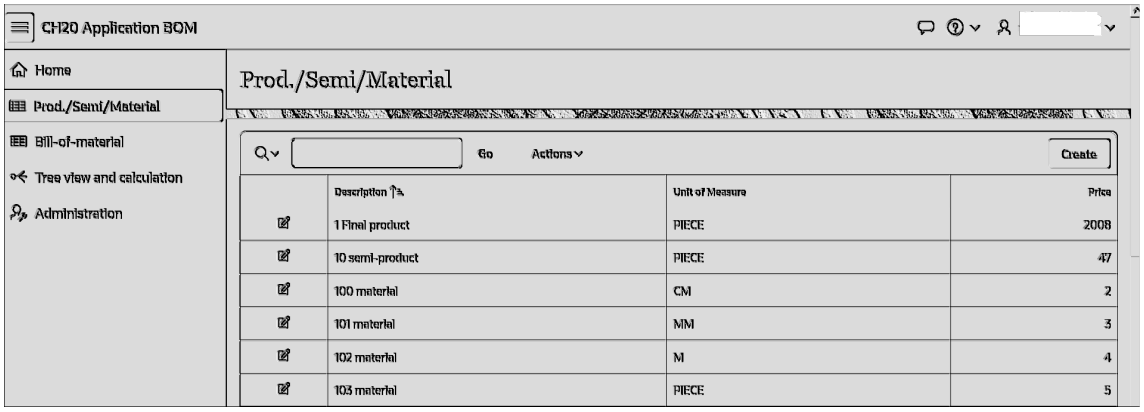


Figure 20.7: Managing basic data - report.

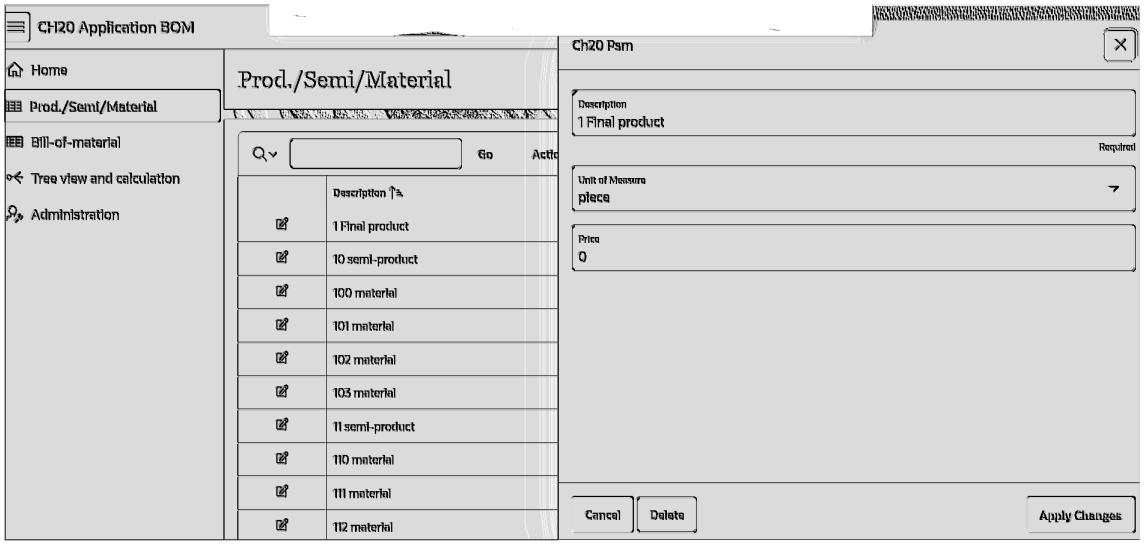


Figure 20.8: Managing basic data - form.

Further two figures (20.9 and 20.10) present the sketches of report and form for structure, bill

of material. Inserting, updating and deleting is available.



Figure 20.9: Managing structure data - report.

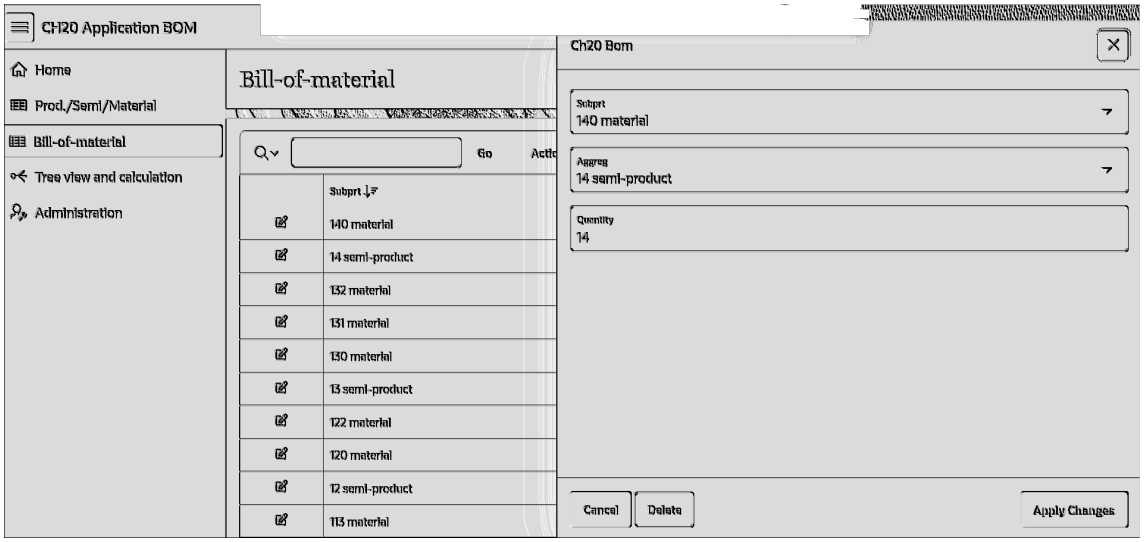


Figure 20.10: Managing structure data - form.

Sketch (20.11) presents the interface to perform a calculation of any aggregate (semi-product, product, inquiry). User first selects an item and then presses Calculate button. The status of calculation and calculated price is displayed as a pop-up and tree component is shown. User can expand and collapse hierarchy.

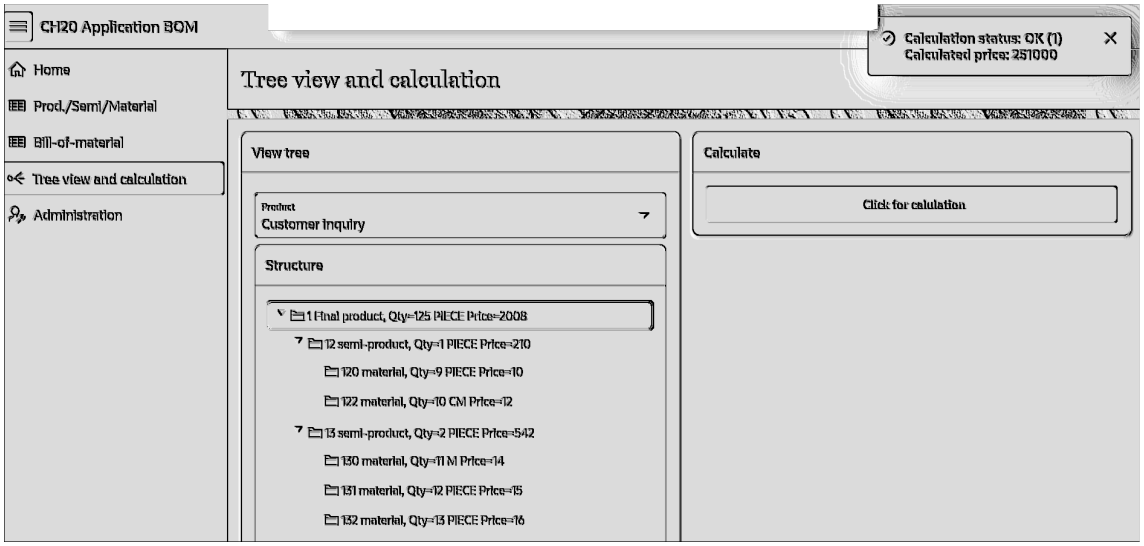


Figure 20.11: Page for calculation of bill of material.

20.6 Supplementary learning material

You can find the following supplementary learning material:

- exported application
- video guides

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter20 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

20.6.1 Exported applications

Exported application is packaged. Installation create tables, index, function, procedure and trigger as well it populate data. De-installation removes all data base objects used in this application.

20.6.2 Video guides

Video guide show every step in application development.

20.7 Questions

1. How would you change logical data model to implement new entity for units of measure, because we want to include tens of units?
2. How would specify Quick SQL to reflect the above change?
3. What will be the consequences of dropping unique index ch20_bom_ui?

20.8 Answers

1. We apply new entity *ch20_uom* and set at least two attributes (ID and description of the unit of measure). Than we set 1-to-many relation between *ch20_psm* and *ch20_uom*.
2. In Quick SQL we first define table *ch20_uom* as:

```
ch20_uom
  description vc80
```

and change the definition *ch20_psm* as:

```
ch20_psm
  description vc80 /nn,
  unit_of_measure num /fk ch20_uom
  price num
```

3. Dropping unique index ch20_bom_ui would enable to store multiple pairs (sub part, aggregate) with the same values. Table would consume more disk space. The calculation functionality would not be affected, however processing would take longer and more RAM would be consumed.



21. Nutrition and diet management

A. ANGEIOPLASTIS, G. MYLLIS, A. TSIMPIRIS, D. VARSAMIS

21.1 Business view of the case

Our example is Greece's most popular food media brand, with a reach of over 500,000 users. The company is the proud owner of Greece's market-leading food magazine and digital edition, the country's number one food website, an expanding number of live events, including the largest food event in Greece - the Greek Food Show - and a hugely successful series of cookbooks.

The company has decided to create and promote a web tool that allows users to access thousands of recipes via their website. In addition, users can personalize their favorite recipes by adding or removing ingredients, and the service will provide the nutrition facts of the new recipes, allowing users to create their own personalized cookbook of favorite recipes.

21.2 Problem definition

Users who enjoy cooking often face limitations when it comes to what they can eat, whether due to health issues such as diabetes or high cholesterol, or personal dietary preferences such as vegan or gluten-free. One issue with existing web-based recipe resources is that they often do not provide comprehensive alternatives to accommodate these limitations. As a result, users may become frustrated with the limited information provided and abandon the recipe search altogether, resulting in poor engagement.

21.3 Use cases

21.3.1 Narrative description

We will create an Oracle APEX application that allows users to view and add various healthy recipes. The application will provide a detailed description of each recipe, including a list of materials and the nutritional information of each ingredient. Users can also add their own recipes and corresponding ingredients using a form. The nutritional values of each recipe, such as the number of calories, total protein, number of portions, and preparation time, will be presented in diagrams for easy readability rather than a simple table of data.

21.3.2 Semi-structured description

The semi-structured description is provided in Table 21.1.

Table 21.1: Use case description: access to recipes, ingredients and nutrition facts calculation

Keyword	**Value**
ID:	*Ch21-01*
Title:	*The platform offers access to a variety of pre-made recipes and ingredients, as well as the option to add or remove ingredients to customize the recipes to individual preferences.Users can create a new personalized recipe by selecting their preferred ingredients, and the service will calculate the nutrition facts for the recipe.*
Description:	*Using the Apex application, users can browse and select recipes, and also configure recipe data such as creating, updating, and deleting ingredients. Both recipes and ingredients are defined by a set of attributes. Users can create a new recipe by adding or removing ingredients and calculate the updated nutrition facts. The results are displayed in diagrams that are updated in real-time based on the changes made to the recipe.*
Primary Actor:	*User*
Preconditions:	User has an account for APEX instance.
Post conditions:	After updating and choose recipe user can calculate ingredients facts. After calculating the nutrition facts for a recipe, users can repeat the process for other recipes by adding or removing ingredients to create new personalized recipes.
Main	Scenario
Success Scenario:	1. Open your web browser and sign in to the application. 2. Select either the **recipes** or **ingredients** option from the menu or page navigation. 3. To add a recipe, click the **add a recipe** button. To add an ingredient, click the **Add Ingredient** button. 4. To edit a recipe or an ingredient click on pencil symbol to the specific entry. 5. After making any changes, click the **Apply Changes** button to save them. To delete data, click on the **Delete** button. 6. To check the nutrition facts for the recipe, go to the menu or page navigation and select **Recipe total nutrition facts**. 7. The nutrition details provided include the portion size of the food that can be prepared based on specific quantities of the ingredients, the total protein and calorie count of the recipe, and the total time needed to prepare the recipe.
Extensions:	*None*
Frequency of Use:	*The users of the website publish on average 5 recipes per day, or approximately 1 thousand recipes per year. There are approximately 10 thousand users daily who browse only.*
Status:	Finished
Owner:	User
Priority:	*moderate*

21.3.3 Use case diagram

The above story is depicted on use case diagram (see Figure 21.1).

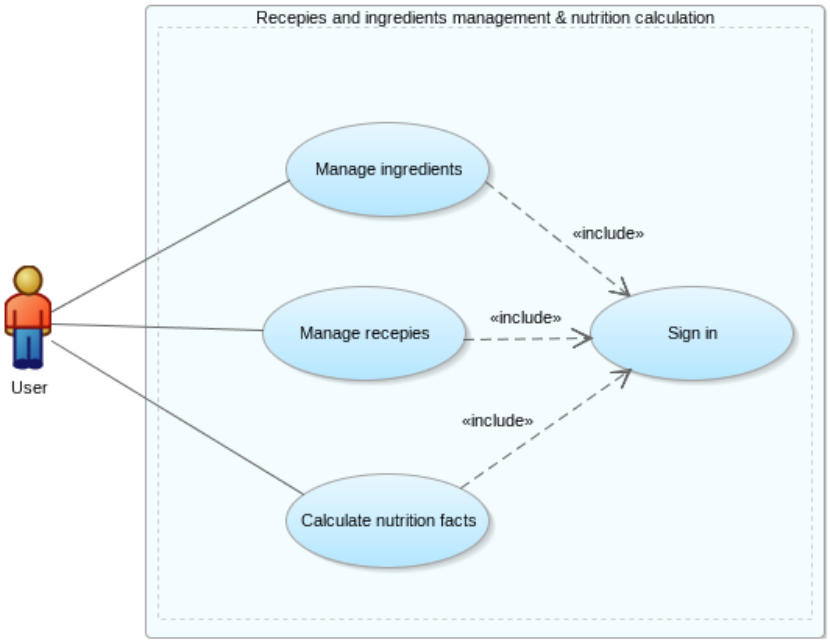


Figure 21.1: Use case diagram.

21.4 Data model

Data model presents three entities: recipes, ingredients and ingredients in recipes.

21.4.1 Logical data model

The logical data model is shown on Figure 21.2.

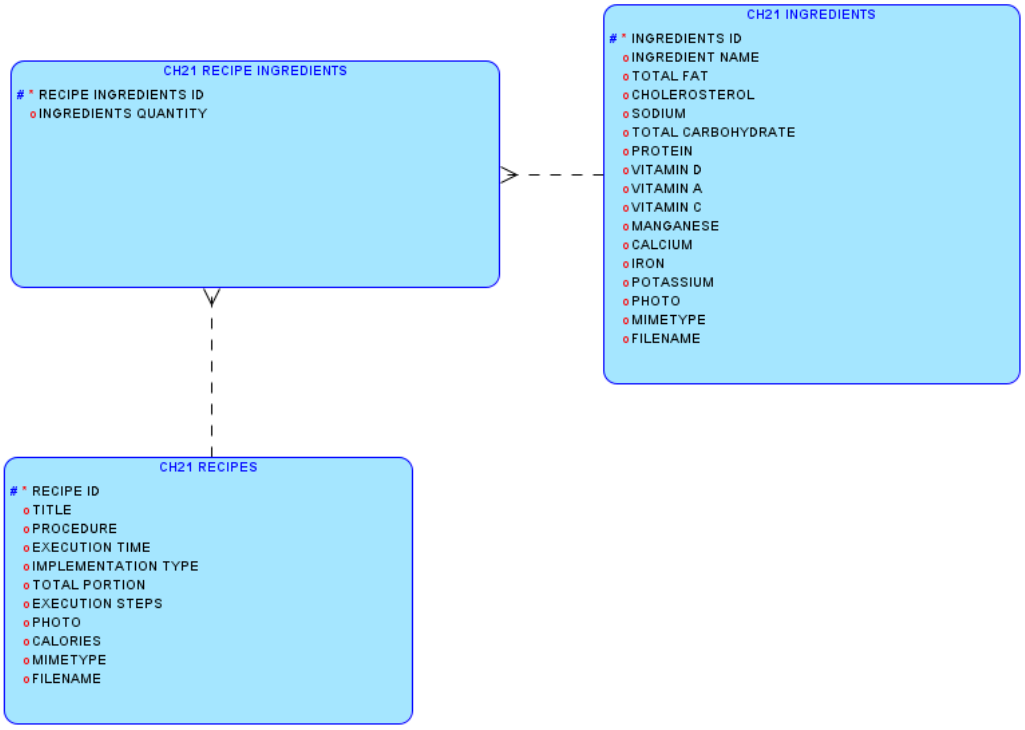


Figure 21.2: Logical data model.

21.4.2 Relational data model

The Relational data model is shown on Figure 21.3.

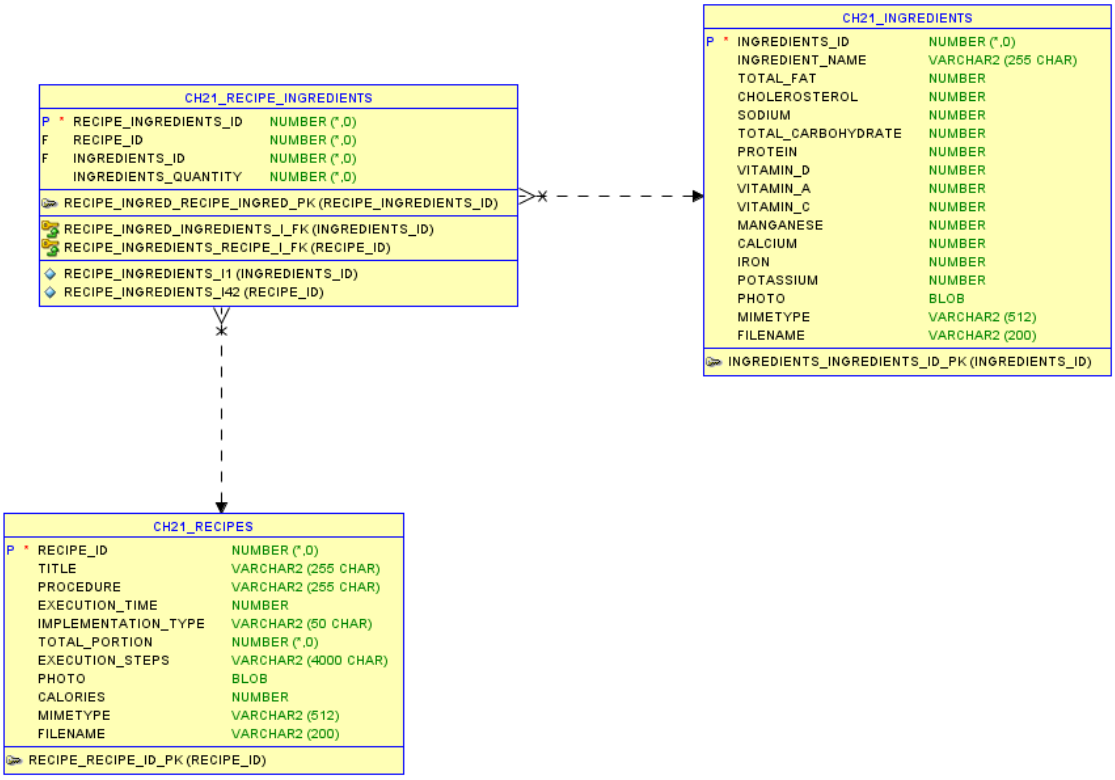


Figure 21.3: Relational data model.

21.4.3 QuickSQL and Script SQL Script

The provided code appears to be a set of instructions for generating sample data using the Quick SQL tool.The tables we need to create for our application are: CH21_RECIPES,CH21_RECIPE_INGREDIENTS, and CH21_INGREDIENTS. The fields for each table should be structured with as follows:

```
CH21_RECIPES
    RECIPE_ID int/pk,
    TITLE VC255,
    PROCEDURE VC255,
    EXECUTION_TIME NUM,
    IMPLEMENTATION_TYPE VC50,
    TOTAL_PORTION int,
    EXECUTION_STEPS  VC,
    PHOTO BLOB

CH21_INGREDIENTS
    INGREDIENTS_ID int/pk,
    INGREDIENT_NAME VC255,
    TOTAL_FAT NUM,
    CHOLEROSTEROL NUM,
    SODIUM NUM,
    TOTAL_CARBOHYDRATE NUM,
    PROTEIN NUM,
    VITAMIN_D NUM,
    VITAMIN_A NUM,
    VITAMIN_C NUM,
```

```
    MANGANESE NUM,
    CALCIUM NUM,
    IRON NUM,
    POTASSIUM NUM,
    PHOTO BLOB,
    MIMETYPE VC512,
    FILENAME VC200

CH21_RECIPE_INGREDIENTS
    RECIPE_INGREDIENTS_ID INT/PK,
    RECIPE_ID INT/FK RECIPE,
    INGREDIENTS_ID INT/FK INGREDIENTS,
    INGREDIENTS_QUANTITY INT
```

To run Quick SQL code, we must first access the **SQL Workshop** and select **SQL Scripts**. From there, we can choose **Quick SQL** to create tables. The process is simple; we only need to specify the table name and its fields, as well as any child tables related to our table. Figure 21.4 illustrates how to quickly create tables using this feature.

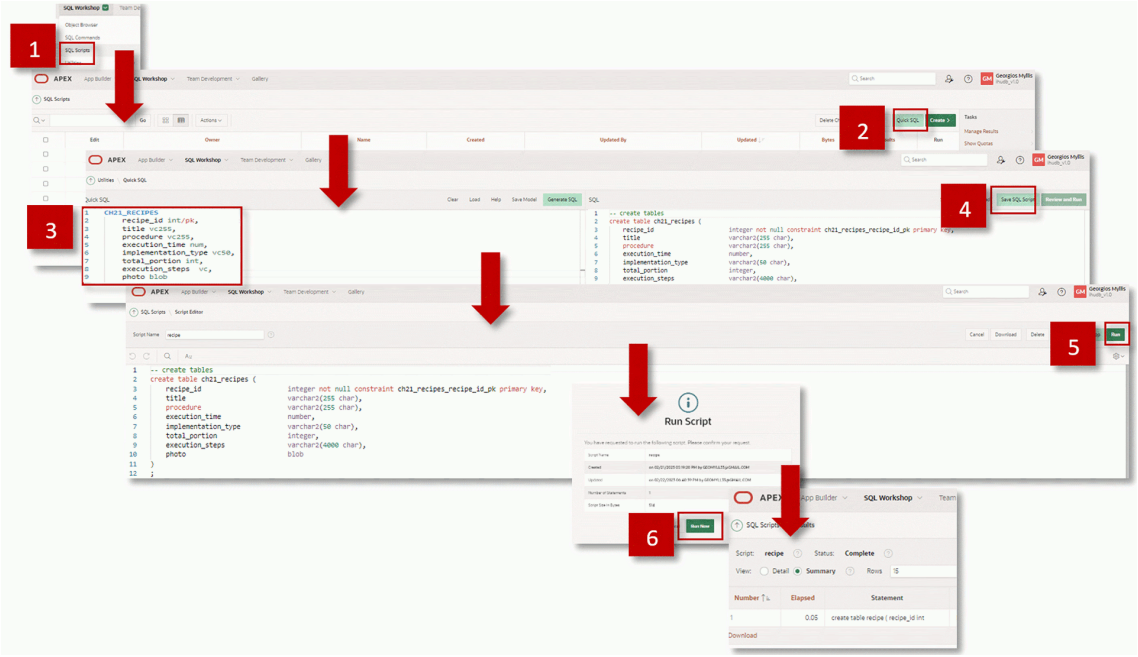


Figure 21.4: Quick SQL tool to create SQL script.

Our next step for the proper functioning of our tables is to ensure that each addition to the table is unique and that the Primary Key of each table is automatically incremented. To achieve this, we need to create three SEQUENCES.

Increase Recipe ID

```
CREATE SEQUENCE seq_RECIPE
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;
```

```
CREATE OR REPLACE TRIGGER RECIPE_trg
BEFORE INSERT
ON RECIPE
FOR EACH ROW
BEGIN
:new.RECIPE_ID:=seq_RECIPE.nextval;
END;
```

Increase Ingredient ID

```
CREATE SEQUENCE seq_INGREDIENTS
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;

CREATE OR REPLACE TRIGGER INGREDIENTS_trg
BEFORE INSERT
ON INGREDIENTS
FOR EACH ROW
BEGIN
:new.INGREDIENTS_ID:=seq_INGREDIENTS.nextval;
END;
```

Increase Recipe Ingredient ID

```
CREATE SEQUENCE seq_RECIPE_INGREDIENTS
START WITH 31
INCREMENT BY 1
NOCACHE
NOCYCLE;

CREATE OR REPLACE TRIGGER RECIPE_INGREDIENTS_trg
BEFORE INSERT
ON RECIPE_INGREDIENTS
FOR EACH ROW
BEGIN
:new.RECIPE_INGREDIENTS_ID:=seq_RECIPE_INGREDIENTS.nextval;
END;
```

To run the SEQUENCE, we need to navigate to the **SQL Workshop** and click on **SQL Scripts**. Then, we can click on the **Create** button and copy and paste the code for each SEQUENCE separately. After giving a name to the script, we can click on the **Run** button to execute the script. This ensures that each record in the tables is unique and eliminates the need to manually keep track of the last record added. Everything will be done automatically. An example of how to run the SEQUENCE in the APEX environment is shown on Figure 21.5.

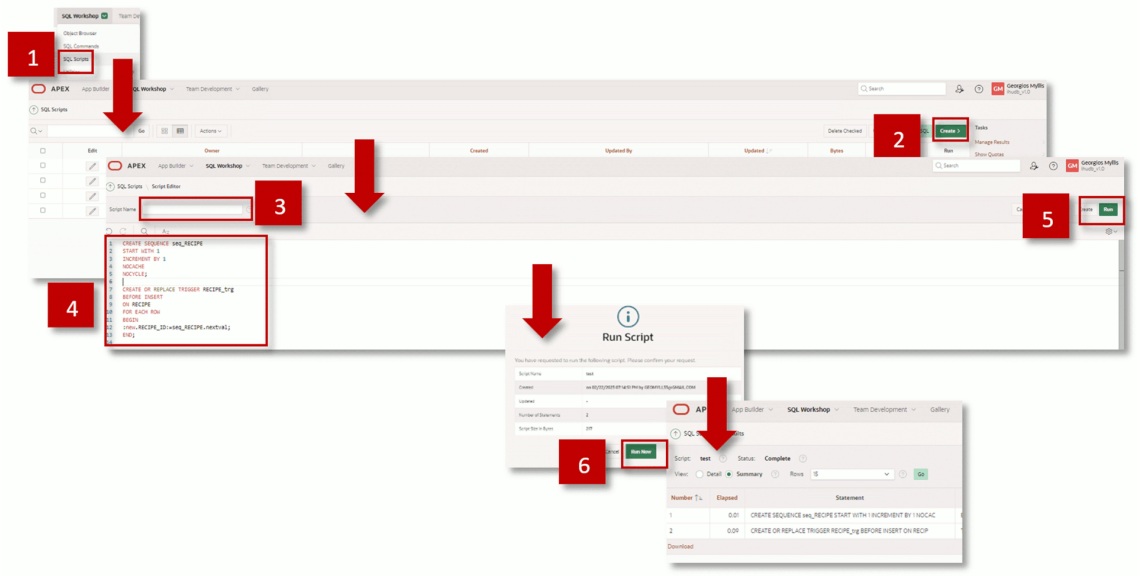


Figure 21.5: SQL script procedure to build SEQUENCE.

21.5 Application interfaces

In order to create the application as on Figure 21.6, click on **App Builder** click on **Create**. Then a new window will appear in which we must give the name of our application and select the Features we want for our application to have. The tables we will use for our application is: RECIPE, RECIPE INGREDIENTS, INGREDIENTS as explained earlier.

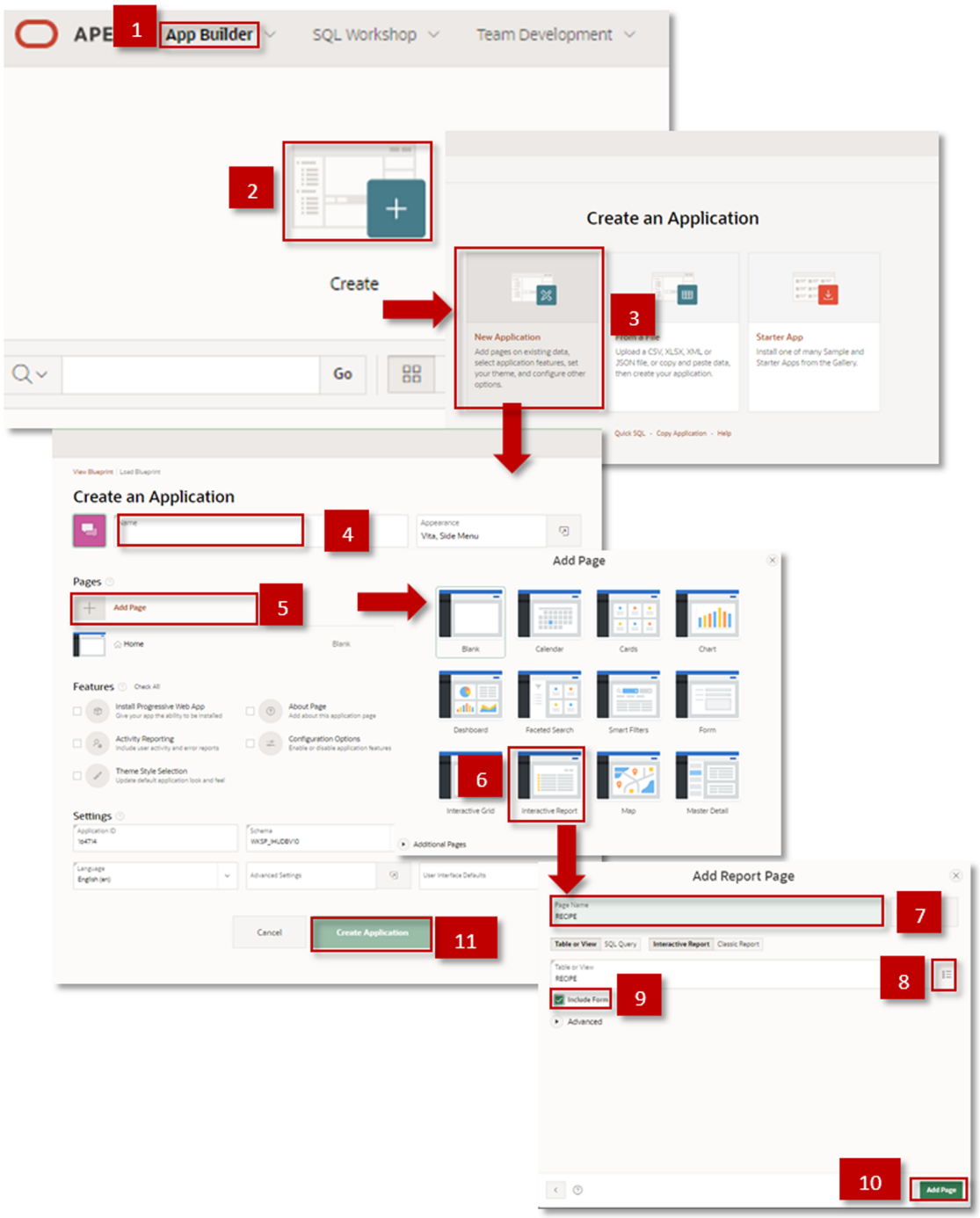


Figure 21.6: Create new application.

On Figure 21.7, a new App Builder environment has been created for your new application. To access your app, click on it and provide your administrator credentials, which should be the same as those for the workspace. Once you sign in, you can navigate your app using the mega menu located at the top.



Figure 21.7: Nutrition application.

Figure 21.8 is the recipe page, users can access all the necessary information for cooking the recipe, such as the procedure, execution time, total portions, execution steps, and a photo of the desired outcome. Depending on their permissions, users can also add a new recipe or update an existing one to suit their preferences.

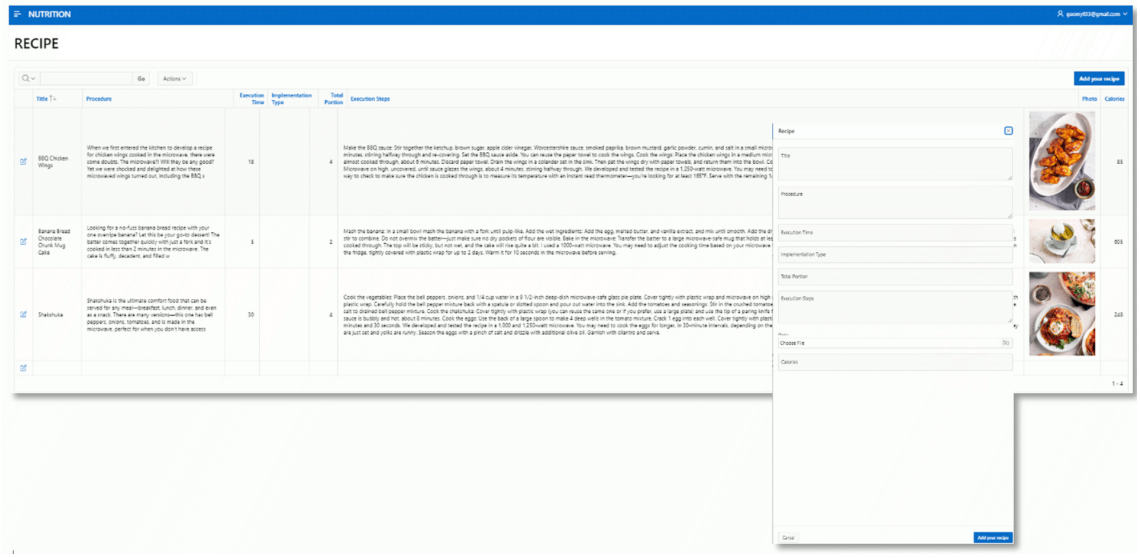


Figure 21.8: Nutrition application- recipe page.

On Figure 21.9, the ingredients page displays various attributes of the ingredient such as total fat, cholesterol, sodium, total carbohydrate, protein, etc. These attributes are used to calculate the nutrition facts of the recipes. Users with appropriate rights can maintain the page by adding new ingredients or updating existing ones as needed.

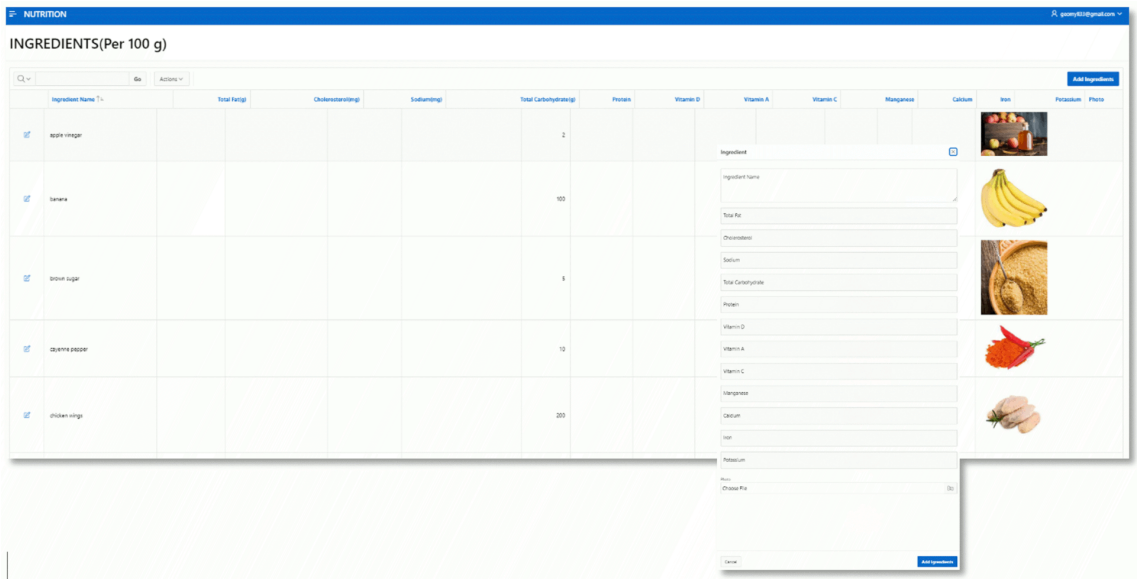


Figure 21.9: Nutrition application- ingredients page.

On Figure 21.10, the recipe ingredients page provides users with the benefit of this web service, allowing them to create recipes according to their needs or modify existing ones and then check the nutrition facts. Users with the appropriate permissions on this page can add or remove ingredients from the recipes.

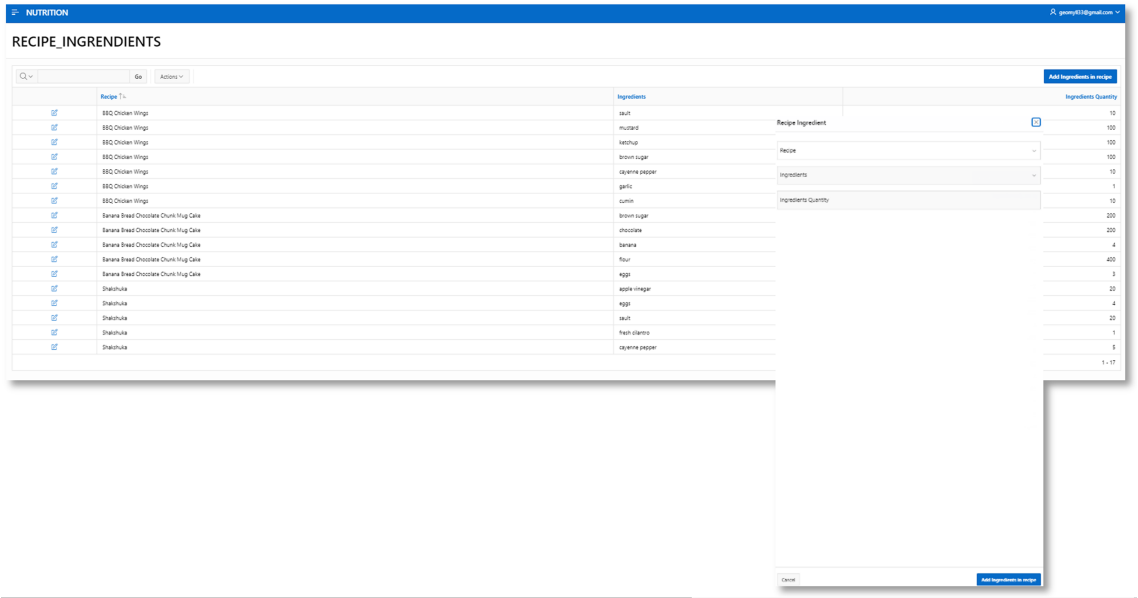


Figure 21.10: Nutrition application- recipe ingredients page.

On Figure 21.11, the nutrition facts reports page provides the user with all the necessary information concerning the recipe they built. Information such as: the total protein of the recipe, total calories of the recipe, the number of portions the recipe serves, and how long it takes to cook the recipe are all provided.

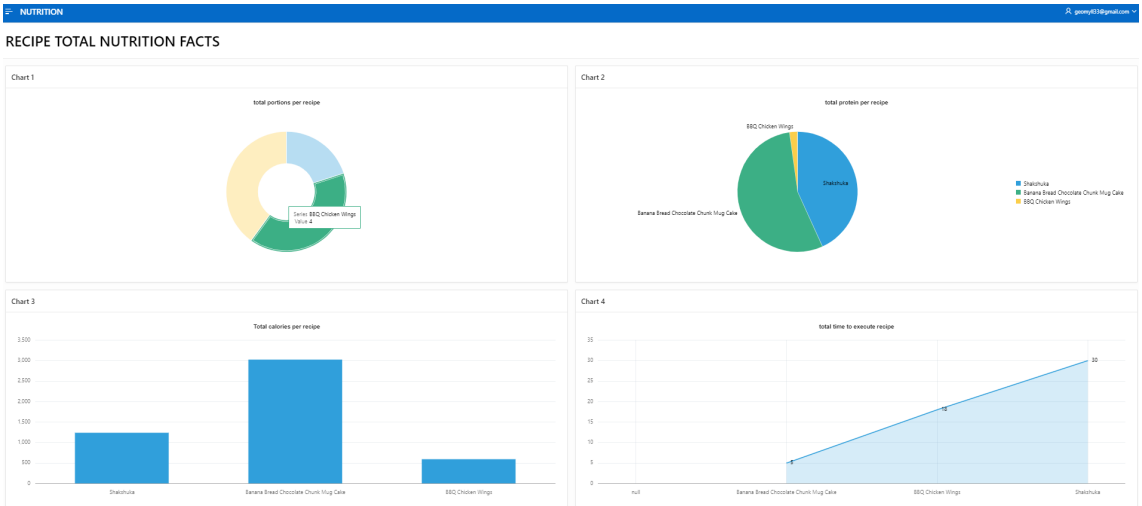


Figure 21.11: Nutrition application- nutrition facts page.

21.5.1 How to build a dashboard

To create a dashboard page for your application, follow the steps shown on Figure 21.12.

- Click on your application environment.
- Click on the **Create Page** button.
- Select **Dashboard** from the list of available page types.
- Click on the **Next**. button.
- Choose your preferred dashboard layout.
- Click on the **Create Page** button to create the dashboard page.

These steps will guide you through the process of creating a dashboard page for your application.

Once created, you can customize the dashboard by adding widgets and configuring the layout to display the data you need to monitor and manage your application.

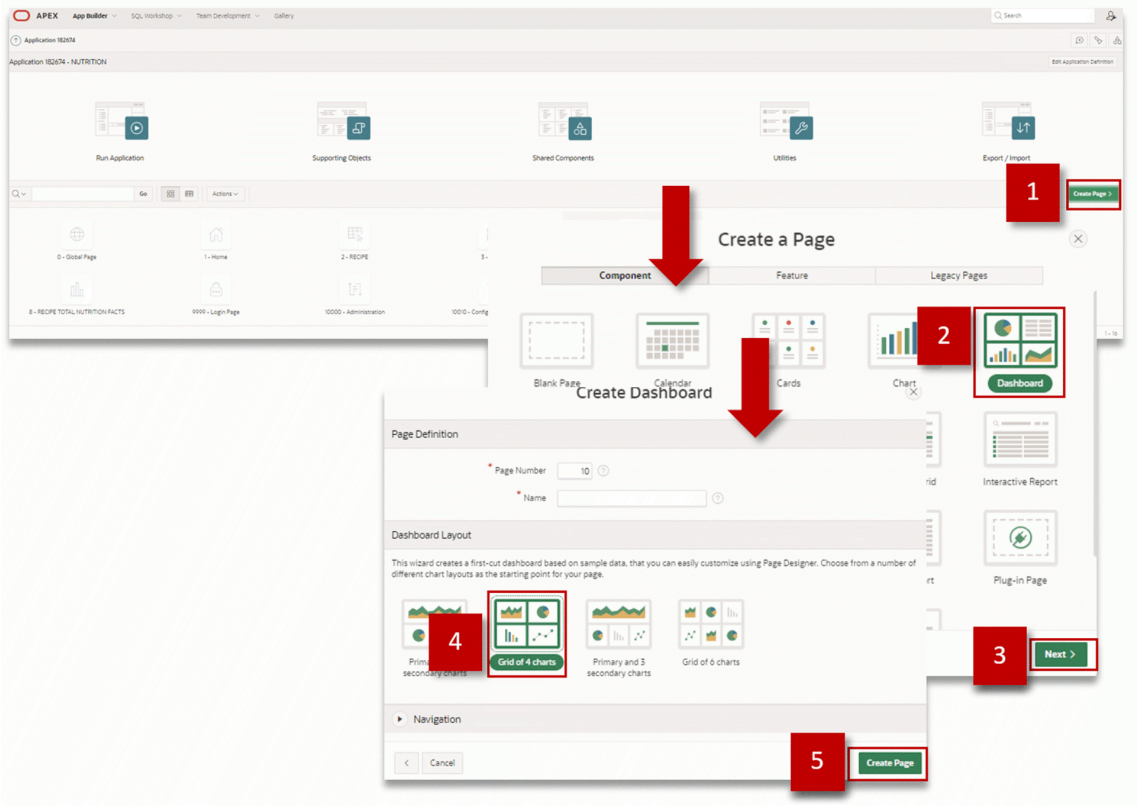


Figure 21.12: Create a dashboard page for your application.

Now, navigate to your dashboard report, which can be found in your application environment, and enter the page designer. As shown on Figure 21.12, we have selected four charts and will add different data to each using SQL queries.

To begin, select Chart 1 and rename it to “Total Portion per Recipe”. Then, select **Attribute** and choose the Donuts Chart type. Next, select Chart 2 and rename it to “Total Protein per Recipe”, which will display the recipes with the most protein. For this chart, select the **Pie Chart** type.

Moving on to Chart 3, rename it to “Total Calories per Recipe”. This chart will display the total number of calories for each recipe and we will use the **Bar Chart type**. Finally, rename Chart 4 to “Total Time to Execute Recipe”. Select **Attributes** and then choose the **Line with Area** chart type (see Figure 21.13).

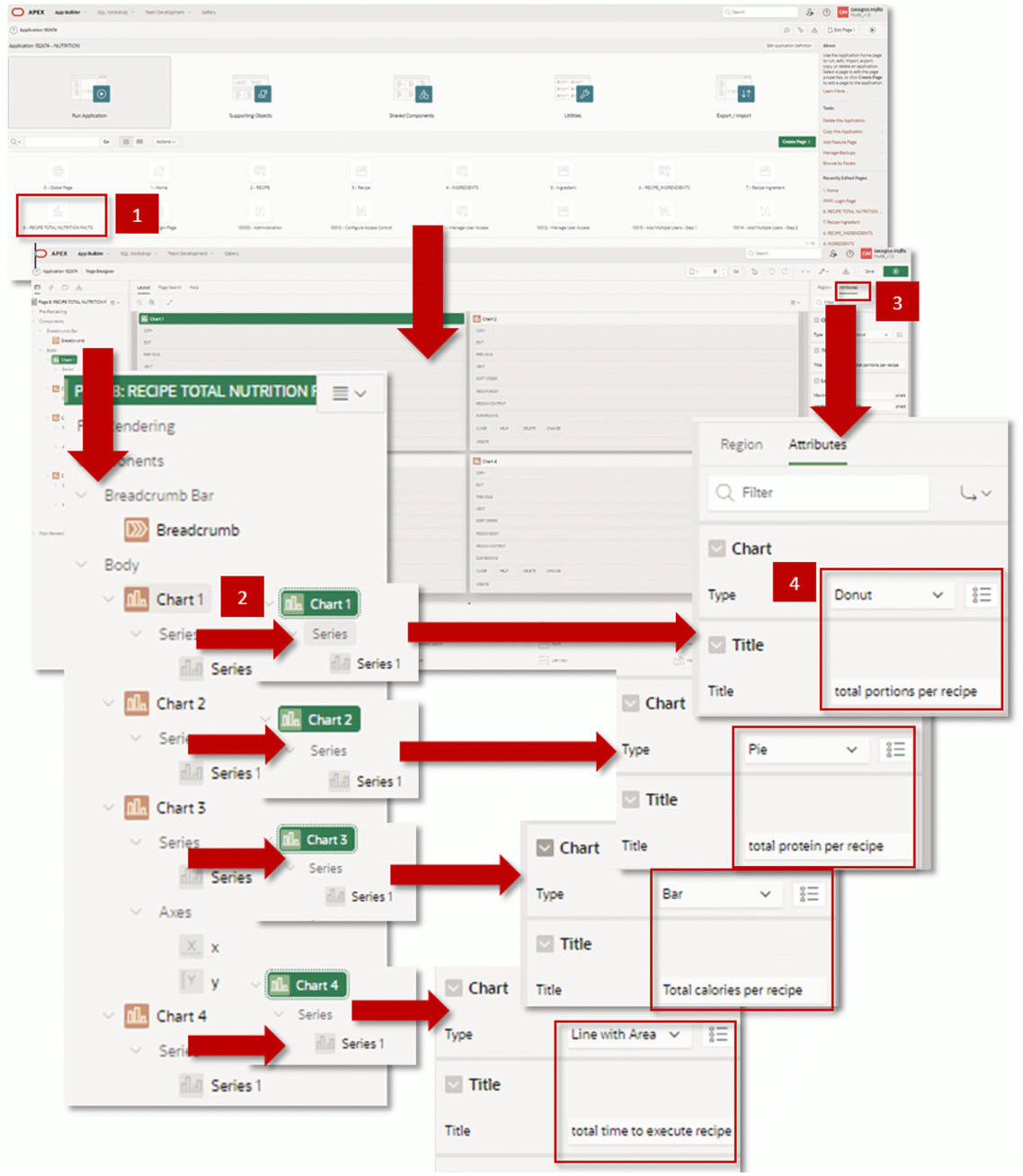


Figure 21.13: Configure Title and chat type on Dashboard reports.

Then click on **Series 1** of chart 1 and in Source put the following settings:

```
SELECT DISTINCT RECIPE.TITLE,
RECIPE.TOTAL PORTION
FROM RECIPE
```

On Series 1 of chart 2 under Source put the following settings

```
SELECT DISTINCT RECIPE.TITLE, SUM(protein) AS protein
FROM RECIPE
INNER JOIN RECIPE_INGREDIENTS ON
RECIPE.RECIPE_ID=RECIPE_INGREDIENTS.RECIPE_ID
```

```
INNER JOIN INGREDIENTS ON
INGREDIENTS.INGREDIENTS_ID=RECIPE_INGREDIENTS.INGREDIENTS_ID
GROUP BY RECIPE.TITLE
```

On Series 1 of chart 3 under Source put the following settings:

```
select distinct recipe.title, sum(CALORIES) as CALORIES
from recipe
inner join recipe_ingredients on
recipe.recipe_id=recipe_ingredients.recipe_id
inner join ingredients on
ingredients.ingredients_id=recipe_ingredients.ingredients_id
group by recipe.title
```

and on Series 1 of chart 4 under Source put the following settings:

```
SELECT DISTINCT RECIPE.TITLE,RECIPE.EXECUTION_TIME
FROM RECIPE
```

Figure 21.14 provide some guidelines how to configure it.

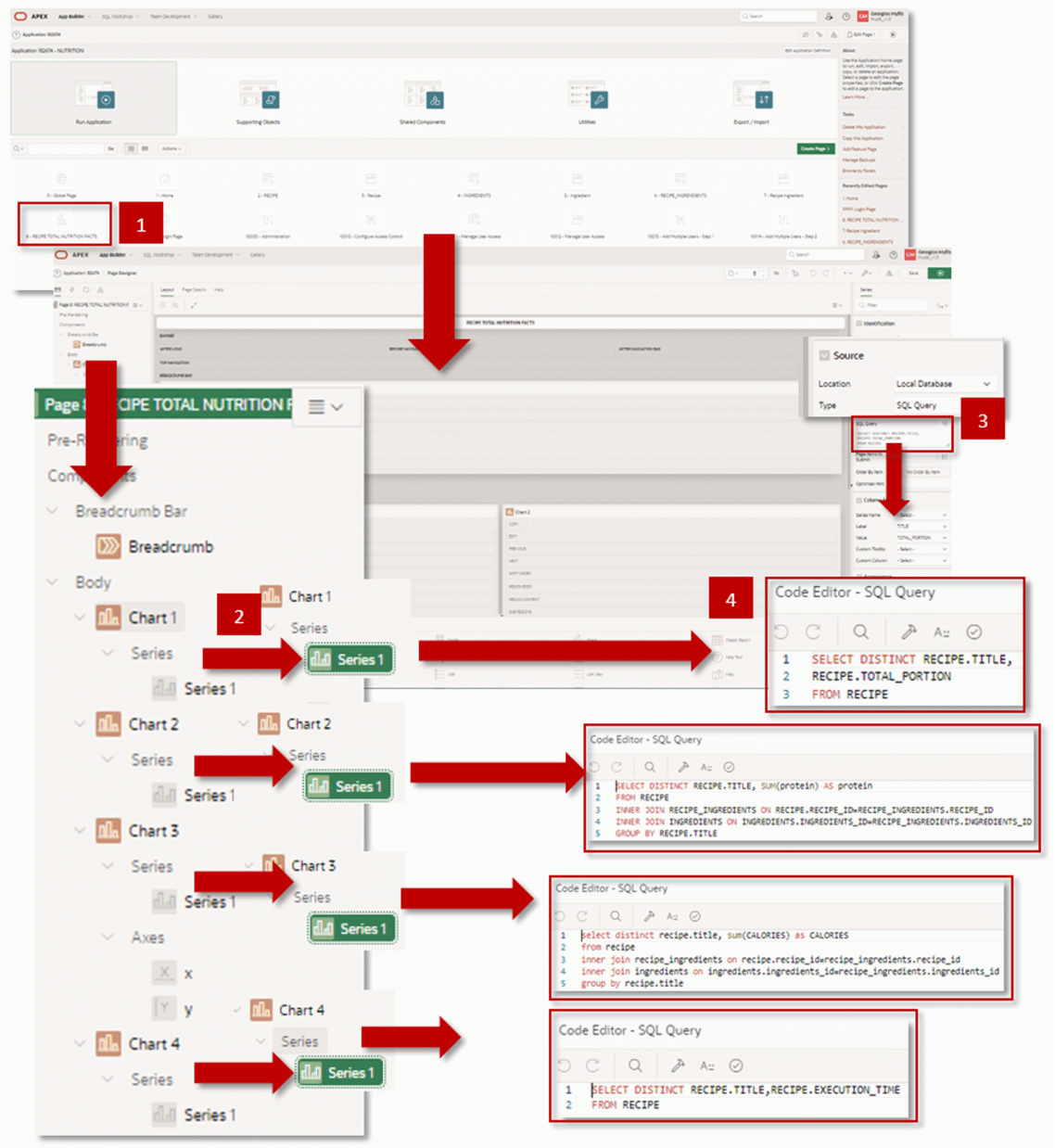


Figure 21.14: Configure SOURCE with SQL QUERY.

21.5.2 Developers and application users

In Oracle APEX developers and end users must have accounts, however we must distinct privileges for workspace, application and specific page. Workspace privileges are related to workspace administration, development, access to App Builder, access to SQL workshop and access to Team development. Application users have all these privileges disabled. To create new user account select **Manage Users and Groups** from the drop-down menu. Then, select **Create User**, enter the user's identification and password, define privileges (administrator, developer, access to tools) and click on **Create User** to complete the process (see Figure 21.15).

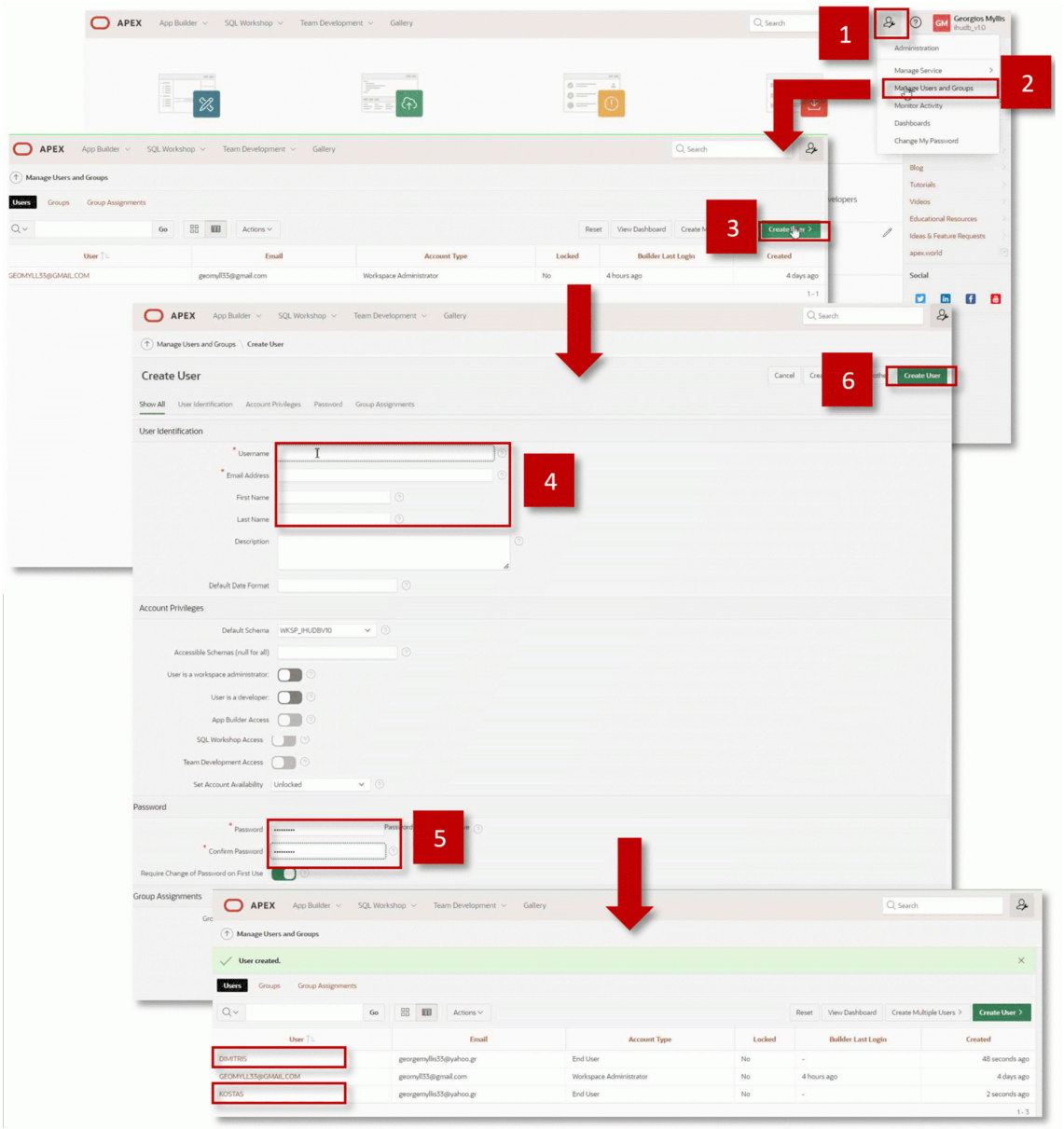


Figure 21.15: Creation of new user account.

Application users and their application privileges (authorization) can be defined (created) in the application Administration page. To have this page generated, tick the Access Control feature executing the application generator. For this specific application, access the mega menu and click **Administration**. From there, click on **Add User**, enter the user's identification and role, and finally click on **Add User**. You can refer to figure 21.16 for more guidance.

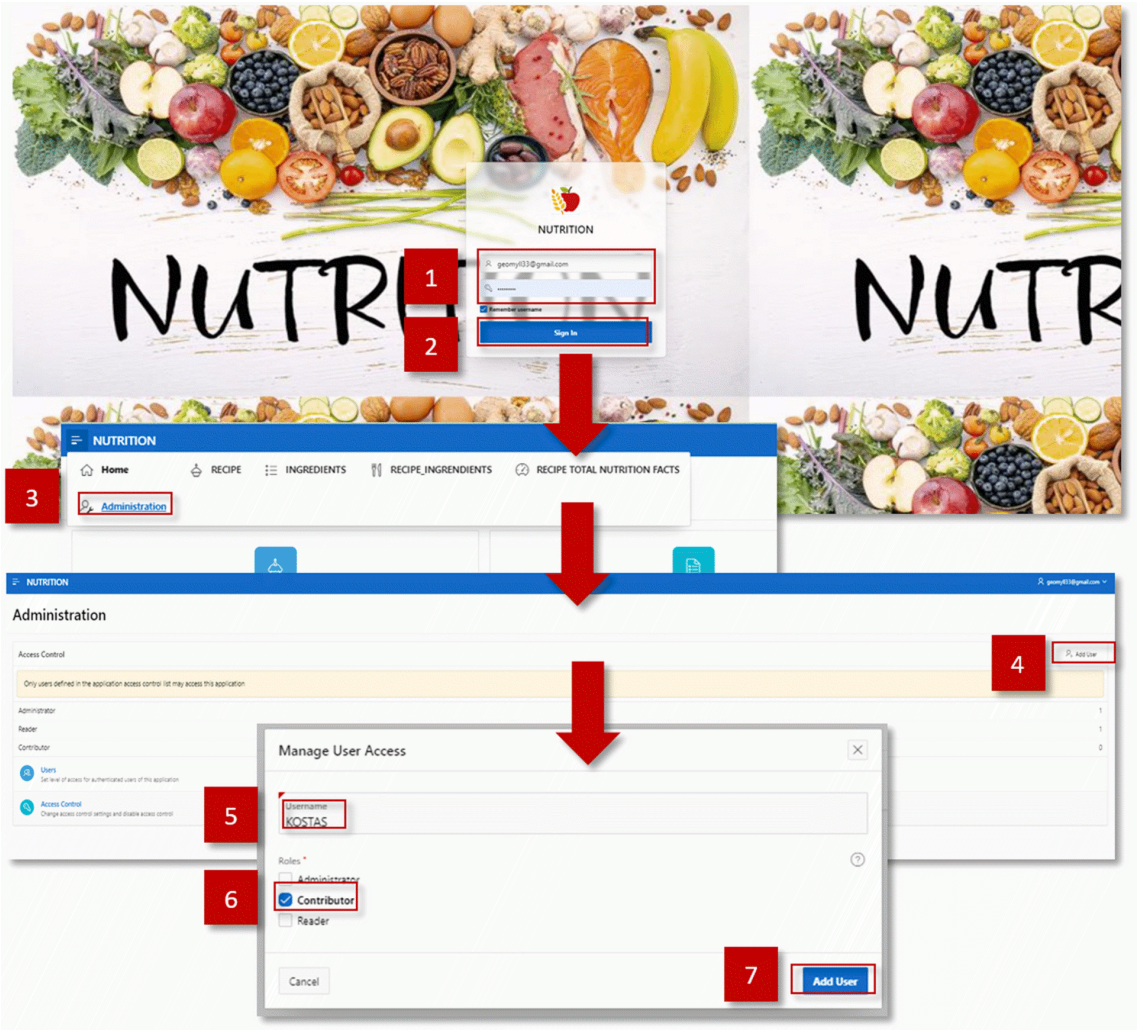


Figure 21.16: Create User account within application.

You can grant privilege to access to specific user or role. In page designer (see Figure 21.17 for recipe page) find security options. Select **Contribution Rights** as the authorization scheme. Click on **Save**. This will ensure that only users with the contribution role will be able to create recipes.

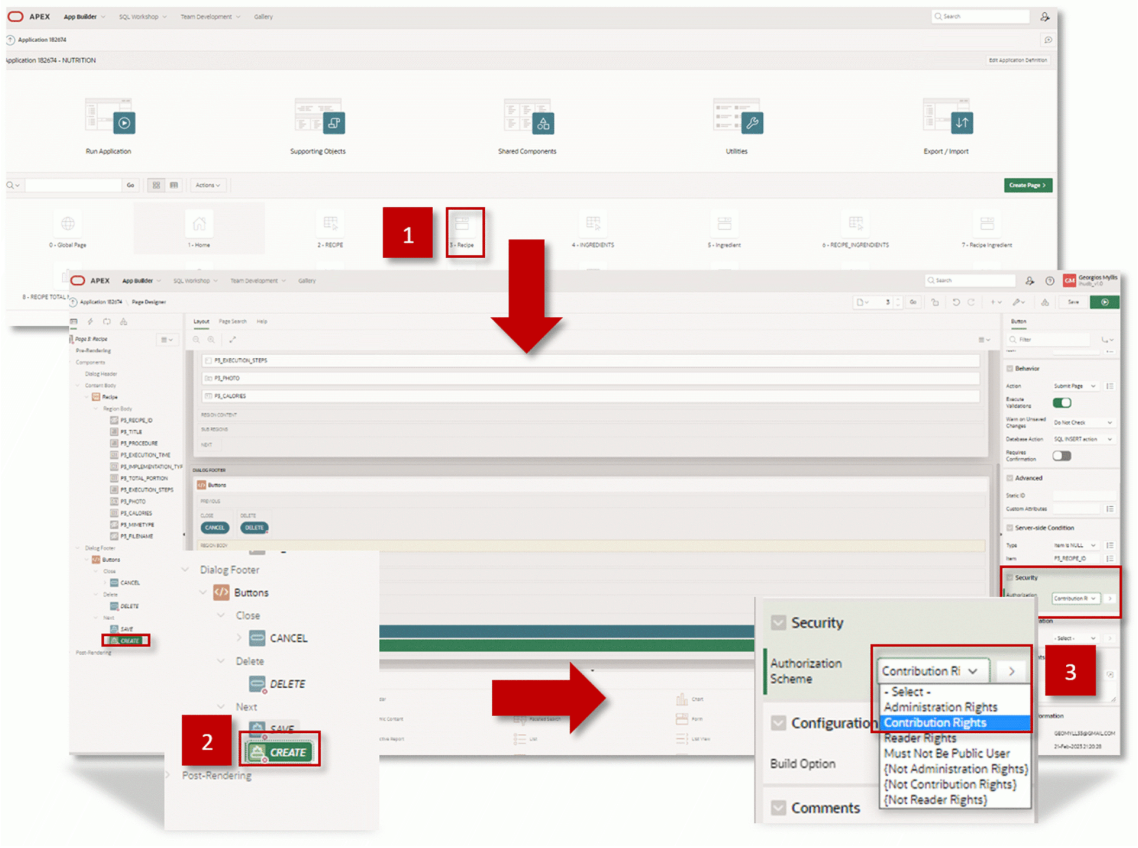


Figure 21.17: Configure pages with user rights.

21.6 Supplementary learning material

You can find the following supplementary learning material:

- exported application
- video guides

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter 21 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

21.6.1 Exported application

Exported application is packaged. Installation create tables, index, function, procedure and trigger as well it populate data. De-installation removes all data base objects used in this application.

21.6.2 Video guides

Video guide show every step in application development.

21.7 Questions

1. How can you change the background in your log in page?
2. Where else can we find role management in Oracle APEX environment if it is not enabled in our app?
3. How can you configure the button name of a report?
4. How can you change the page name after creating it?

21.8 Answers

1. In your app environment, choose **Shared Components** then choose **Static Application Files** click on **Create File** and in the pop up window add the image of your preference. Then copy the reference name of the image, go back to your app environment and choose **9999-Login Page**, go to css field and type in inline

```
body {
background-image:
url("#APP_FILES#\textit{the reference name of the image}");
background-size:1700px 900 px:
}
```

2. In your app environment, choose **Shared Components** then choose **Application Access control**.
3. To change the label of the **Create** button to a different name, you should enter the Page Designer of the report, locate the **Create** button, and change the label in the Identification field.
4. To change the name of a page that you have created, log in to your app and click on **Quick Edit**. Then, click on the page title and select the arrow in the source area of the breadcrumb. Next, click on **Edit Component**, and in the **Short Name** field, update the name to the desired new name. Finally, click on **Apply Changes**, and the new name will appear in the page title.



22. Office Hours Scheduling

J. MAŃKO, M. SOŃTA, R. LESKOVAR

22.1 Business view of the case

A university Office Hours booking application from a business perspective can be seen as a tool to streamline and optimize the scheduling process for university staff. It can help to improve scheduling efficiency, reduce administrative workload, increase student satisfaction, and provide real-time availability information. The application can also generate data and insights on appointment patterns and utilization, which can inform decision making and resource allocation. In addition, the application can offer a convenient and accessible booking platform for students, helping to foster a more proactive and engaged student community. Each week, the academic staff is obliged to make 60 minutes of their time available to the students, which is internally called as 'Office Hours'. Moreover, it happens quite frequently, especially as semester end approaches, when the inquiries from students intensify both in number and scope. In such case, 60 minutes per week is not enough, as students are determined to get urgently desired outcome from the academic staff as soon as possible. Even though, there are some standardized topics for discussion such as: dissertation, exams, final assignments, however, some of the cases are related to irrelevant subjects which perhaps should be discussed, but not necessarily during the Office Hours with the teacher (for example: administrative requests or the problems that can be managed easily without the intervention of the academic teacher). In the last years and especially due to COVID-19 pandemics, universities underwent radical technological transition enabling various forms of remote teaching using available platforms and communicators that would facilitate online teaching process. In fact, virtually all academic activities were transferred online including, naturally, remote Office Hours. Even though there is an agreement among academic community that remote teaching will never fully replace on-site teaching, remote teaching and even working continues to be implemented to some extent worldwide. More importantly, some university activities and processes already transitioned or will transition to online mode entirely in the upcoming future, for example to ensure efficient paperless document circulation. These circumstances strengthen even further an urgent business need of a viable digital tool that would enable online booking of Office Hours that could take place either online or offline. Clearly, a whole academic community would greatly benefit from such application which should soon become everyday tool for everybody within academic community to use, such as email box, and not some distant digital fad.

22.2 Problem definition

As of now, it is not possible to book (enroll) the time slot and plan the content of the meeting in advance. In fact, students, if they appear in whatever number, are served on the first-come-first-served basis. Not only does it increase frustration of the students, who need to wait in the line without any certainty that their appointment will even take place, but it also upsets academic staff who must choose between letting some students wait in vain or rescheduling their plan for the day. As a result, there come situations in which academic staff is effectively doing pro bono counseling for students, or even at the expense of their private life, just to avoid being perceived as unavailable or unfriendly by students. Furthermore, this status quo amounts to inefficient time allocation for both students and academic staff, as well as unnecessary scheduling workload for the latter.

This administrative systemic inefficiency enforces academic staff to engage in additional, often unpaid responsibilities, while also keeping students uncertain about the outcomes and most likely barely satisfied with the whole process. It is a paramount example of shifting the consequences of a malfunctioning meeting appointment system to groups that are situated lower in the academic hierarchy (students and academic staff), instead of finding top-down solution initiated by the university authorities. Therefore, technology offers here a viable, empowering, and implementable solution that would greatly alleviate academic staff workload and the same time ensuring higher satisfaction and efficiency of use for the students.

Students frequently address student office the questions about dates and places of office hours. Since there is no reliable source they rely on experiences and they often direct students to wrong places at wrong times. According to the rules of academic institution every change of office hours should be approved by vice dean. The approvals are granted in direct communication between teacher and vice dean and student office has no information about it. Vice dean therefore wishes the student office would enter the rescheduled office hours.

Management has little evidence of the big picture of office hours in academic institution. Rumors and other unreliable information are not good foundations for rational decisions about increased or decreased number of duty hours. It can also be related to the awarding teachers for their extra efforts.

22.3 Use cases

The Office Hours booking application enables access to management, student office personnel, students and academic staff (also referred as teachers), each having different privileges:

- Management get summary information gathered on management dashboard (number of office hours by year and month, top 10 teachers by number of students, top 5 overbooked teachers and top 5 teachers with no students during office hours).
- Student office personnel access office hours interactive report and reschedule any office hours.
- Students can access calendar of his/her appointments, calendar of all office hours offered by teachers engaged in student's study program, enrollment through calendar and detailed interactive report on student office hours enrollment.
- Academic staff/teachers can get detailed interactive report on teacher's office hours (duty hours), calendar with teacher's office hours (duty hours) and may reschedule office hours with no enrollments.

22.3.1 Narrative description

For the conciseness sake we will present only two use cases out of several:

- Rescheduling of office hours by the teacher where no students are enrolled requires the teacher to log-in, overview scheduled office hours with no one enrolled and "move" one by one to other dates.

Table 22.1: Use case description: rescheduling of office hours by teacher.

Keyword	**Value**
ID:	*Ch22-01*
Title:	*Rescheduling office hours by teacher*
Description:	*Teacher uses APEX application to reschedule office hours where no student is enrolled.*
Primary Actor:	*teacher*
Preconditions:	access to web browser on mobile device or PC, user has Teacher credentials and privileges, application web site is available.
Postconditions:	data stored in data base table ch22_duty_hours
Main	-
Success Scenario:	1. teacher opens the web browser and sign-in to the application 2. select reschedule duty hours with none enrolled 3. click the slot marked as none enrolled 4. on form enter rescheduled data (location, start and end time) 5. review and confirm the rescheduling by clicking appropriate button
Extensions:	-
Frequency of Use:	*Approximately 4 out of 30 obligatory dates in one academic year.*
Status:	[Finished]
Owner:	*User with the teacher privileges*
Priority:	*moderate*

- Student enrollment for office hours consists of overview of calendar (all office hours offered by all teachers engaged in student-s study program), selecting desired office hours, entering the purpose and confirming enrollment. It is not allowed to enroll more than once per scheduled office hours.

22.3.2 Semi-structured description

First use case is rescheduling office hours by teacher when no student enrolls (see Table 22.1). We limit the rescheduling to this type because no student is affected. The second use case targets student enrollment for office hours (see Table 22.2). The application should also prevent student to enroll to one office hour slot more than once.

22.3.3 Use case diagram

Figure 22.1 presents identified use cases. Note that use case "sign-in" is performed only once per user session and that none of the other use cases can be executed before successful completion of "sign-in". Three use cases, namely "reschedule office hours","enrollment to office hours" and "reschedule office hours with none enrolled" are dependent and noted with "extend" association. These use cases are executed upon request after parent use case is finished.

Table 22.2: Use case description: student enrollment for office hours.

Keyword	**Value**
ID:	*Ch22-02*
Title:	*Student enrollment for office hours*
Description:	*Student uses APEX application to enroll office hours. The calendar has all scheduled office hours for all teachers that are engaged with student study programs (departments). Usually one student is enrolled to one study program, but exceptional individuals may be permitted to study two or more. Application will prevent multiple enrollments of one student to specific office hours.*
Primary Actor:	*student*
Preconditions:	access to web browser on mobile device or PC, user has teachers credentials and privileges, application web site is available
Postconditions:	data stored in data base in table ch22_dh_slot
Main	-
Success Scenario:	1. student opens the web browser and sign-in to the application 2. enroll through calendar 3. click the desired slot (slots are colored indicating the statuses) 4. on new form student clicks the button and confirms the enrollment
Extensions:	-
Frequency of Use:	*Over 1000 students use office hours at on average 2 times per semester for 10 to 15 teachers.*
Status:	[Finished]
Owner:	*Student*
Priority:	*high*

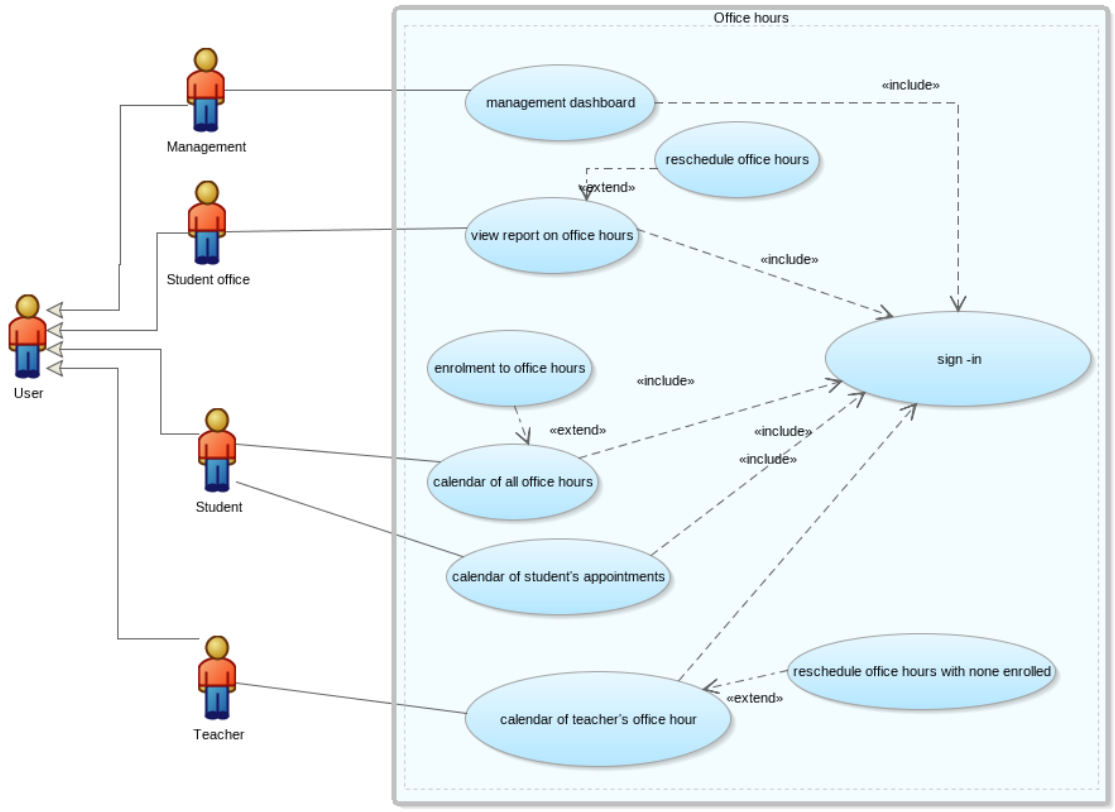


Figure 22.1: Use case diagram.

22.4 Data model

22.4.1 Narrative description of data model

There are ten entities in the logical data model:

- **ch22_dh_slot** present one student-teacher meeting;
- **ch22_duty_hour** present preallocated teacher time slice (office hours) which points to several slots aka student-teacher meetings;
- **ch22_purpose** stores reasons and average duration of student-teacher meetings;
- **ch22_student_dept** points to one study program of one student, Exceptional students are allowed to be enrolled in more than one program (synonymous for department) in one academic year;
- **ch22_student** is a person who is studying at a university and involved in the study program (synonymous for department);
- **ch22_teacher_dept** points to one study program (aka. department) of one teacher. One teacher is usually engaged in more than one study program;
- **ch22_teacher** is a person that conducts academic activities (lessons, lab exercises, etc.)
- **ch22_location** is either physical (R) or virtual (I) or both (B) - it is a place where office hours happens;
- **ch22_department** is a synonymous for study program;
- **ch22_acad_cal_umb** holds specifies working/non-working days. This entity is related to all entities with some date attributes however we will not model these relations explicitly because implicit relation is enough for completeness of the model.

22.4.2 Logical data model

Logical data model is presented on Figure 22.2.

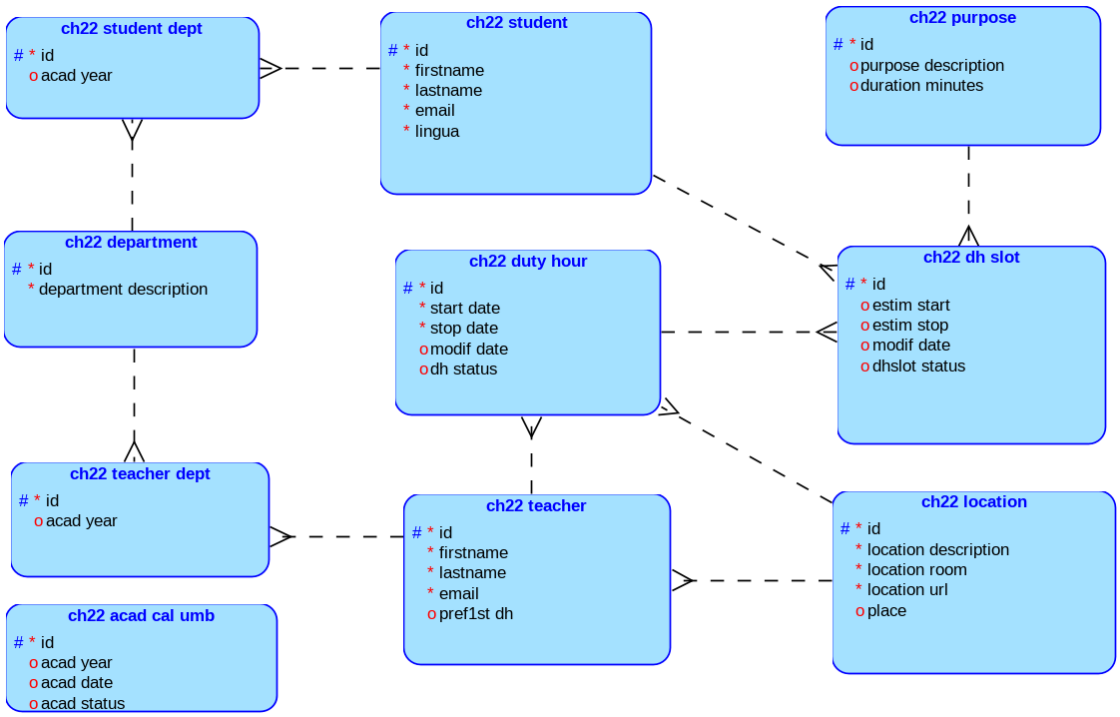


Figure 22.2: Logical data model.

22.4.3 Relational data model

Automatic transformation from logical data model to relational data model in Oracle SQL Data Modeler is provided by function *Engineering to relational*. The result, relational data model ready to be exported as SQL script is shown on Figure 22.3.

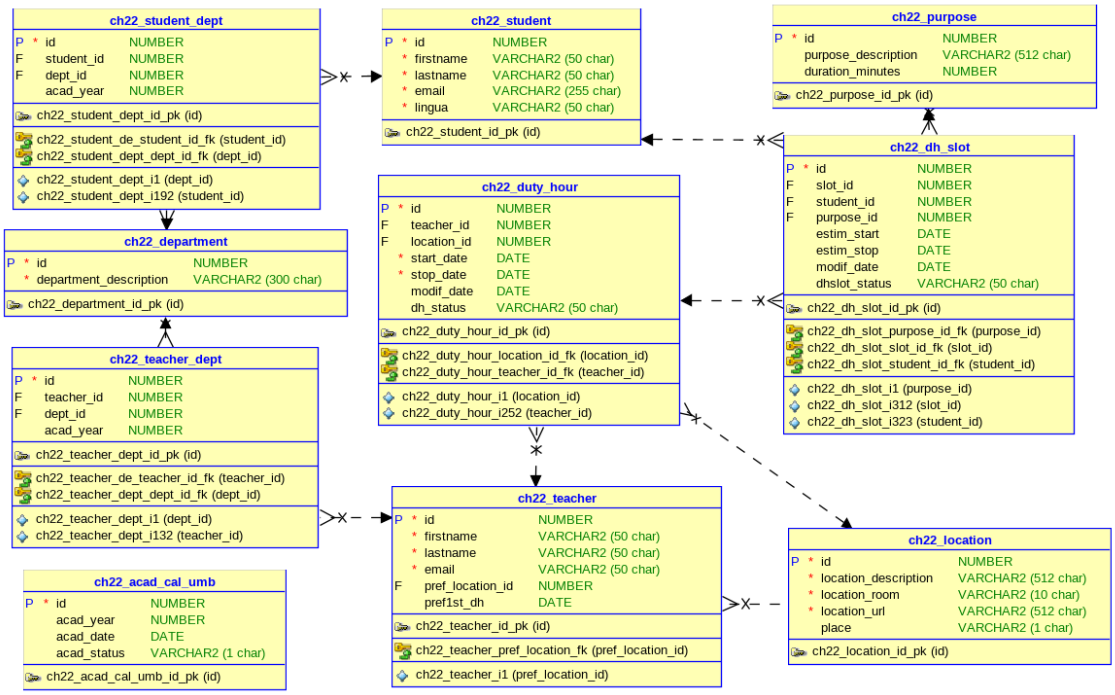


Figure 22.3: Relational data model.

Relational data model can be developed without SQL Data Developer Data Modeler. We can use Quick SQL in APEX.

22.4.4 Quick SQL

Presented Quick SQL (see Figure 22.4) contains all necessary facts for generating SQL script.

```
ch22_acad_cal_umb
    acad_year num
    acad_date date
    acad_status vc1 /check y n
ch22_department
    department_description vc300 /nn
ch22_purpose
    purpose_description vc512
    duration_minutes num /check 10 15 20 25 30 45 60 /default 15
ch22_location
    location_description vc512 /nn
    location_room vc10 /nn /default R1
    location_url vc512 /nn /default https://beeapex.eu/bbb
    place vc1 /check R,I,B /default I
ch22_teacher
    firstname vc50 /nn
    lastname vc50 /nn
    email vc50 /nn
    pref_location_id num /fk ch22_location
    pref1st_dh date
ch22_teacher_dept
    teacher_id num /fk ch22_teacher
    dept_id num /fk ch22_department
    acad_year num
ch22_student_dept
    student_id num /fk ch22_student
    dept_id num /fk ch22_department
    acad_year num
ch22_duty_hour
    teacher_id /fk ch22_teacher
    location_id /fk ch22_location
    start_date date /nn
    stop_date date /nn
    modif_date date
    dh_status vc50
ch22_dh_slot
    slot_id /fk ch22_duty_hour
    student_id /fk ch22_student
    purpose_id /fk ch22_purpose
    estim_start date
    estim_stop date
    modif_date date
    dhslot_status vc50
ch22_student
    firstname vc50 /nn
    lastname vc50 /nn
    email vc255 /nn
    lingua vc50 /nn
```

Figure 22.4: Data model described with Quick SQL.

22.4.5 SQL script for creating tables

SQL script for creating tables is provided in learning materials as file CH22CREATE.sql. Script for creating tables is also available in packaged application in learning materials.

22.4.6 Query builder in APEX

This particular application will use quite long queries that may scare APEX beginners. But there is remedy - Query Builder which is embedded in APEX (SQL Workshop > Utilities > Query Builder). It can help anyone to construct correct queries just by clicking and dragging. Figure 22.5 presents an example query for enrolled students calendar.

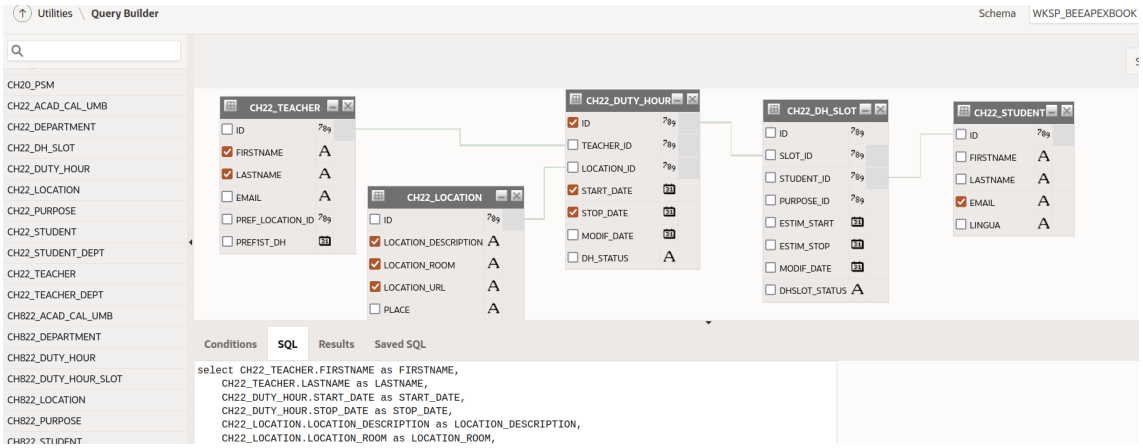


Figure 22.5: An example Query Builder usage.

Generated SQL statement has the following form:

```
select CH22_TEACHER.FIRSTNAME as FIRSTNAME,
       CH22_TEACHER.LASTNAME as LASTNAME,
       CH22_DUTY_HOUR.START_DATE as START_DATE,
       CH22_DUTY_HOUR.STOP_DATE as STOP_DATE,
       CH22_LOCATION.LOCATION_DESCRIPTION as
              LOCATION_DESCRIPTION,
       CH22_LOCATION.LOCATION_ROOM as LOCATION_ROOM,
       CH22_LOCATION.LOCATION_URL as LOCATION_URL,
       CH22_DUTY_HOUR.ID as ID,
       CH22_STUDENT.EMAIL as EMAIL
from CH22_DH_SLOT CH22_DH_SLOT,
       CH22_DUTY_HOUR CH22_DUTY_HOUR,
       CH22_STUDENT CH22_STUDENT,
       CH22_TEACHER CH22_TEACHER,
       CH22_LOCATION CH22_LOCATION
       where CH22_DH_SLOT.SLOT_ID=CH22_DUTY_HOUR.ID
       and CH22_DUTY_HOUR.LOCATION_ID=CH22_LOCATION.ID
       and CH22_DUTY_HOUR.TEACHER_ID=CH22_TEACHER.ID
       and CH22_STUDENT.ID=CH22_DH_SLOT.STUDENT_ID
```

By adding additional condition at the end "and CH22_TEACHER.EMAIL = :APP_USER" the query will return enrolled student for that specific teacher. Query Builder would be necessary feature for absolute beginners but experienced users can also benefit using it.

Queries used in this application utilize concatenation operator (||), EXTRACT function, common

table expressions, fetching first 10 or 5 rows only, custom build function CH22_dh_utilization. Readers of this book may benefit by installing packaged application first and examine queries used on calendars, dashboard and reports.

22.5 Application interfaces

Application have one common page - Home page. This page provides some basic information about application (see Figure 22.6).

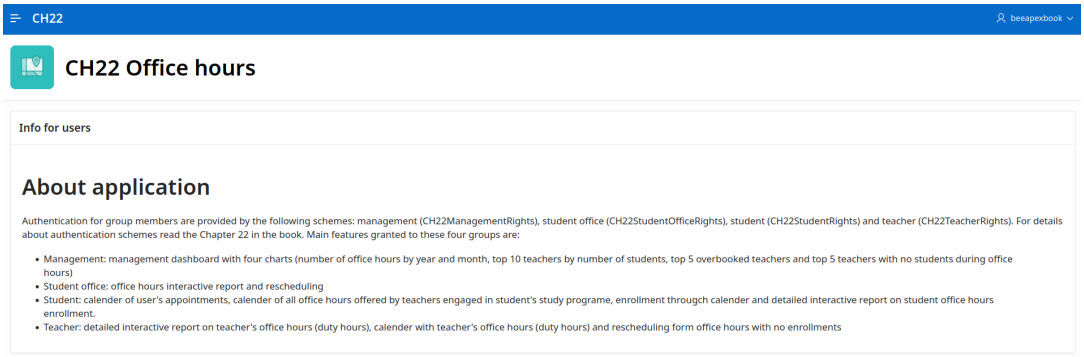


Figure 22.6: Application home page.

Management, student office personnel, students and teachers will be served only with customized interfaces.

22.5.1 Management application interfaces

Users with management rights can access management dashboard (see Figure 22.7). It contains four charts to support decision making related to office hours and teachers:

- number of office hours by year and month
- top 10 teachers by number of students
- top 5 overbooked teachers and
- top 5 teachers with no students during office hours

At the moment management no further management requirements are defined.

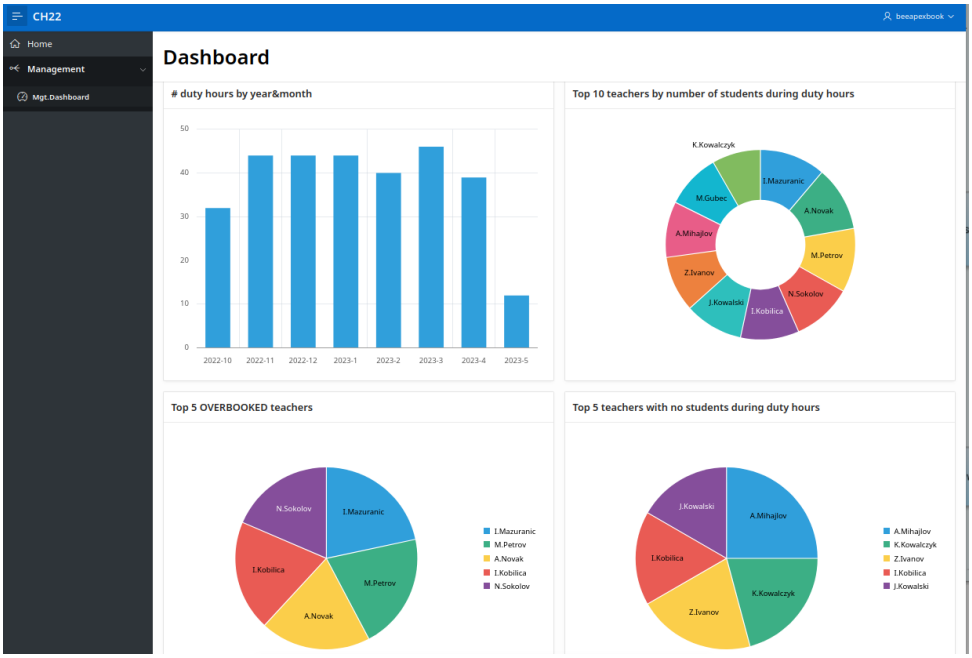


Figure 22.7: Management dashboard.

22.5.2 Student office application interfaces

Users with student office rights can access office hours interactive report and reschedule any office hours. Figure 22.8 shows the interactive report with filter activated (only one teacher is shown). Next, Figure 22.9 shows form which makes rescheduling possible. User can change teacher, location, start and stop dates of selected office hours. Figure 22.10 show report after rescheduling. Two fields are automatically updated: modification date and status of office hours. The latter is concatenated string which also contains the username who made change.

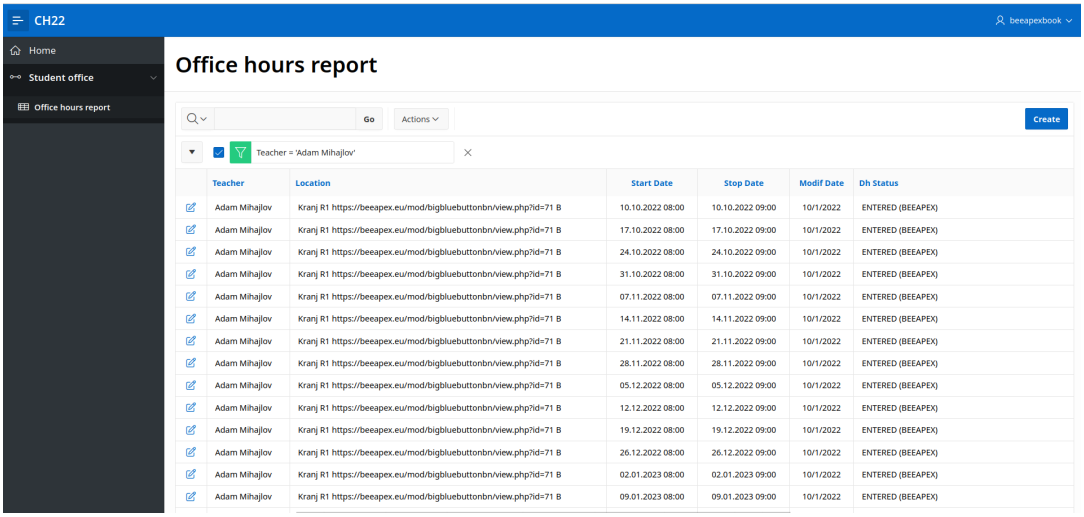


Figure 22.8: Office hours interactive report for student office.

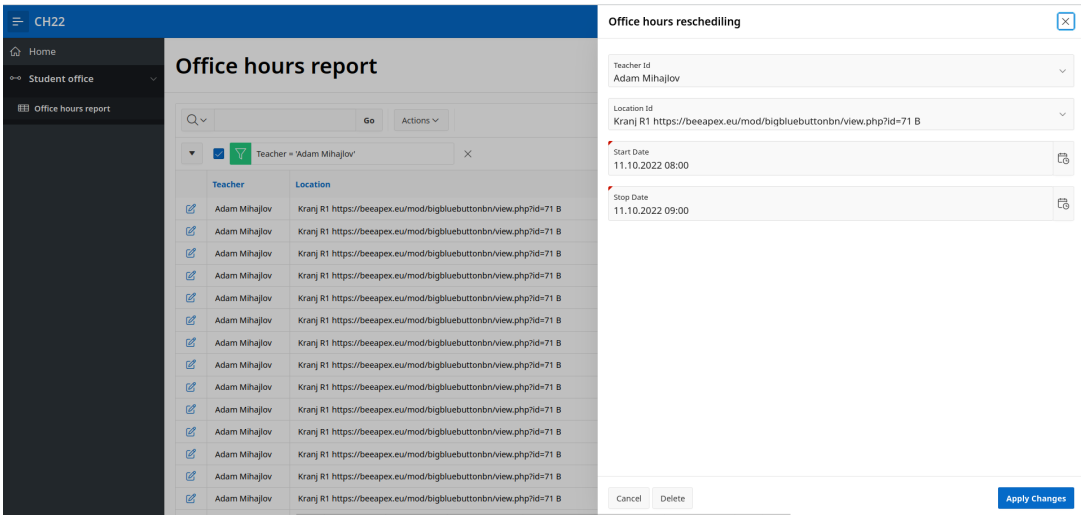


Figure 22.9: Rescheduling form for student office.

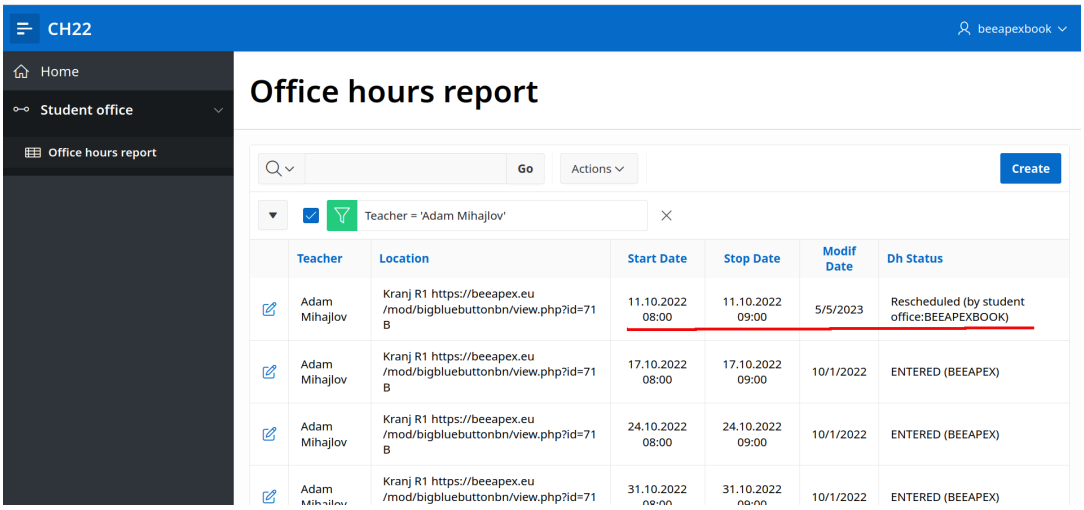


Figure 22.10: Office hours interactive report after rescheduling.

22.5.3 Student application interfaces

Calendar items are colored depending on the utilization. NONE enrolled is presented as green, AVAILABLE is presented as blue and OVEBOOKED is presented as brown. At this moment student may enroll even if OVERBOOKED status is determined. Users with student rights can access:

- calendar of his/her appointments (see Figure 22.11).
- calendar of all office hours offered by teachers engaged in student's study program (see Figure 22.12).
- enrollment to office hours through calendar. First, users picks the desired office hours (see Figure 22.13). On form student select only the purpose (see Figure 22.14) and clicks "Enroll" button. If enrollment is successful then students gets feedback (see Figure 22.15). Checking the enrollment through the "View my appointments" menu item confirms successful enrollment (see Figure 22.16).
- Detailed interactive report present all student appointments in tabular form (see Figure 22.17).

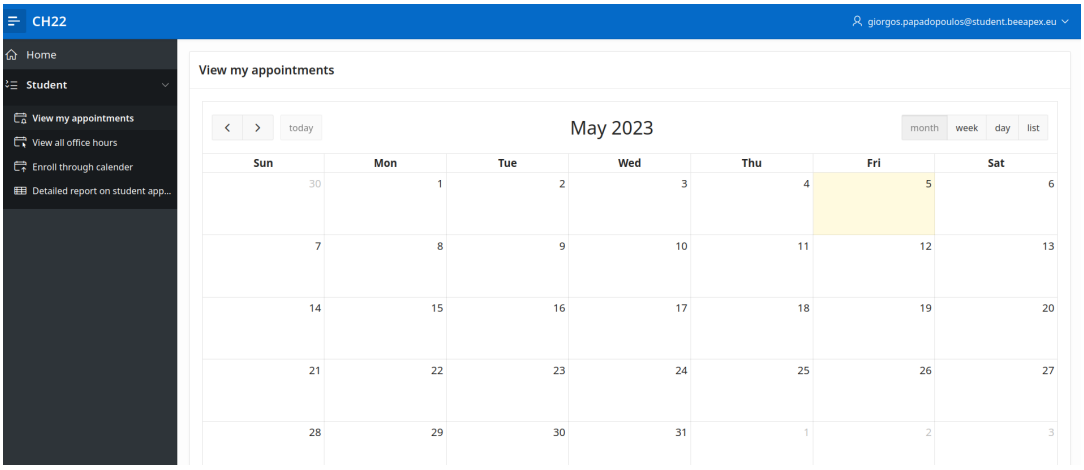


Figure 22.11: Calendar of student appointments.

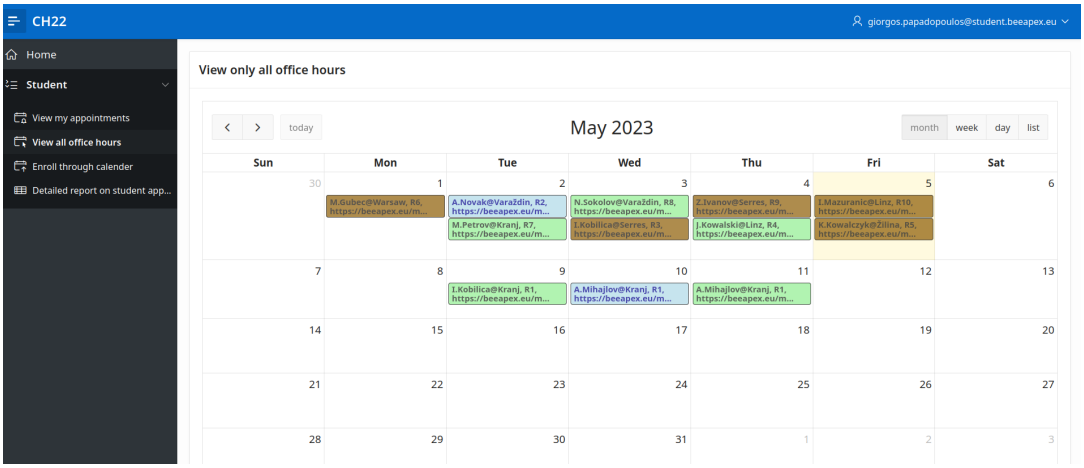


Figure 22.12: Calendar of all office hours offered by teachers in student's study program.

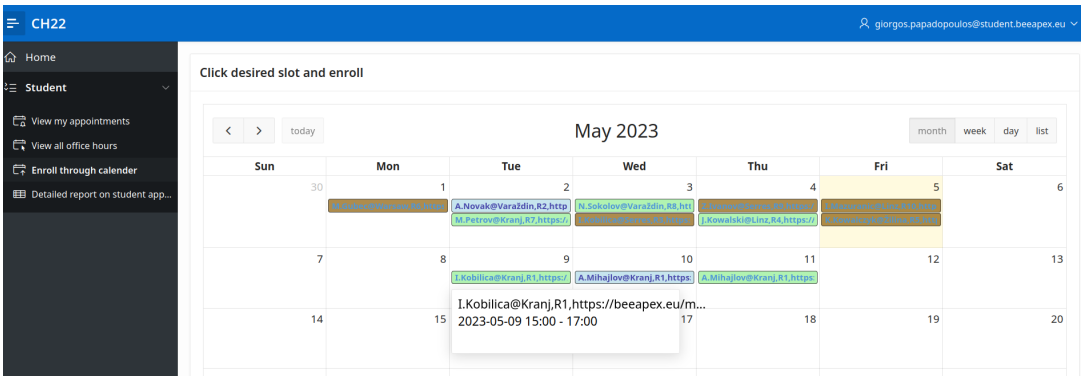


Figure 22.13: Enrollment to office hours through calendar - picking calendar slot.

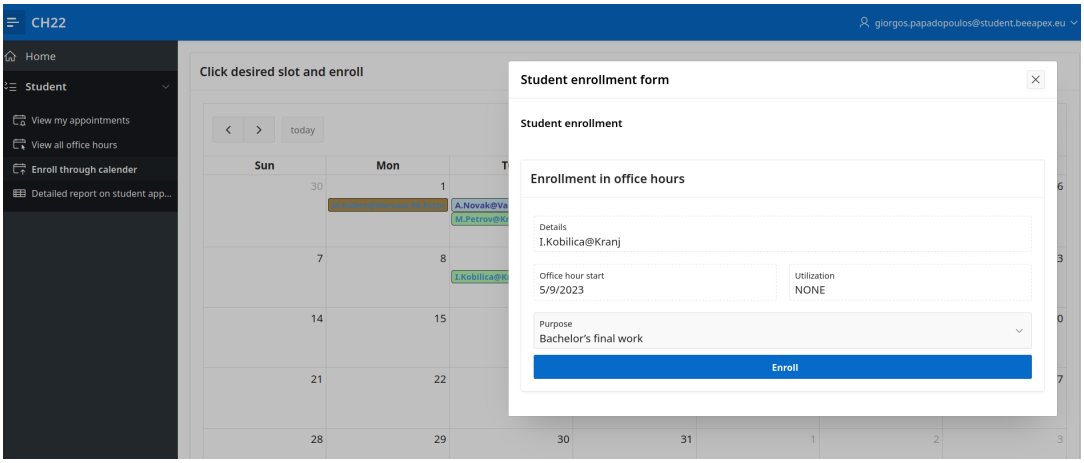


Figure 22.14: Enrollment to office hours through calendar - selecting the purpose.

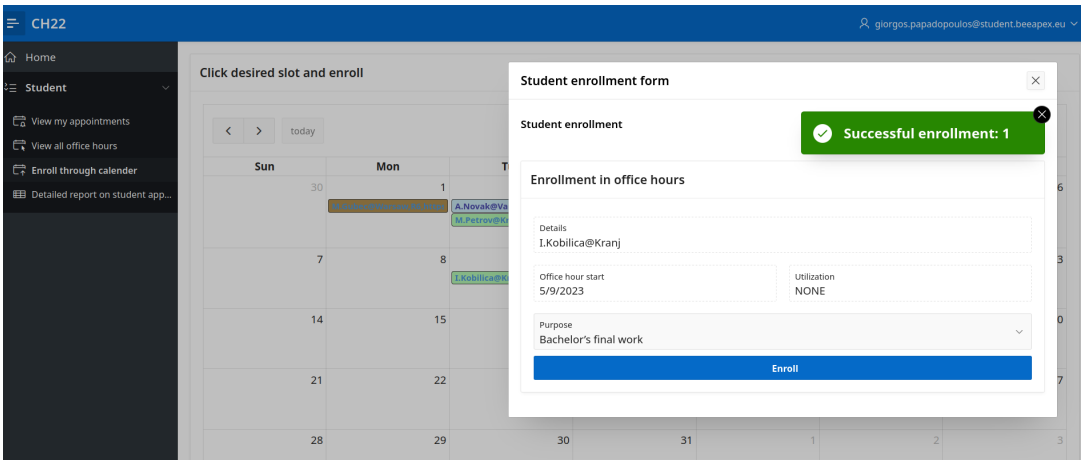


Figure 22.15: Enrollment to office hours through calendar - successful enrollment.

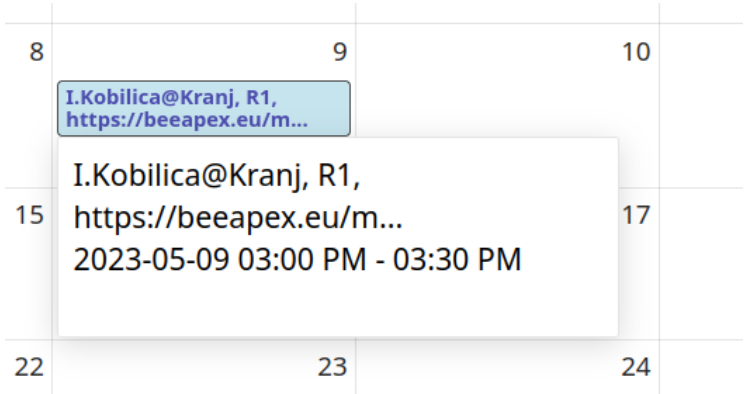


Figure 22.16: Checking the enrollment through the "View my appointments" menu item.

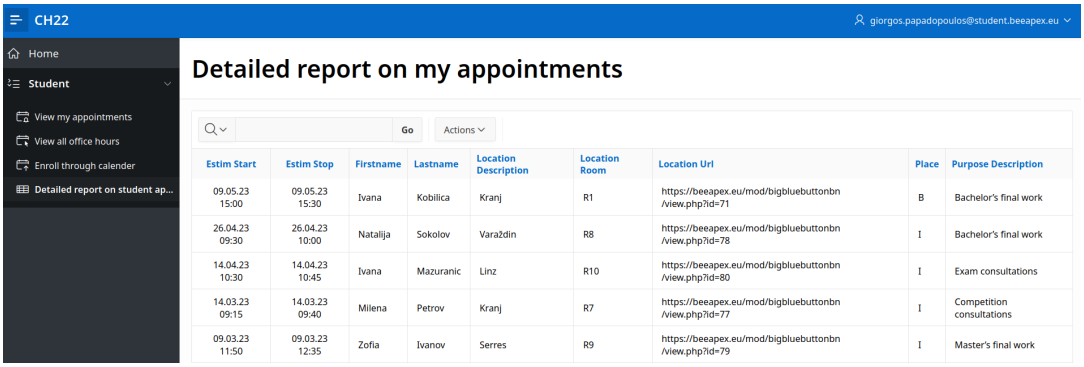


Figure 22.17: Detailed interactive report of all student appointments.

22.5.4 Teacher application interfaces

Calendar items are colored depending on the utilization. Colors of the statuses are identical to ones in student interface (NONE is green, AVAILABLE is blue and OVEBOOKED is brown). Users with teacher rights can:

- view detailed report on teachers' appointments (see Figure 22.18).
- view teacher calendar. Student name, language and purpose is displayed for each student enrolled (see Figure 22.19).
- reschedule office hours with none enrolled. First, the calendar of office hours with NONE status are shown (see Figure 22.20). Clicking the item on calendar opens modal form on which start and stop dates (also hours and minutes) are entered (see Figure 22.21). By clicking "Apply changes" office hours are moved to desired new date (see Figure 22.22).

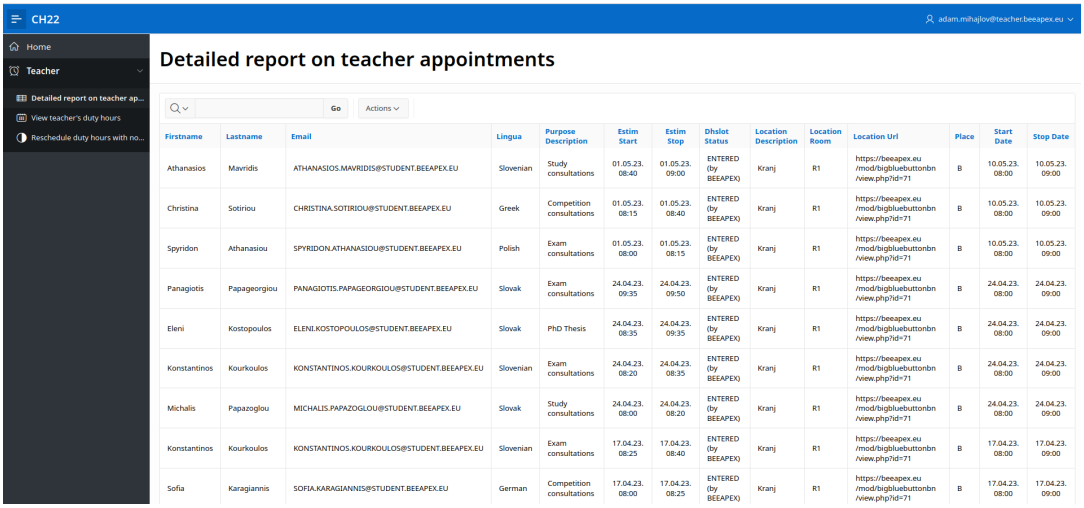


Figure 22.18: Detailed report on teacher appointments.

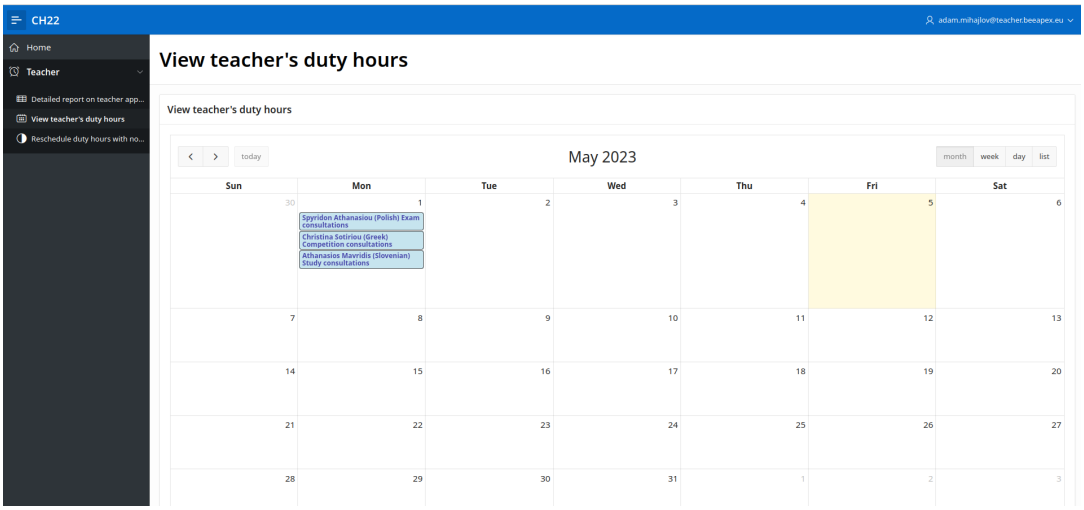


Figure 22.19: Teacher calendar with visible student names, their languages and purposes.

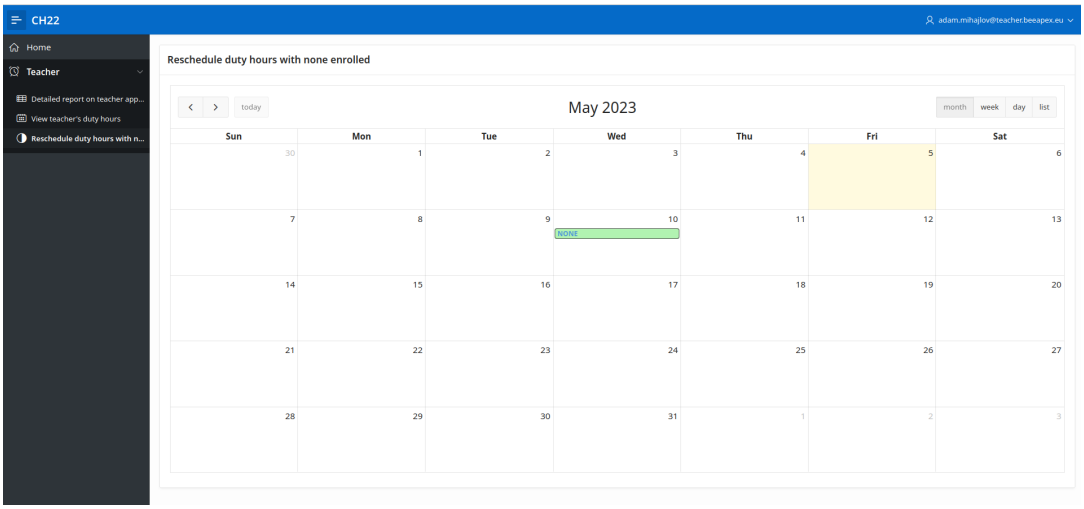


Figure 22.20: Rescheduling teacher office hours with NONE enrolled - the calendar view.

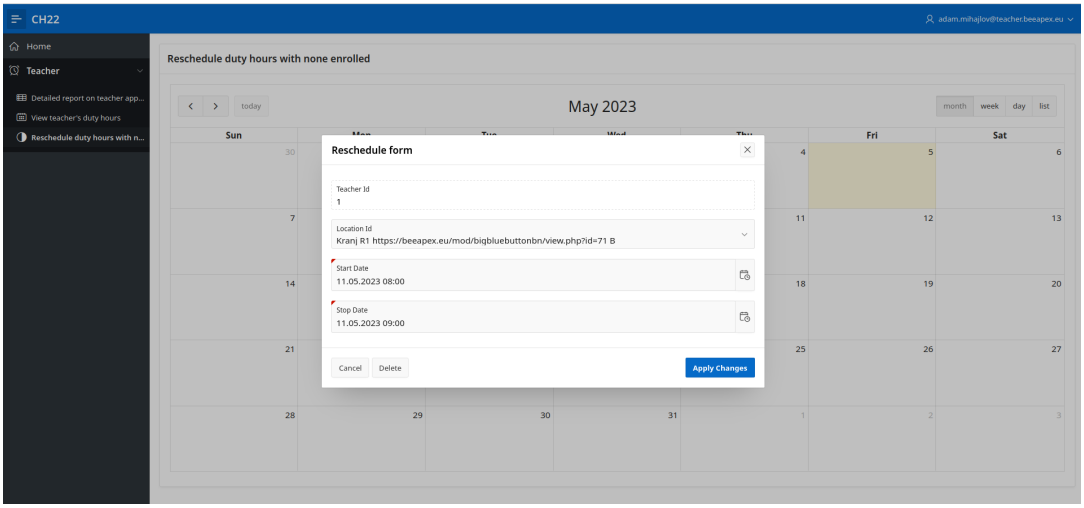


Figure 22.21: Rescheduling teacher office hours with NONE enrolled - new date entered.

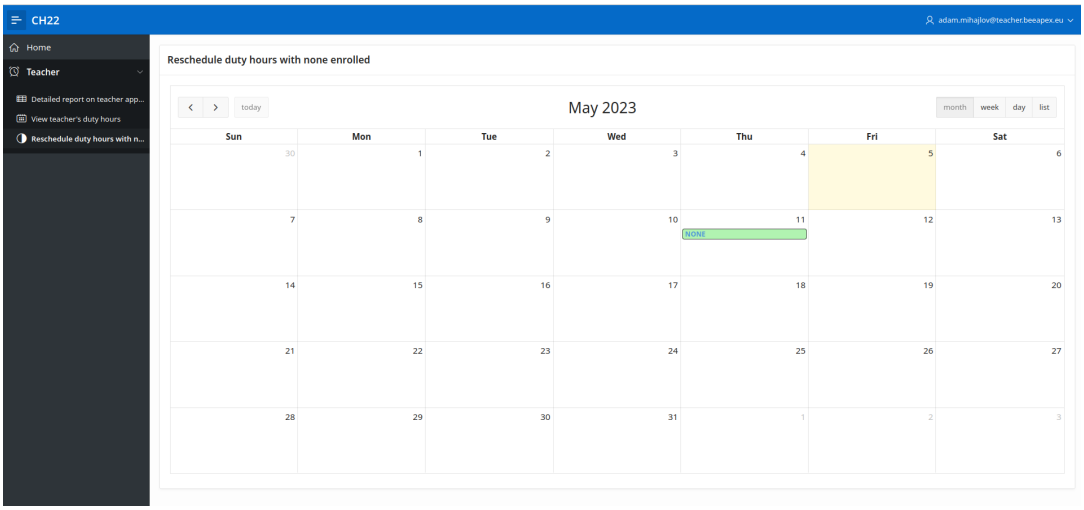


Figure 22.22: The results of rescheduling teacher office hours with NONE enrolled.

22.6 Supplementary learning material

You can find the following supplementary learning material:

- exported application, scripts for creating tables, inserting data, creating PL/SQL function and dropping tables
- video guides: a) installing packaged application and creating users (students and teachers) and b) developing application from scratch

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter22 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

22.6.1 Exported application

Exported application is packaged. Installation creates tables, index, function, as well it populates data. De-installation removes all data base objects used in this application. Packaged application has embedded CSS file (ch22_cal.css) which is referenced by most calendar pages as #APP_FILES#ch22_cal#MIN#.css:

```
.fc-event .fc-content div.fc-time { display: none;}

.fc-event.my-cal-blue {
background-color: lightblue;
border: 0.5pt solid black;
opacity: 0.7;
}

.fc-event.my-cal-blue .fc-event-title {
color: darkblue;
font-weight: bold;
}

.fc-event.my-cal-orange {
background-color: orange;
border: 0.5pt solid black;
```

```
opacity: 0.7;
}

.fc-event.my-cal-orange .fc-title {
color: darkred;
font-weight: bold;
}

.fc-event.my-cal-dark-orange {
background-color: #8B5A00;
border: 0.5pt solid black;
opacity: 0.7;
}

.fc-event.my-cal-dark-orange .fc-title {
color: white;
font-weight: bold;
}

.fc-event.my-cal-white {
background-color: white;
border: 0.5pt solid black;
opacity: 0.7;
}

.fc-event.my-cal-white .fc-title {
color: black;
font-weight: bold;
}

.fc-event.my-cal-green {
background-color: lightgreen;
border: 0.5pt solid black;
opacity: 0.7;
}

.fc-event.my-cal-green .fc-title {
color: darkgreen;
font-weight: bold;
}
```

Management dashboard uses the following queries:

```
SELECT
        EXTRACT(YEAR FROM start_date) || '-' ||
        EXTRACT(MONTH FROM  start_date) AS "Year␣and␣Month",
        COUNT(id) AS count
FROM ch22_duty_hour
        GROUP BY EXTRACT(YEAR FROM start_date),
                EXTRACT(MONTH FROM start_date)
order by 1;

```

```
with teacher_appointmets (appointment, teacher) as
(select CH22_DH_SLOT.ID,
substr(CH22_TEACHER.FIRSTNAME,1,1) || '.' ||
        CH22_TEACHER.LASTNAME
from CH22_DH_SLOT CH22_DH_SLOT,
CH22_DUTY_HOUR CH22_DUTY_HOUR,
CH22_TEACHER CH22_TEACHER
where CH22_DUTY_HOUR.ID=CH22_DH_SLOT.SLOT_ID
and CH22_TEACHER.ID=CH22_DUTY_HOUR.TEACHER_ID)
select count(appointment), teacher from teacher_appointmets
        group by teacher
order by 1 desc
fetch  first 10 rows only;

with teacher_dh_statuses (duty_hour_id, teacher,dh_utilization) as
(select CH22_DUTY_HOUR.ID,
substr(CH22_TEACHER.FIRSTNAME,1,1) || '.' ||
        CH22_TEACHER.LASTNAME,
CH22_dh_utilization(CH22_DUTY_HOUR.ID)
from CH22_DUTY_HOUR CH22_DUTY_HOUR,
CH22_TEACHER CH22_TEACHER
where CH22_DUTY_HOUR.TEACHER_ID=CH22_TEACHER.ID and
CH22_dh_utilization(CH22_DUTY_HOUR.ID) = 'OVERBOOKED')
select count(duty_hour_id), teacher from teacher_dh_statuses
group by teacher
order by 1 DESC
fetch  first 5 rows only;

with teacher_dh_statuses (duty_hour_id, teacher,dh_utilization) as
(select CH22_DUTY_HOUR.ID,
substr(CH22_TEACHER.FIRSTNAME,1,1) || '.' || CH22_TEACHER.LASTNAME,
CH22_dh_utilization(CH22_DUTY_HOUR.ID)
from CH22_DUTY_HOUR CH22_DUTY_HOUR,
CH22_TEACHER CH22_TEACHER
where CH22_DUTY_HOUR.TEACHER_ID=CH22_TEACHER.ID and
CH22_dh_utilization(CH22_DUTY_HOUR.ID) = 'NONE')
select count(duty_hour_id), teacher from teacher_dh_statuses
group by teacher
order by 1 DESC
fetch  first 5 rows only;
```

See other queries for pages which are based on calendar element in the packaged application.

22.6.2 Video guides

Video guides show installation of packaged application and developing application from scratch.

22.7 Questions

1. How to add a user with teacher rights?
2. How would you enhance application by enabling student to take notes for particular meeting?
3. How would you limit query results to rows which have date columns equal or greater than

current time?

22.8 Answers

1. Use Manage Users and Group. Add new user. Use SQL Workshop > Object Browser and select table CH22_TEACHER. Select "Data" tab and "Insert row". Email of inserted row must match created APEX user.
2. Create new table i.e. CH22_NOTES with fields: ID, note and DH_SLOT_ID as a foreign key which references table CH22_DH_SLOT. Create new report and form for student and authorize the page for student. Depending on institutional rules you can grant teachers the right to view or modify student notes.
3. To limit query results to rows which have date columns equal or greater than current time the condition WHERE date_column >= SYSDATE must be added.



23. Telco case

VERONIKA ŠALGOVÁ, JOZEF KOSTOLNÝ, MICHAL MRENA, MICHAL KVET, MIROSLAV POTOČÁR

23.1 Business view of the case

To effectively demonstrate the step-by-step process of developing a prototype application, let's consider the hypothetical company VEYOMI as an example. VEYOMI is a small company focusing on selling telecommunications services to its clients. As VEYOMI aims to expand its product portfolio and cater to a growing customer base, it becomes imperative to establish and maintain a robust application that facilitates seamless management of diverse aspects, including product inventory, client information, and automated billing systems. In an increasingly competitive market landscape, VEYOMI recognizes the paramount importance of delivering quick and convenient access to vital information for its valued customers. By developing an intuitive application, VEYOMI can enhance its responsiveness, ensuring that pertinent details regarding products, services, pricing, and billing are readily available at the fingertips of its discerning clientele.

23.2 Problem definition

In the world of business, it is extremely important to keep records of your clients, the products or services they have purchased, and any ongoing services they are receiving. This allows you to have all the necessary information at your fingertips, making it easier to serve your clients effectively. Additionally, it is crucial to provide a simple and accessible way for your customers to check the status of their services. By offering a user-friendly platform or application, you enable your customers to easily find out what's happening with their services and stay updated. By maintaining accurate records and providing easy access to service updates, you can enhance customer satisfaction, build trust, and improve your overall business operations.

23.3 Use cases

When creating an application, it is necessary to start with the specification of the services that the application should provide and what it should cover. Therefore, preparing a list of individual functionalities and services is crucial. First, however, it is necessary to focus on defining the types of users. For our use, we can consider three fundamental roles – customer, manager, and administrator. Each role has specific services to ensure overall coverage of the application's functionality.

Table 23.1: Use case description: add service.

Keyword	**Value**
ID:	*Ch23-01*
Title:	*Add service*
Description:	*Customer uses APEX application to add new service among available ones.*
Primary actor:	*Customer*
Preconditions:	*Access to application, valid credentials, web application is accessible.*
Post conditions:	*Data on new service is stored in database.*
Main:	*Scenarios*
Success scenario:	1. *Customer log-in to application.* 2. *Customer navigates to Buy extra region of the Home page.* 3. *Customer enters amount.* 4. *With drop-down menu customer selects new service (buy type).* 5. *By clicking "Buy" button customer confirms activation of new service.*
Extensions:	-
Frequency of use:	*Customer changes a set of services approximately once per six months*
Status:	*Finished*
Owner:	*Customer*
Priority:	*high*

23.3.1 Narrative description

For the conciseness sake we will present only two use cases out of several:

- Add Service - Customers can easily request to add new services to their existing subscription. The system activates the new service and updates the customer's billing accordingly. The customer receives confirmation and information about the added service.
- Show State of Services - It is possible to easily view the current status and details of the customer's subscribed services. The system retrieves and presents an overview of the customer's subscribed services. This includes information such as the service type, activation status, remaining usage or validity period, and any associated features or limitations. Within the service overview, customers can access more detailed information about their service usage. This may include data consumption, call minutes, text messages sent, or any other relevant usage metrics.

23.3.2 Semi-structured description

Two use cases are presented in Tables 23.1 and 23.2.

23.3.3 Use case diagram

The customer needs to ensure the purchase of flat rate and extras services, displaying the status of the services, which he should be able to cancel as well. Another essential functionality to cover is the display of an overview of invoices.

The manager user role can access the overview of customers and their addition, editing, and deletion. In addition, the manager can also generate invoices.

The last user role is the administrator, which has one basic functionality: managing managers. The representation of these roles is illustrated in Figure 23.1.

Table 23.2: Use case description: show service status.

Keyword	**Value**
ID:	*Ch23-02*
Title:	*Show services usage*
Description:	*Customer uses APEX application to get insight into services usage.*
Primary actor:	*Customer*
Preconditions:	*Access to application, valid credentials, web application is accessible.*
Post conditions:	-
Main:	*Scenarios*
Success scenario:	1. *Customer log-in to application.* 2. *Customer selects "Stats" tab.* 3. *With drop-down menu customer selects service. Bar graph of that specific service is displayed.*
Extensions:	-
Frequency of use:	*Customer demand an insight approximately once per month*
Status:	*Finished*
Owner:	*Customer*
Priority:	*high*

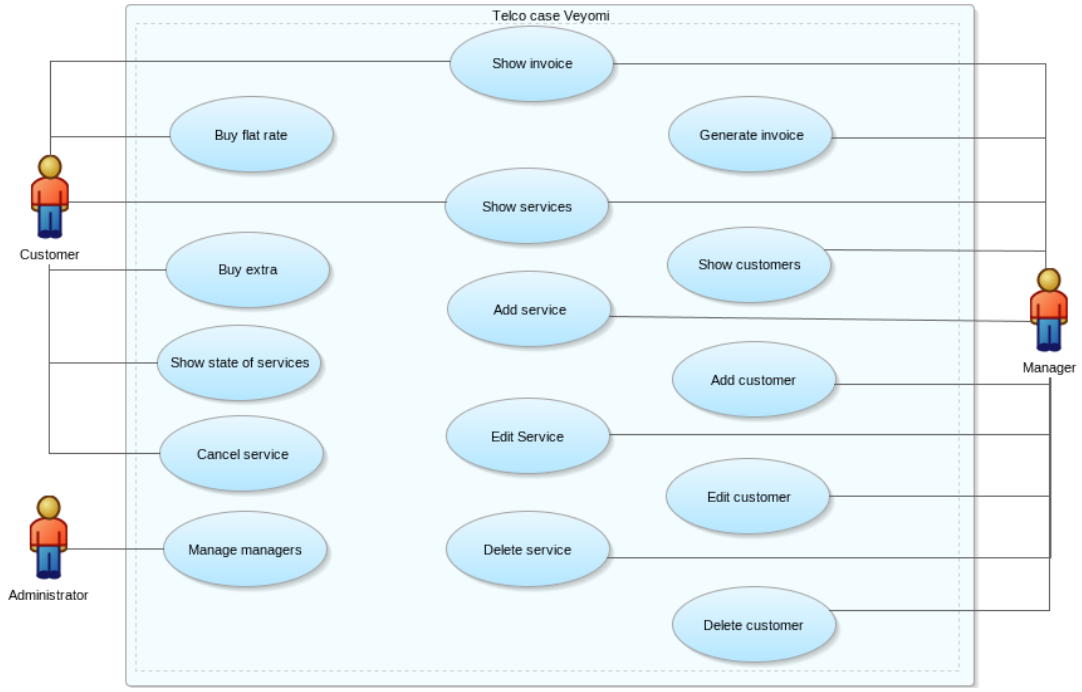


Figure 23.1: Use Case Diagram.

23.4 Data model

23.4.1 Narrative description of data model

The **CH23_Person** table stores the primary personal data and user role type. When the person is a customer, he has a record created in the **CH23_Customer** table, where his address and phone

number are stored in addition to the keys. The **CH23_FlatRate** table stores information about the flat rate, namely the name and price. Data about individual services, their name, price, and offered units are stored in the **CH23_Service** table. Individual services assigned to flat rates are stored in the **CH23_FlatRateService** table, where we know the allocated quantity in addition to the keys. If the customer purchases a flat rate, information about this is stored in the **CH23_CustomerFlatRate** table, including the start and end date. Consumed units from the flat rate are registered in the **CH23_UsageLog** table, where we know which customer used which service within which flat rate, along with the date and quantity. In addition to the flat rate, the customer can purchase extra services recorded in the **CH23_ExtraService** table and their amount.

23.4.2 Logical data model

Logical data model is presented in Figure 23.2.

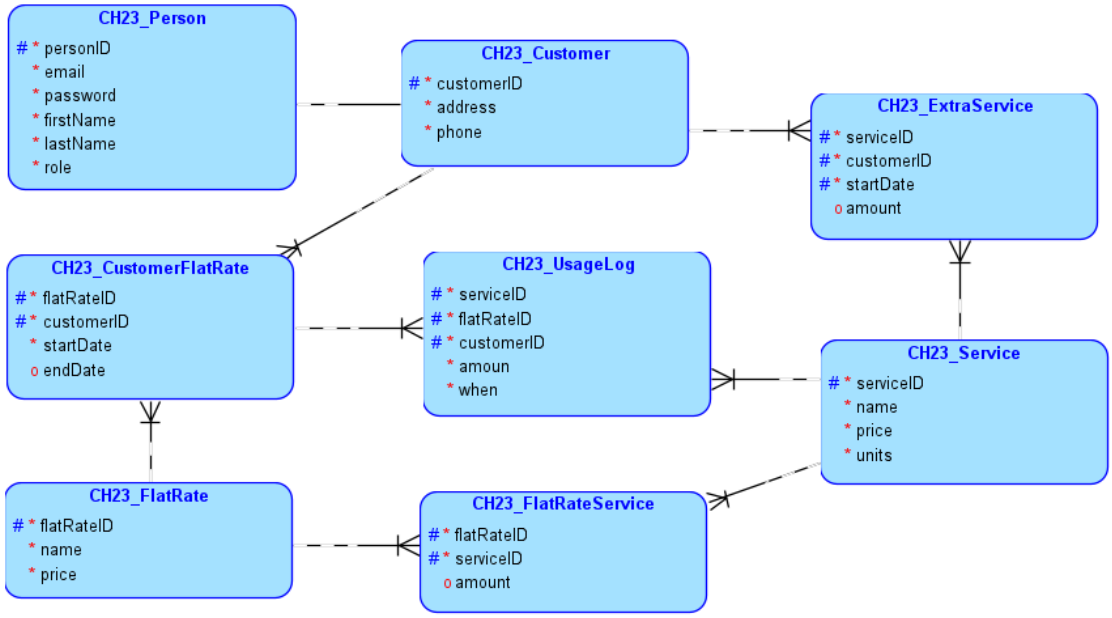


Figure 23.2: Logical data model.

23.4.3 Relational data model

Relational data model is presented in Figure 23.3.

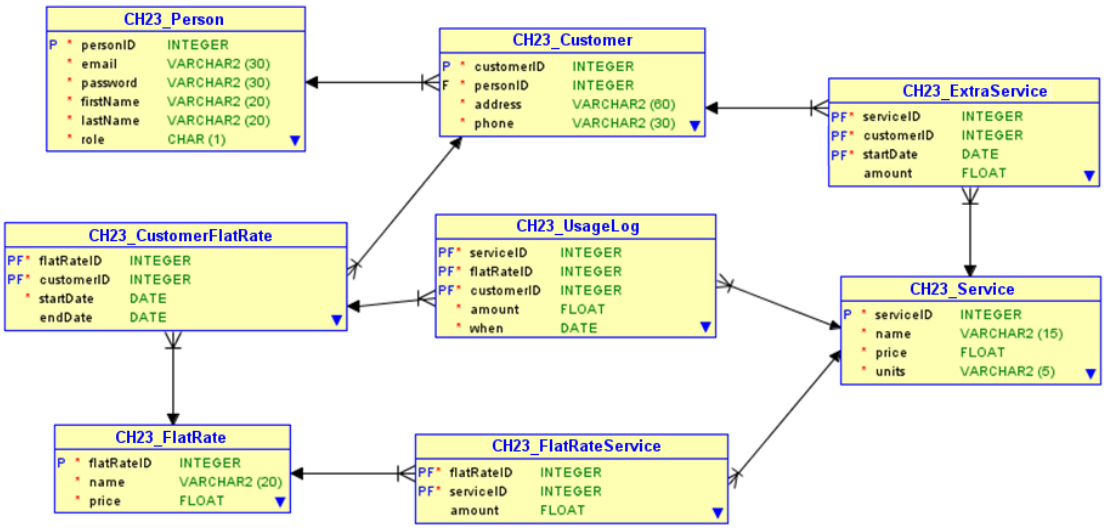


Figure 23.3: Relational data model.

23.5 User authentication and user roles

In the description above, we have identified three user roles. After logging in, the application makes content available to the user according to his role. However, the login process itself is the same for all users. APEX offers us several ways to manage users and authenticate them. In the following list, we present a few selected methods:

- Database account,
- LDAP server,
- Oracle APEX account,
- Social platform login,
- Custom.

In the Oracle Apex tool, we choose the authentication method - **Authentication Scheme** - in the **Shared Components \ Authentication Schemes** sub page. In this sub page, we find a list of existing login schemes, and we can also create a new login scheme here. When creating a new scheme, we can choose one of the above-mentioned authentication methods. For the needs of our application, we chose the most general method, **Custom**.

The main part of **Custom** authentication is the so-called authentication function. The input to this function is the username and password the user enters on the login page, and the output is a **boolean** value. As expected, the function returns **true** if the given name and password match an existing user and **false** otherwise. In Listing 23.1, we can see a header of such a function. Therefore, when creating a **Custom** login scheme, we need to provide the name of such a function. That is, such a function must already exist.

```
CREATE OR REPLACE FUNCTION authenticate_user
(p_username IN varchar2,
p_password IN varchar2)
RETURN boolean;
```

Listing 23.1: Header of an authentication function.

We implement the body of the authentication function using the PL/SQL language, i.e., it is up to us to decide how we authenticate the user. That is the reason why the **Custom** method is the most general. We can, for example, contact an external authentication service using REST API, search for a user in a local database, or leave authentication to another local process. In our

application, we have chosen the commonly used procedure of searching and verifying the user in the local database.

We can see the CH23_Person table in the data model we presented above. The **email** and **password** columns are relevant for authentication. The user's **email** also serves as a unique login name. The **password** column contains user passwords stored in a secure hashed form. The description of secure password storage is beyond the scope of this chapter. The reader can find out more about it in the literature. Nevertheless, before entering data or editing the **password** column in the **CH23_Person** table, it is necessary to hash the password. For this purpose, we created an auxiliary function **hash_password**, the header of which we can see in Listing 23.2. This function takes the username and password in plaintext form and generates a password in a secured hashed format which can be stored in the database.

```
CREATE OR REPLACE FUNCTION hash_password
(p_username IN varchar2,
p_password IN varchar2)
RETURN varchar2;
```

Listing 23.2: The header of the auxiliary function hash_password.

We solved automatic password hashing when working with the **CH23_Person** table by creating triggers for the CH23_Person table. The triggers hash the plaintext password using the above-described **hash_password** function before inserting or updating the **password** column. It will ensure that all passwords in the **CH23_Person** table are stored securely – not even the database administrator has access to the actual password. In Table 23.3, we can see a sample of the data stored in the **CH23_Person** table. Interestingly, two users listed in the table have the same password. However, we cannot determine this fact based on securely stored passwords.

PERSONID	EMAIL	PASSWORD	ROLE
5	DUIS.A@OUTLOOK.CA	$2a$12$EfSjb2zSiKjqWCoZ47mQeOY6IPoHP/6LvECZxGEWjFZIZcXTF3UgGasd	'a'
11	LOBORTIS@OUTLOOK.EDU	$2a$12$IUqALTj3reLAoTdtC5SNde2.C5A.DOm..7pFw9UBnIsaf7gSNcZyapql	'c'
14	AC.FERMENTUM@AOL.NET	$2a$12$GuaO7Tjp.65d7NYLiEjSzeO6VWQfDz8eXja7dWubcjy77aZDAKRTutyq	'm'

Table 23.3: Sample data stored in the CH23_Person table.

Implementation of the authentication function is now simple. In this function, we first search the **CH23_Person** table for the user's password with the given username. We then hash the password that the user entered with the **hash_password** function. If both hashes match, the user authentication succeeds, and we return **true** from the function. On the other hand, if the hashes do not match or there is no user with the given name in the CH23_Person table, we return false.

The disadvantage of the **Custom** method we chose is the greater initial difficulty in creating a database, within which it is necessary to develop auxiliary functions and triggers. On the other hand, the advantage is that we, the application developers, control the entire authentication process. Another advantage is that the implementation used is easily scalable. If, for example, we would like to use an external service for authentication in the future, we would need to change the implementation of the authentication function so that it contacts the external service instead of the local database.

The second important task related to user management is user role management. While the login process was the same for all users, we needed to make different parts of the application access based on the user's role when managing roles. Role management is closely related to user management. First, all possible user roles must be specified in the Oracle APEX environment. We can find role management in the **Shared Components \ Application Access Control** subpage. In this subpage, we see a list of existing roles, and we can also create a new role here. For each new role, it is necessary to specify its static identifier, which we will use later when assigning the role to individual users. In our application, we use roles with the following static identifiers:

- ADMIN,
- CUSTOMER,
- MANAGER.

We assign a role to a user using PL/SQL code. In Listing 23.3, we can see a sample code for assigning the role **CUSTOMER** (static role identifier) to the user **LOBORTIS@OUTLOOK.EDU**. We used another set of triggers for the Person table to automate this process. The **Role** column in this table can take values from the set {'a', 'm', 'c'} that agree with the initial letters of the above-defined roles. In the body of the trigger, before inserting or updating data, we set the appropriate role for the given user according to the value of the Role column using code similar to the code in Listing 23.3.

```
APEX_ACL.REPLACE_USER_ROLES (
 p_application_id    => 151905,
p_user_name          => 'LOBORTIS@OUTLOOK.EDU',
 p_role_static_ids   => wwv_flow_t_varchar2('CUSTOMER') );
RETURN boolean;
```

Listing 23.3: PL/SQL Code that associates a given user with a given role.

The last step of implementing user roles in the application is to display different subpages based on the logged-in user's role. For each subpage that is intended only for a specific user role, it is necessary to select the user role for which the given subpage is intended in the **Page Designer** settings of the page in the **Security \ Authorization Scheme** section. This step ensures that users with other roles cannot access that subpage. For example, the Home page of our application is specific in that it does not require a login to view it. We also set this behaviour in the **Security** section with the **Authentication** item, where we select the **Page Is Public** option.

The settings described above will ensure that the user only has access to the subpages of his role. However, all users will still see all subpages in the navigation menu - even those they do not have access. Such behaviour is certainly not desirable. For the application to show users only the subpages to which they have access, it is necessary to adjust the navigation menu settings. We can set this in the **Shared Components \ Navigation and Search \ Navigation Menu** subpage. Here, in the **Authorization Scheme** column for each item in the menu, it is necessary to set the user role to which the given item will be displayed.

23.6 Application interfaces

The application's design is to create individual subpages that will provide the proposed services. The primary page is the home page, which provides an overview of the offered services and the portfolio of flat-rate services. It is possible to use this page to present the company and the services provided, which the customer can purchase. This page is accessible without logging in for all visitors. The main page offers one more important functionality: access to the administration after logging in. By administration, we mean a set of subpages divided according to the type of account to which one is logging in - customer, manager, or admin. So, for each of them, it is necessary to create separate subpages that cover the proposed functionality of the services of the individual role.

The customer page provides an overview of the current status of ordered services and extra services in individual fields. Another extra tab enables the purchase of particular services by selecting the type of service from the combo box. Another functionality provides displaying and generating invoices, which can be solved with a pop-up window, in which, after opening, the invoice is displayed in the form of a report, which allows the configuration of saving and printing in PDF format. The report for the invoice is called by a particular script summarizing the data for the service provided in the given month. Another critical part of the customer page is the service usage overview with details such as quantity, type of service, and time. This table enables this review to be filtered and sorted using the filter specification and sorting by individual columns. Such functionality can be beneficial for the client when he wants to get, for example, an overview of the number of SMS he sent in a given period. Finally, the customer is provided with the functionality of displaying statistics in graphs in a particular folder. This functionality is handled on a separate page, which is attached to the customer's page by a tab, and it is possible to switch between individual functionalities. The statistics page contains a demonstration of the graphic representation of the logos. Such a representation is a suitable representation for showing change over time. For example, we can easily display minutes called up during individual months, the number of SMS sent, or the amount of transferred data in the ordered program.

23.6.1 Application design

This chapter displays screenshots of the application forms of different user roles.

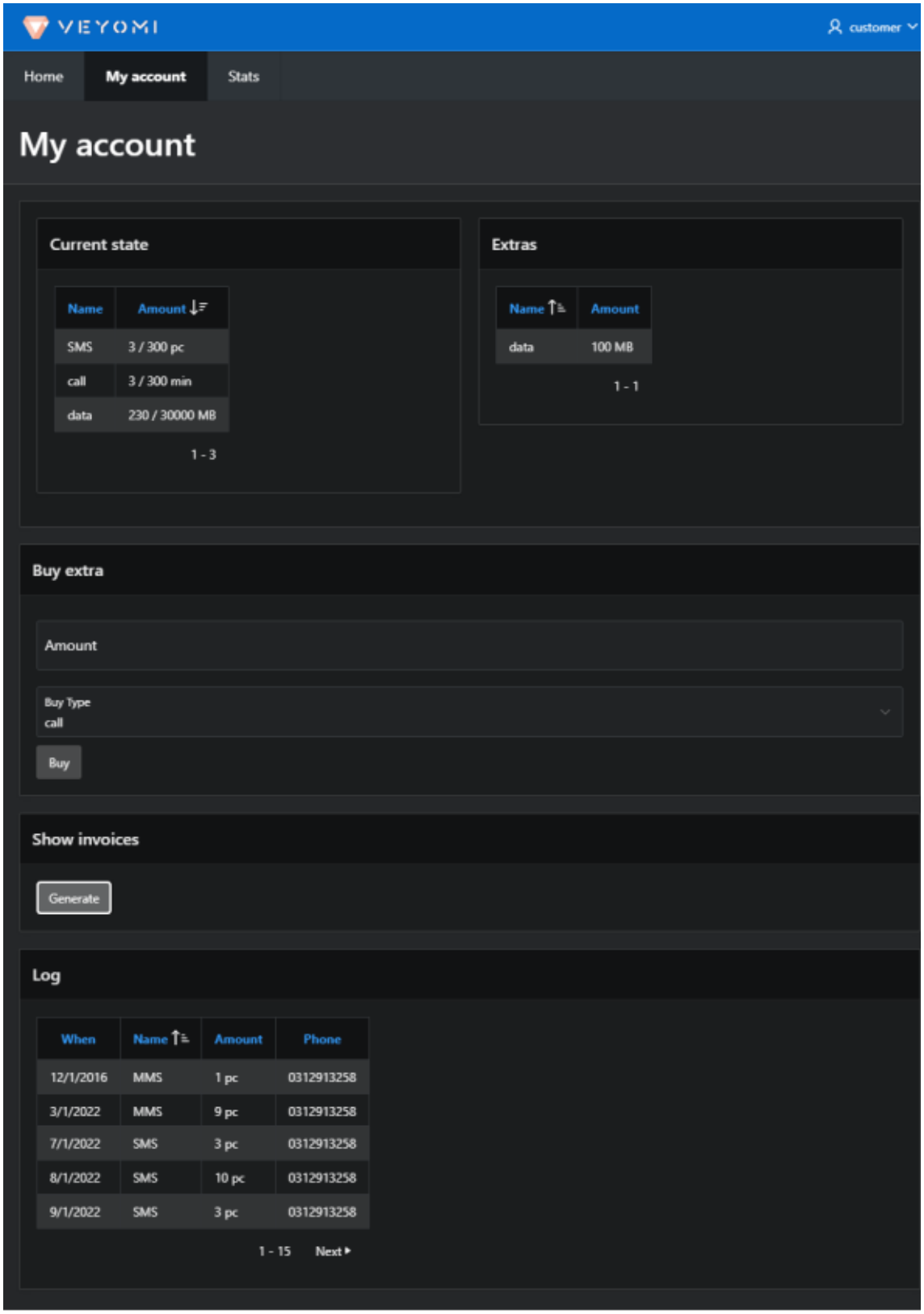


Figure 23.4: Customer dashboard.

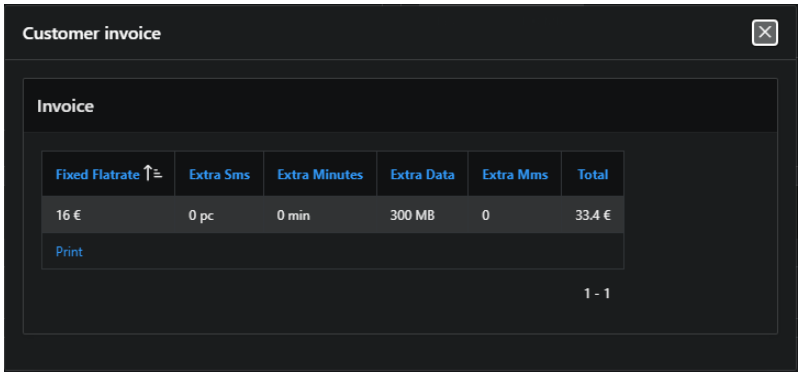


Figure 23.5: Customer dashboard – Invoice modal window.

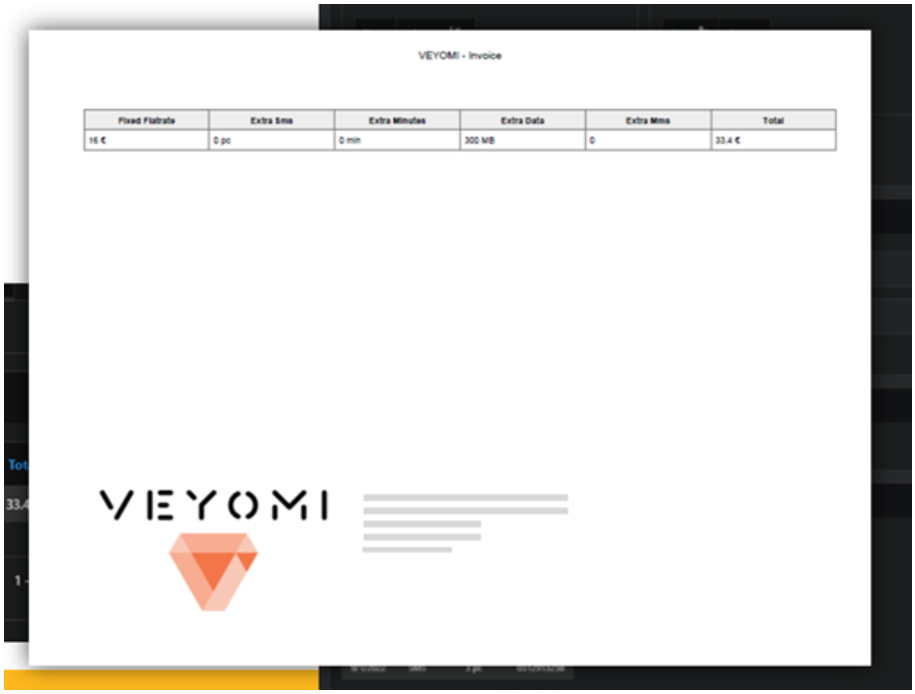


Figure 23.6: Customer dashboard – Invoice in PDF.

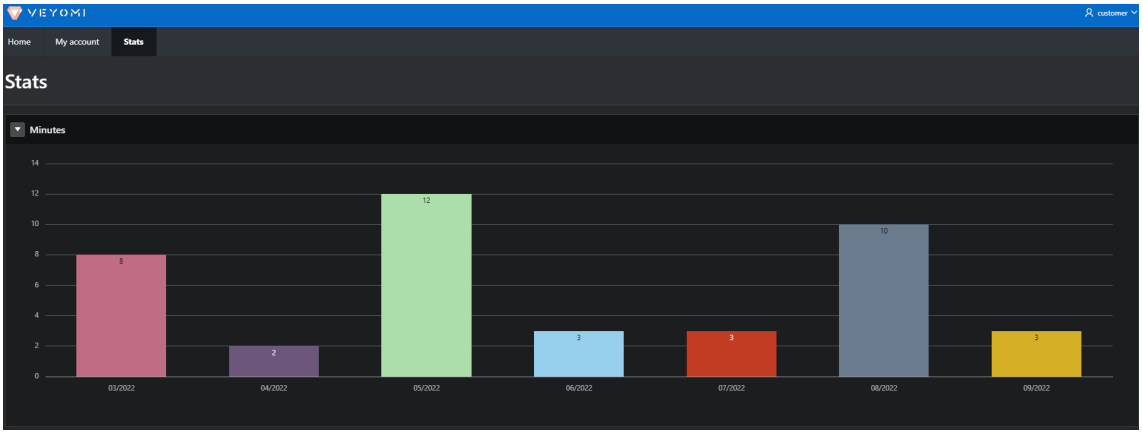


Figure 23.7: Customer dashboard – Stats of minutes.



Figure 23.8: Customer dashboard – Stats of SMS.

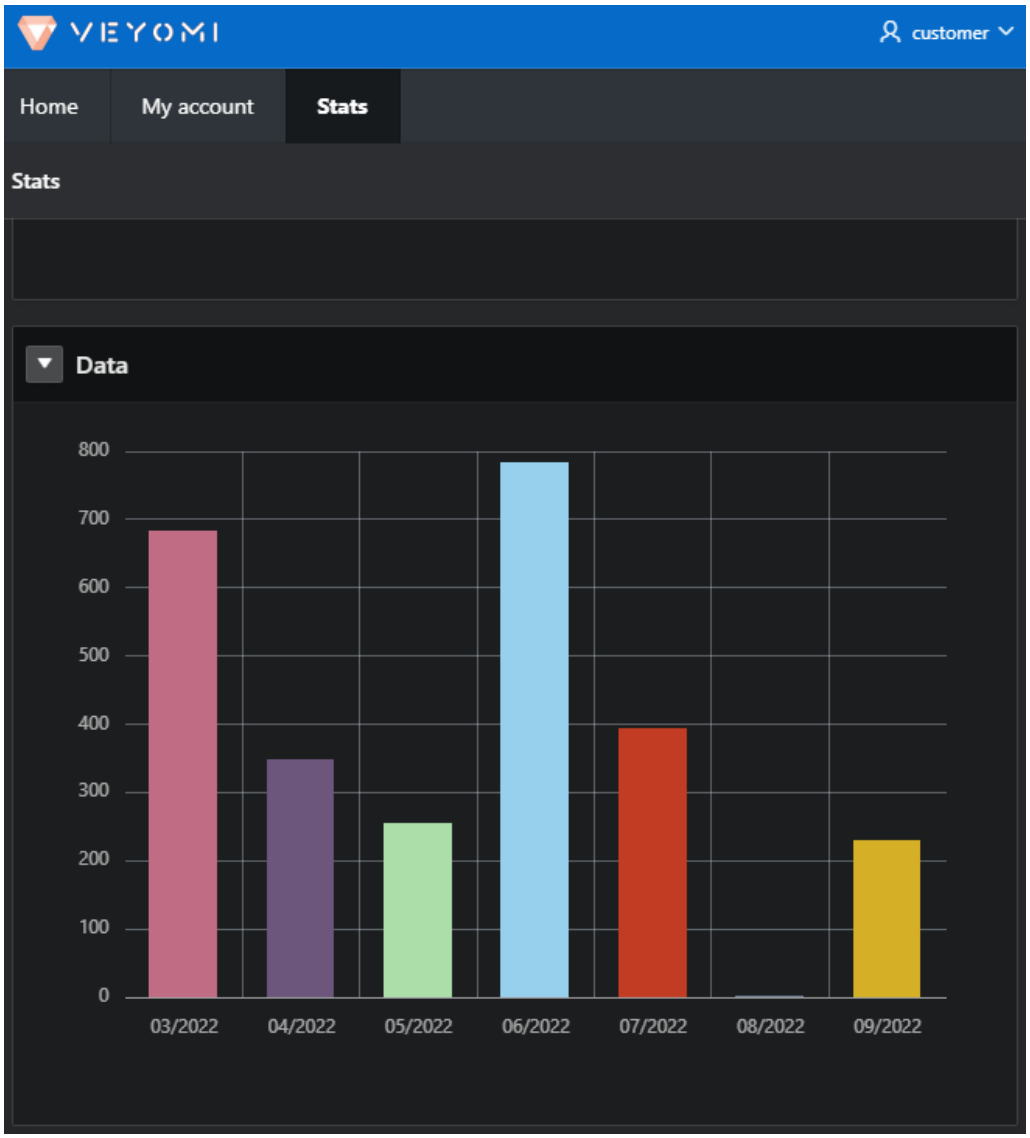


Figure 23.9: Customer dashboard – Stats of data.

After the manager logs in to the application, the functionality of managing customers is provided on separate tabs that are part of the manager's page. The first functionality is adding a new

customer, where the manager can add and create a new customer account. Next, we use the customer management component to edit customer information, which is designed in the data grid. This component also provides the functionality of simple reporting or exporting customer information to other third-party applications, if necessary. Finally, the customer display tab demonstrates a simplified form of displaying an overview of customer information. This grid is configured so that it does not allow editing of individual items, but it has the possibility of creating reports and exports from this list.

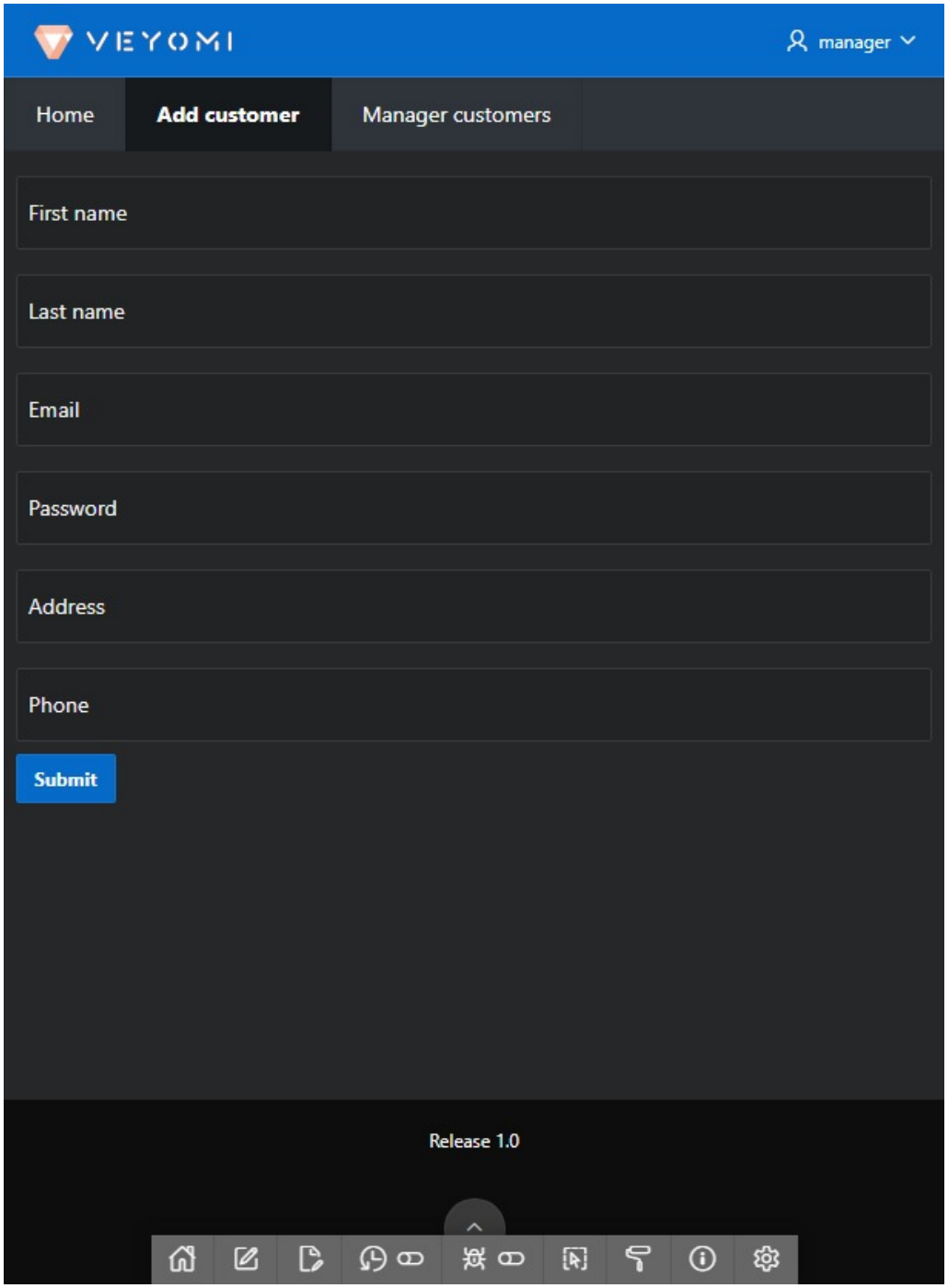


Figure 23.10: Manager dashboard – Add customer.

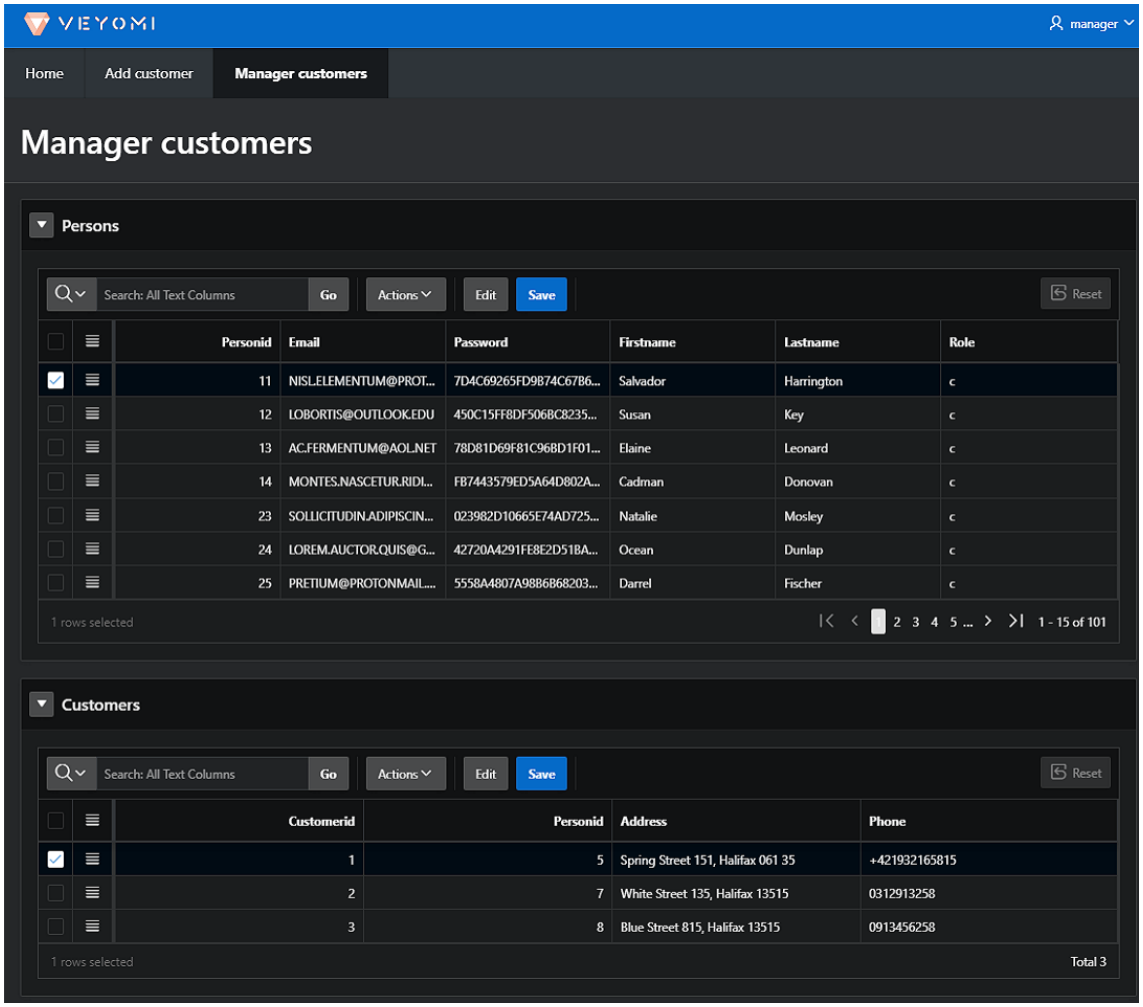


Figure 23.11: Manager dashboard – Manage customer.

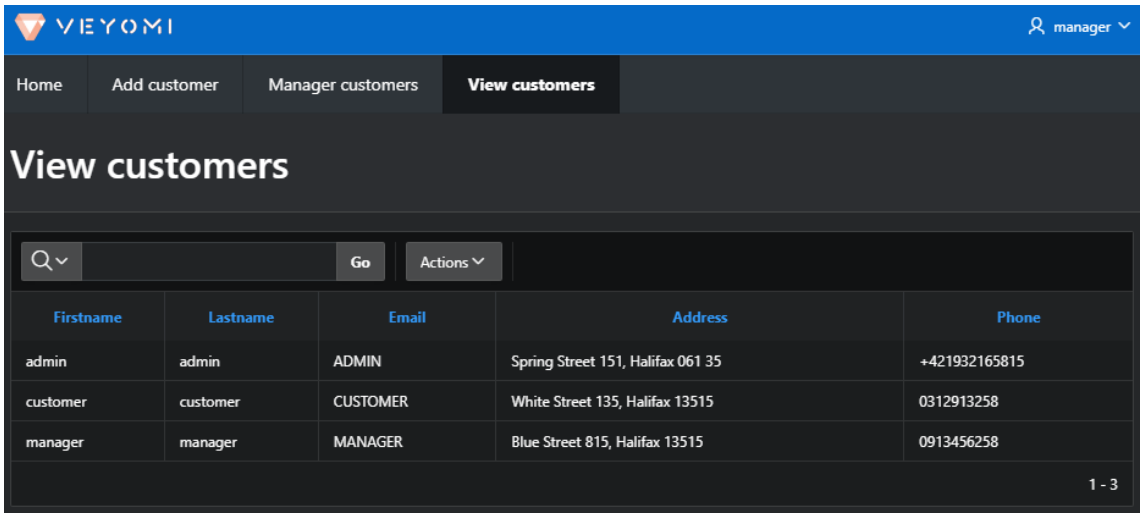


Figure 23.12: Manager dashboard – View customers.

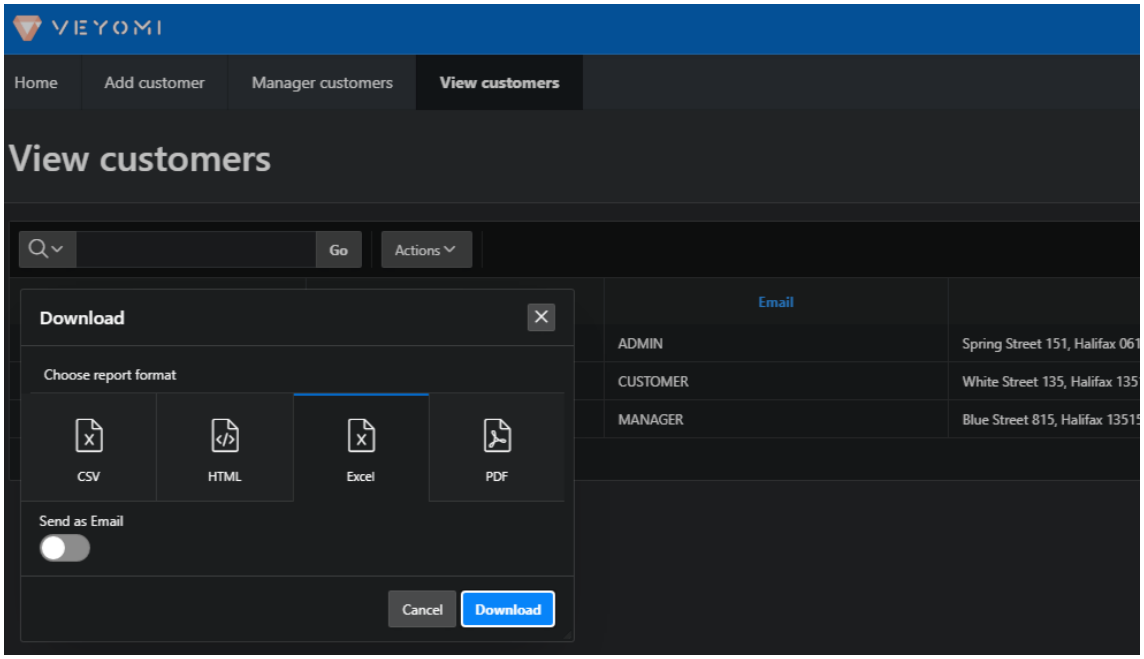


Figure 23.13: Manager dashboard – Customer export.

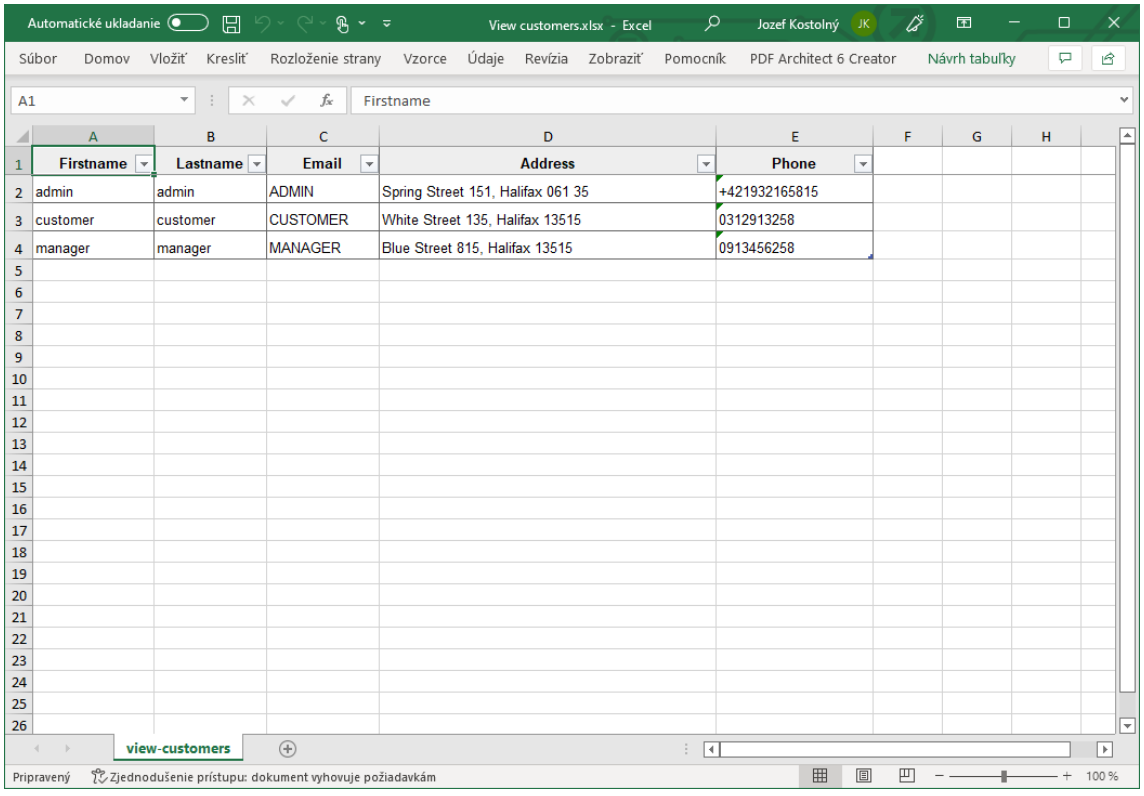


Figure 23.14: Manager dashboard – Customer export in XLS file.

The last page that needs to be created is the page for the application administrator. On this page, we provide the service of managing managers in an editable data grid, enabling export reports and filter functionality.

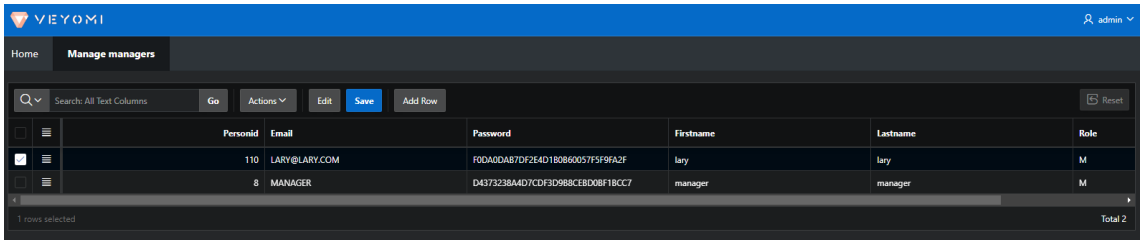


Figure 23.15: Administrator dashboard.

An important part is the creation of the login form. In our application, we are considering placing this form on a particular page, which is always called when entering secure pages, where it is necessary to authenticate as soon as possible.



Figure 23.16: A landing page with login.

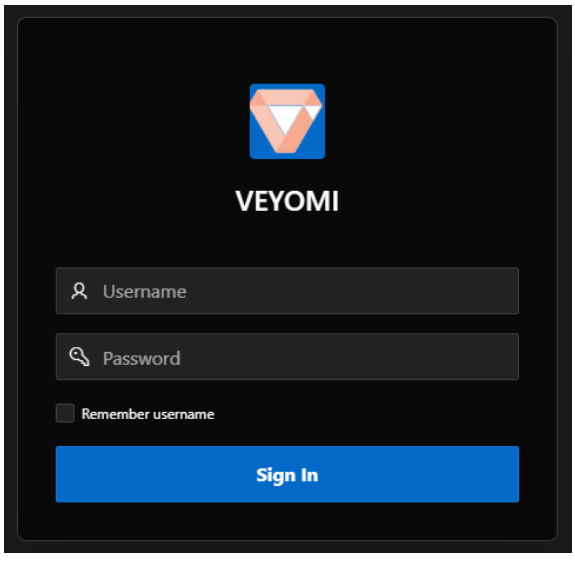


Figure 23.17: Login page.

23.7 Scripts

```
CREATE SEQUENCE "CUSTOMER_SEQ"
MINVALUE 4 MAXVALUE 9999999999999999999999999999
INCREMENT BY 1
START WITH 4
NOCACHE NOORDER NOCYCLE NOKEEP NOSCALE GLOBAL
/

CREATE OR REPLACE EDITIONABLE TRIGGER "BI_CUSTOMER"
BEFORE INSERT ON CH23_CUSTOMER
FOR EACH ROW
BEGIN
SELECT CUSTOMER_SEQ.nextval INTO :new.customerid
FROM dual;
END;
/

ALTER TRIGGER "BI_CUSTOMER" ENABLE;
/
```

Listing 23.4: Trigger on customer sequence.

```
CREATE SEQUENCE "PERSON_SEQ"
MINVALUE 1 MAXVALUE 9999999999999999999999999999
INCREMENT BY 1
START WITH 1
NOCACHE NOORDER NOCYCLE NOKEEP NOSCALE GLOBAL
/
CREATE OR REPLACE EDITIONABLE TRIGGER "BI_PERSON"
BEFORE INSERT ON CH23_PERSON
FOR EACH ROW
BEGIN
SELECT PERSON_SEQ.nextval INTO :new.personid
FROM dual;
:new.email := UPPER(:new.email);
:new.password := hash_password(UPPER(:new.email), :new.password);
END;
/

ALTER TRIGGER "BI_PERSON" ENABLE;
/

CREATE OR REPLACE EDITIONABLE TRIGGER "BU_PERSON"
BEFORE UPDATE ON CH23_PERSON
FOR EACH ROW
BEGIN
:new.email := UPPER(:new.email);
IF :new.password IS NOT NULL THEN
:new.password := hash_password(UPPER(:new.email), :new.password);
ELSE
```

```
:new.password := :old.password;
END IF;
END;
/
ALTER TRIGGER "BU_PERSON" ENABLE;
/
```

Listing 23.5: Trigger on inserting a person.

```
CREATE OR REPLACE FUNCTION AUTHENTICATE_USER
(p_username in varchar2,
p_password in varchar2)
 return boolean
IS
l_user_name          CH23_PERSON.email%type := UPPER(p_username);
l_password           CH23_PERSON.password%type;
l_hashed_password varchar2(1000);
l_count              number;
BEGIN
SELECT COUNT(*) INTO l_count
FROM CH23_PERSON
WHERE email = l_user_name;

IF l_count > 0 THEN
l_hashed_password := hash_password(l_user_name, p_password);
SELECT password INTO l_password
FROM CH23_PERSON
WHERE email = l_user_name;

IF l_hashed_password = l_password THEN
APEX_UTIL.SET_AUTHENTICATION_RESULT(0);
 return true;
ELSE
APEX_UTIL.SET_AUTHENTICATION_RESULT(4);
 return false;
END IF;

ELSE
APEX_UTIL.SET_AUTHENTICATION_RESULT(1);
 return false;
END IF;
APEX_UTIL.SET_AUTHENTICATION_RESULT(7);
 return false;
EXCEPTION
WHEN OTHERS THEN
APEX_UTIL.SET_AUTHENTICATION_RESULT(7);
APEX_UTIL.SET_CUSTOM_AUTH_STATUS(sqlerrm);
 return false;
END AUTHENTICATE_USER;
/
```

Listing 23.6: Function AUTHENTICATE_USER.

23.8 Creating a home page

In Create a Page, you can choose from various templates, a blank page, or feature pages with predefined functionalities, such as Login, About page, and Configuration Options. First, you need to fill in basic information, such as a page number and name and choose a page model, such as Normal, Modal Dialog, or Drawer. You can also fill in additional information, such as setting a navigation entry for the page (whether it will be a top entry or nested in an existing entry) and choosing an icon shown in a dashboard menu.

After clicking on Create Page, the dashboard for editing information and content of the page is displayed. The main content of the page (Body) is divided into regions.

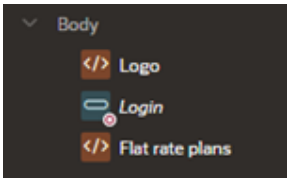


Figure 23.18: Regions of the body.

Each region can contain HTML or various contents, such as static HTML or predefined content.

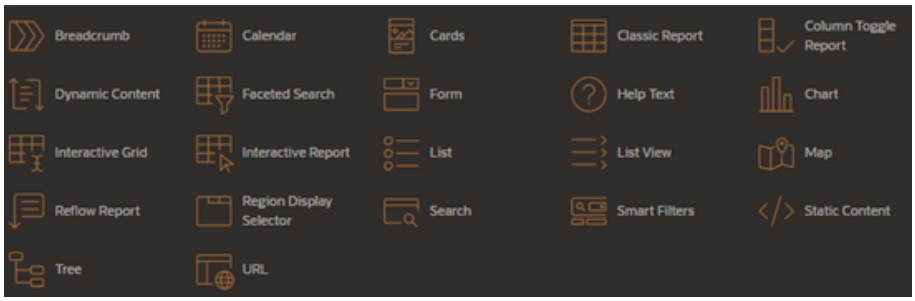


Figure 23.19: List of content.

Our **Login** region contains a button to redirect a page into the user´s login form. In the Behaviour property of the button, we have defined Action, which was set as Redirect to Page in this Application. Then we set Target to a predefined login page. Since it is not necessary to show this button when the user is already logged in, we need to hide the button by setting the Type as User is the Public User (user has not authenticated). Finally, we created **Flat rate plans** as a static content region containing HTML code, for which we can define CSS.

```
</head>
<body>
<div id="container">
    <div class="whole">
    <div class="type">
        <p>VEYOMI 100+</p>
        </div>
    <div class="plan">

        <div class="header">
            11<span>€</span>
            <br>
        </div>
        <div class="content">
            <ul>
                <li>100 minutes calls</li>
                <li>100 SMS/MMS</li>
                <li>10 GB data</li>
            </ul>

        </div>

        <div class="price">
```

Figure 23.20: HTML code of a static region.

23.9 Creating a customer page

We created this page using similar steps as when creating a home page. When a page is created, we continue to develop a layout of a customer´s page, which is divided into several regions, such as for displaying the current state of services, buying extras, showing invoices, and displaying logs.

The **Current state** region consists of two subregions, which display numerical data – the current price of consumed units of flat rate and extras. The SQL query is used to obtain these data.

```
SELECT name, sum(u.amount) || ' ' || ' / ' || fs.amount || ' ' || s.units AS
    amount
FROM CH23_USAGELOG u
JOIN CH23_CUSTOMERFLATRATE cf ON (u.customerid = cf.customerid AND u.
    flatrateid = cf.flatrateid)
JOIN CH23_CUSTOMER c ON (cf.customerid = c.customerid)
JOIN CH23_PERSON p ON (c.personid = p.personid)
JOIN CH23_SERVICE s ON (u.serviceid = s.serviceid)
JOIN CH23_FLATRATESERVICE fs ON (fs.flatrateid = cf.flatrateid AND fs.serviceid = s
    .serviceid )
WHERE email = :APP_USER
AND EXTRACT(month from when) = EXTRACT(month from sysdate)
GROUP BY s.serviceid, s.name, s.units, fs.amount;
```

Listing 23.7: SQL Query - Consumed units of a flat rate.

```
SELECT name, SUM(amount) || ' ' || units AS amount
FROM CH23_EXTRASERVICE JOIN CH23_CUSTOMER USING(customerid)
JOIN CH23_PERSON USING (personid)
JOIN CH23_SERVICE USING (serviceid)
WHERE email = :APP_USER
AND EXTRACT(month from startDate) = EXTRACT(month from sysdate)
GROUP BY serviceid, name, units;
```

Listing 23.8: SQL Query - Consumed units of extra services.

The **Buy extra** region contains a number field, a select list, and a button. It is possible to set an amount and to select which extra service we want to buy. It is operated using a button that executes a process running a PL/SQL code:

```
INSERT INTO CH23_EXTRASERVICE VALUES (
:BUY_TYPE,
(SELECT customerid
FROM CH23_CUSTOMER JOIN CH23_PERSON USING (personid)
WHERE email = :APP_USER),
sysdate,
:BUY_AMOUNT
);
```

Listing 23.9: PL/SQL code for inserting an extra service.

The **Show invoices** region consists of a button used for generating invoices. In the *Behaviour* property of the button, we have defined *Action*, which was set as *Redirect to Page in this Application*. Then we set *Target* to a predefined customer invoice page, which was set to be modal. On this page, a classical report, which can also be printed, is displayed using an SQL query:

```
SELECT f.price || ' €' as "Fixed Flatrate ",
SUM(CASE e.serviceid WHEN 1 THEN NVL(e.amount, 0) ELSE 0 END)
|| ' ' || listagg (CASE s.serviceid WHEN 1 THEN s.units ELSE '' END, '') AS "Extra
    SMS",
SUM(CASE e.serviceid WHEN 2 THEN NVL(e.amount, 0) ELSE 0 END)
|| ' ' || listagg (CASE s.serviceid WHEN 2 THEN s.units ELSE '' END, '') AS "Extra
    Minutes",
SUM(CASE e.serviceid WHEN 3 THEN NVL(e.amount, 0) ELSE 0 END)
|| ' ' || listagg (CASE s.serviceid WHEN 3 THEN s.units ELSE '' END, '') AS "Extra
    Data",
SUM(CASE e.serviceid WHEN 4 THEN NVL(e.amount, 0) ELSE 0 END)
|| ' ' || listagg (CASE s.serviceid WHEN 4 THEN s.units ELSE '' END, '') AS "Extra
    MMS",
(f.price
+ SUM(CASE e.serviceid WHEN 1 THEN NVL(e.amount, 0) * s.price ELSE 0 END)
+ SUM(CASE e.serviceid WHEN 2 THEN NVL(e.amount, 0) * s.price ELSE 0 END)
+ SUM(CASE e.serviceid WHEN 3 THEN NVL(e.amount, 0) * s.price ELSE 0 END)
+ SUM(CASE e.serviceid WHEN 4 THEN NVL(e.amount, 0) * s.price ELSE 0 END)
|| ' €') AS "Total"
FROM CH23_CUSTOMER c
JOIN CH23_PERSON p ON (p.personid = c.personid)
JOIN CH23_CUSTOMERFLATRATE cf ON (cf.customerid = c.customerid)
JOIN CH23_FLATRATE f ON (f.flatrateid = cf. flatrateid )
JOIN CH23_FLATRATESERVICE fs ON (fs.flatrateid = f.flatrateid)
JOIN CH23_SERVICE s ON (s.serviceid = fs.serviceid)
LEFT JOIN CH23_EXTRASERVICE e ON (e.customerid = cf.customerid)
WHERE p.email = :APP_USER
AND (e.startdate IS NULL OR
(EXTRACT(month from e.startdate) = EXTRACT(month from sysdate)
AND EXTRACT(year from e.startdate) = EXTRACT(year from sysdate)))
GROUP BY f.price;
```

Listing 23.10: SQL Query for an invoice report.

The **Log** region contains an interactive report for displaying a usage log by using SQL query:

```
SELECT name, phone, amount || ' ' || units AS amount, when
FROM CH23_USAGELOG JOIN CH23_CUSTOMER USING (customerid)
JOIN CH23_PERSON USING (personid)
JOIN CH23_SERVICE USING (serviceid)
WHERE email = :APP_USER
ORDER BY when DESC;
```

Listing 23.11: SQL Query – Usage log.

Tab **Stats** is a new page consisting of three regions, such as Minutes, SMS, and Data. The region type is set to be *Static Content*, including *Chart* components displaying charts from series. Its source is defined by SQL query. By default, the colour of the chart bars is set to be uniform. It can be adjusted in SQL query:

```
SELECT TO_CHAR(when, 'MM/YYYY') AS "Month",
SUM(amount) AS Amount,
CASE WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '01'
THEN '#F8B195'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '02'
THEN '#F67280'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '03'
THEN '#C06C84'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '04'
THEN '#6C567B'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '05'
THEN '#ACDEAA'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '06'
THEN '#97D0ED'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '07'
THEN '#C23B23'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '08'
THEN '#6B7B8E'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '09'
THEN '#D5AF26'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '10'
THEN '#9AB5C1'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '11'
THEN '#74698C'
WHEN SUBSTR(TO_CHAR(when, 'MM/YYYY'), 0, 2) = '12'
THEN '#C1867B'
ELSE 'black'
END AS MY_COLOR
FROM CH23_USAGELOG
JOIN CH23_CUSTOMER USING (customerid)
JOIN CH23_PERSON USING (personid)
JOIN CH23_SERVICE USING (serviceid)
WHERE email = :APP_USER AND serviceid = 2
GROUP BY TO_CHAR(when, 'MM/YYYY')
ORDER BY 1;
```

Listing 23.12: SQL Query - Statistics.

23.10 Creating a manager page

In the manager page, we used similar methods and components to create subpages. Each subpage is created as a new page, such as Add customer, Manage customers, and View customers.

Page *Add customer* contains a simple form that is created from components. The process of adding a new customer is executed by PL/SQL code:

```
INSERT INTO CH23_PERSON (EMAIL, PASSWORD, FIRSTNAME, LASTNAME,
    ROLE)
VALUES (:CUSTOMER_EMAIL, :CUSTOMER_PASSWORD, :
    CUSTOMER_FIRSTNAME,
:CUSTOMER_LASTNAME, 'C');
```

Listing 23.13: PL/SQL code for inserting a person.

Page *Manage customers* is created with two interactive grids: Manage persons and Manage current customers. Interactive grids are connected to the process of editing and saving changes. The type of the process is set to be *Interactive Grid – Automatic Row Processing (DML)*.

Page *View customers* contains an Interactive Report to display data without editing them. This report is created using SQL query:

```
SELECT firstname, lastname, email, address, phone
FROM CH23_PERSON
JOIN CH23_CUSTOMER USING (personid);
```

Listing 23.14: SQL Query - Customers.

23.11 Supplementary learning material

You can find the following supplementary learning material:

- script for creating and populating tables
- script for dropping tables
- exported packaged application
- video which demonstrate how to generate application

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter23 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

23.11.1 Exported application

Exported application is packaged. Installation creates tables as well it populates data. De-installation removes all data base objects used in this application.

Packaged application is tested and it will run in new workspace if the following requirements are meet:

- add APEX user before running application. Only in development and testing workspace navigate to Shared Components > Application Access Control > Add User Role Assignment; enter APEX user and set this user roles Administrator, Contributor and Reader. In production consultation with skilled personnel before deployment in a must.

If user is not granted appropriate role than imported application will crash. It is necessary to clear web browser cookie (i.e. Firefox: Settings > Cookies and Site Data > Manage Data) after application crashes due to unmet requirements.

23.11.2 Video guides

Video guide shows all steps in application development.

23.12 Questions

1. Which subpage is used for creating a new login scheme?
2. Name at least two authentication methods for user management.
3. Throughout the implementation, various items were referenced in the Select statements. How can you identify those items?

23.13 Answers

1. In the Oracle APEX tool, we choose the authentication method - Authentication Scheme - in the Shared Components => Authentication Schemes sub page.

2. There are multiple methods, which can be used for user authentication, like LDAP server, Database account, Oracle APEX account, etc.
3. Items in the Select statements are referenced using a colon at the beginning of the item name.



24. Car rental case

A. Angeioplastis, G. Myllis, A. Tsimpiris, D. Varsamis

24.1 Business view of the case

Renting a car can be a convenient way to explore new destinations, but there are some potential pitfalls to be aware of. It's important to understand that car rental is more similar to booking a hotel room than booking an airline ticket. When renting a car, there are two types of charges: those you pay when you pick up the car, and prepaid prices.

For first-time renters, it's usually best to book a Pay Later rate, which allows for greater flexibility in case you need to change your plans. This type of rental rate does not typically have a cancellation penalty, which can be helpful if unforeseen circumstances arise.

To avoid additional charges, it's important to carefully review the rental agreement and any additional fees that may be incurred, such as insurance or additional driver fees. Additionally, be sure to inspect the vehicle carefully before leaving the rental lot and report any damage to the rental company immediately to avoid being held responsible for it later.

By understanding the rental process and potential fees, you can ensure a smooth and stress-free rental experience and fully enjoy the freedom and flexibility that renting a car can offer while traveling.

24.2 Problem definition

Renting a car may seem daunting with all the rules and conditions, but they exist for a good reason. A brand new car costs an average of about 36,000 EURO, so when you rent a car, you're essentially paying a small fraction of its overall value for the temporary use of the vehicle.

While it's crucial to have adequate car rental insurance in case of an accident or unforeseen circumstances, most rental experiences are incident-free. However, it's still important to carefully read the rental agreement, understand the terms and conditions, and inspect the vehicle for any damage before accepting it to avoid any potential issues.

By following these guidelines, you can have a safe and enjoyable rental experience without any unexpected surprises. And remember, the rules and conditions are in place to protect both you and the rental car company, so it's always best to adhere to them for a smooth rental experience.

24.3 Use cases

24.3.1 Narrative description of use case

The goal of this project is to develop a functional application that simulates the operation of a car rental shop. The application will utilize the Oracle APEX platform to manage and store customer, car, and rental information in detail.

The application is designed to be user-friendly for the owner of a car rental shop, allowing them to manage all necessary information, including rentals, available cars for rent, and customer details. Users will be able to make rentals by selecting a car based on criteria such as make, model, year, and fuel type, and all necessary details required to complete the rental will be displayed.

Additionally, the application will display data such as current rentals, available cars for rent, and customer details in detail. The current rentals will be displayed to the user in tables, depending on which tab the user is in.

Overall, this application will provide an efficient and organized system for managing a car rental shop, with detailed information available at the click of a button. The user-friendly interface and comprehensive data management tools will make running a car rental shop simpler and more efficient.

24.3.2 Semi-structured description

This project aims to develop of a user-friendly application that utilizes Oracle APEX to manage detailed information on customers, cars, and rentals for a car rental shop. The application will enable users to make rentals based on specific criteria, display necessary rental information, and provide comprehensive data management tools to display current rentals, available cars for rent, and customer details. By providing an efficient and organized system for managing a car rental shop, this application will make running a car rental shop simpler and more efficient. The semi-structure description is provided in Table 24.1.

24.3.3 Use case diagram

The above story is depicted on use case diagram (see Figure 24.1).

Table 24.1: Use case description: accessing cars, customers and car rent reservation

Keyword	**Value**
ID:	*Ch24-01*
Title:	*Access to list of cars and costumers, ability to add and remove data and create a new car rental reservation by selecting preferred dates and calculating the cost.*
Description:	*The user will utilize the APEX application to access and view the cars that belong to the company, as well as all of the customers who have previously made a booking. Both cars and customers are described by a set of attributes. Additionally, the user will have the ability to create, update, and delete data for both categories, allowing for efficient management of the car rental system. Furthermore the user is able to create a new car rental reservation by adding the car, customer, and dates. The system calculates the total cost of the reservation based on the selected dates and updates the results accordingly. The final report presents the reservation details, including the car, customer, rental dates, and total cost.*
Primary Actor:	*User administrator*
Preconditions:	User has an admin account for APEX instance.
Post conditions:	After updating and selecting a car and customer, the user will be able to create a car rental reservation within the APEX instance. After completing a car rental reservation and calculating the total cost.
Main	Scenario
Success Scenario:	1. Open a web browser and sign in to the car rental application. 2. Select the menu item or page navigation labeled **Cars** or **costumers**. 3. To add a car, click on the "Create" button, and for customers, do the same. 4. To edit a car or customer entry, click on the pencil symbol associated with that specific entry. 5. To add or update data, click the "Apply Changes" button. To delete data, click on the "Delete" button. 6. Select menu item or page navigation **rent car**. 7. To create a new rent click **create**. 8. To add car to the reservation click on **carid** and select from the list of available cars. 9. To add costumer to the car reservation click on **custid** and select from the list. 10. To set the dates of the reservation click on **rent date** and select the preferred start date of the reservation. 11. Then click on the **return date** field and select the date when the car rental will end. 12. When the car is returned, the total cost of the reservation is calculated based on the number of days rented and any additional charges. If there is damage to the car, an extra cost will be added to cover the repair costs. 13. The final cost is presented to the customer along with a detailed breakdown of charges in the rental report.
Extensions:	*None*
Frequency of Use:	*The number of reservations per day varies depending on the tourist season, but on average, we receive approximately five reservations per day.*
Status:	Finished
Owner:	User
Priority:	*moderate*

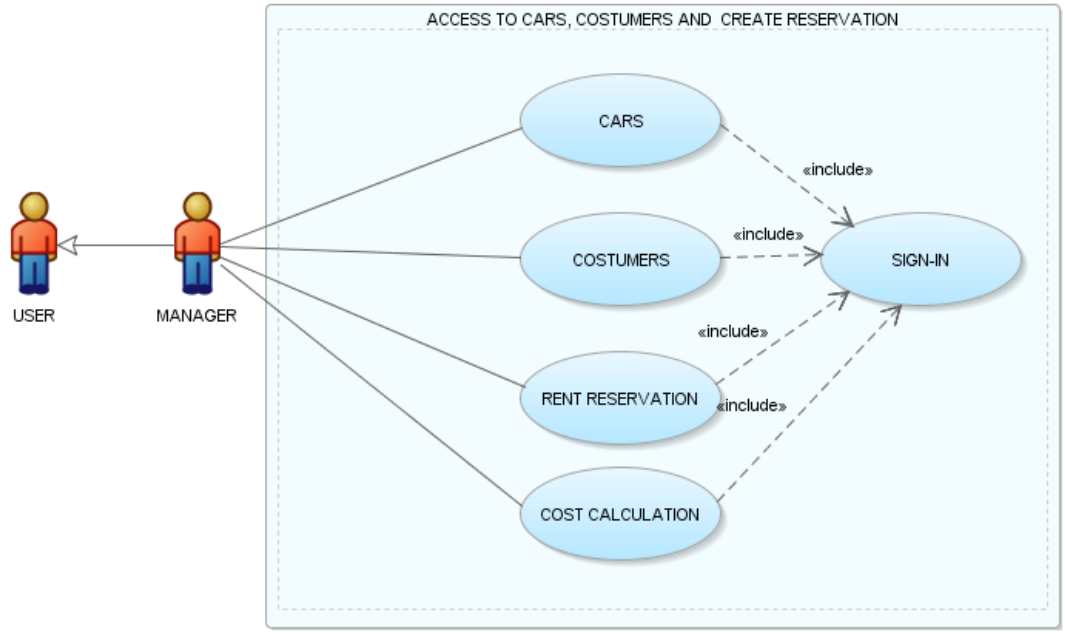


Figure 24.1: Use case diagram.

24.4 Data model

24.4.1 Narrative description of data model

The educational project is implemented using three tables to represent the entities involved: the cars table, the customers table, and the car rental table. These tables contain all the necessary attributes and data needed for managing the car rental process through the APEX application.

1. Cars Table
   - **id** identity number and primary key,
   - **make** car manufacturer
   - **carmodel** the model of the car,
   - **fuel** type benzine, diesel, LNG or electric
   - **doors** how many doors the car have
   - **color** the color of the car
   - **rent** price the cost of a daily rental of the specific car
   - **carphoto** a photo of the car
2. Customers Table
   - **id** identity the number and primary key of the customer,
   - **fname** the first name of the customer,
   - **lname** the last name of the customer,
   - **address** the address of the customer,
   - **zipcode**the zip code of the address,
   - **city** the city of the customer,
   - **cardid** the id license or id card of the customer
   - **telephone-work** a telephone of customers work
   - **mobile** mobile phone
   - **email** the email of the customer
   - **comments** comments related with the behavior of the person as a customers
3. CarRental Table

- **id** identity the number and primary key of the rental table
- **carid id** connection field between car table and rental table
- **custid id** connection field between customer table and rental table
- **rent date** rental date
- **return date** return date,
- **damage** description of a damage of the car if it happens
- **damage cost** the damage cost
- **total cost** the total cost of the rental

24.4.2 Logical data model

Logical data model as shown on Figure 24.2.

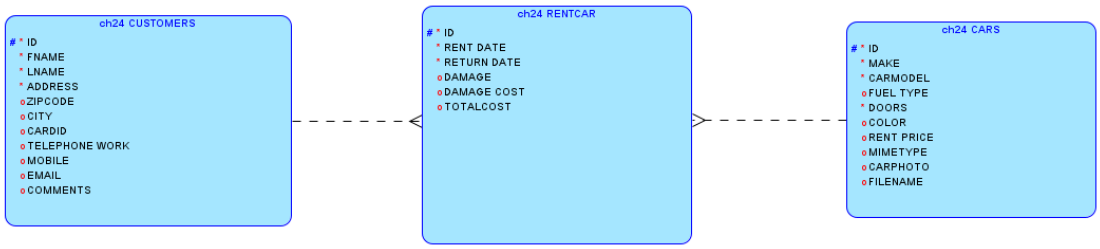


Figure 24.2: Logical model of the Car Rental Project

24.4.3 Relational data model

Relational data model as shown on Figure 24.3.

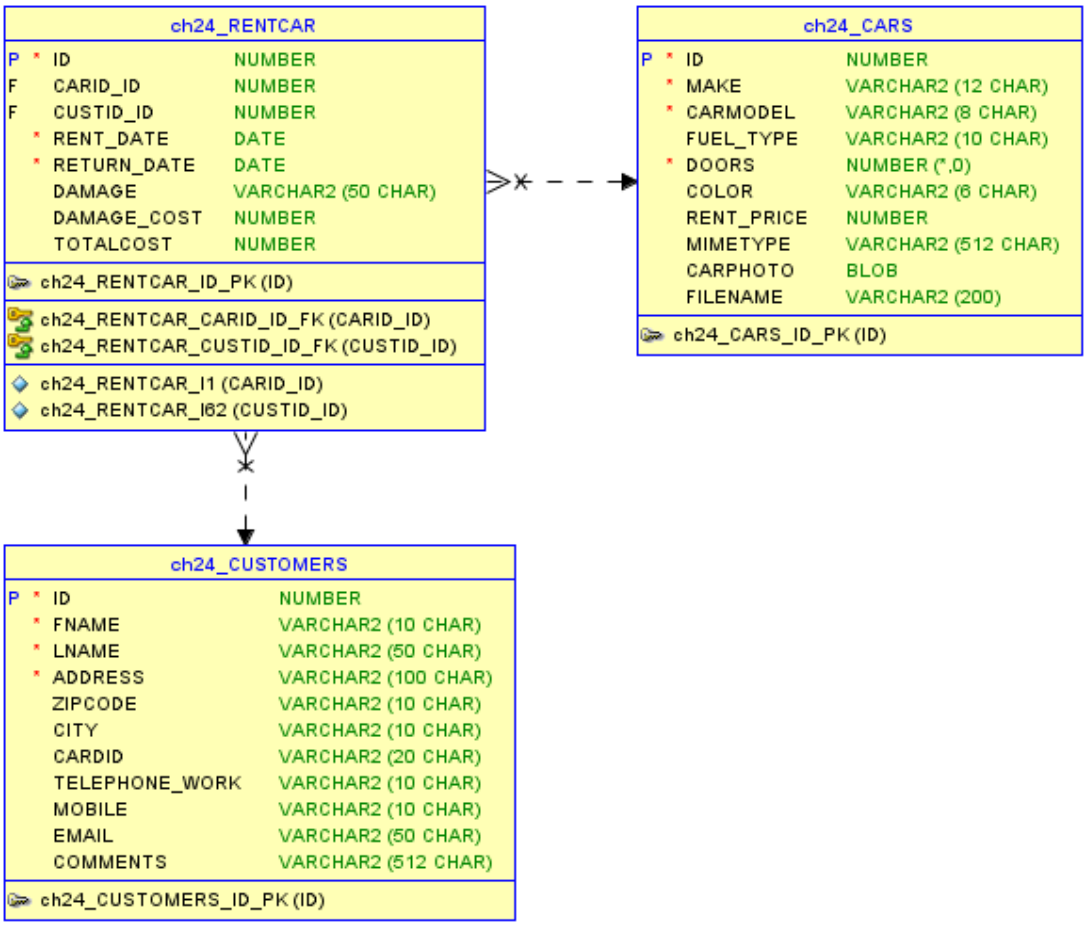


Figure 24.3: Relational Model of the Car Rental Project

24.4.4 SQL Script

The tables we have created for our application are CH24_CARS, CH24_CUSTOMERS and CH24_RENTCAR.The fields of each table are described below:

```
drop table  ch24_cars;
drop table  ch24_customers;
drop table  ch24_rentcar;
-- create tables
create table ch24_cars (
    id                    number generated by default on null as identity
                          constraint ch24_cars_id_pk primary key,
    make                  varchar2(12 char) not null,
    carmodel              varchar2(8 char) not null,
    fuel_type             varchar2(10 char),
    doors                 integer not null,
    color                 varchar2(6 char),
    rent_price            number default '100',
    carphoto              varchar2(512 char)
)
;
```

```
create table ch24_customers (
    id                  number generated by default on null as identity
                        constraint ch24_customers_id_pk primary key,
    fname               varchar2(10 char) not null,
    lname               varchar2(50 char) not null,
    address             varchar2(100 char) not null,
    zipcode             varchar2(10 char),
    city                varchar2(10 char) default 'SERRES',
    cardid              varchar2(20 char),
    telephone_work      varchar2(10 char),
    mobile              varchar2(10 char),
    email               varchar2(50 char),
    comments            varchar2(512 char)
)
;
create table ch24_rentcar (
    id                  number generated by default on null as identity
                        constraint ch24_rentcar_id_pk primary key,
    carid_id            number
                        constraint ch24_rentcar_carid_id_fk
                        references ch24_cars on delete cascade,
    custid_id           number
                        constraint ch24_rentcar_custid_id_fk
                        references ch24_customers on delete cascade,
    rent_date           date not null,
    return_date         date not null,
    damage              varchar2(50 char),
    damage_cost         number,
    totalcost           number
)
;

-- table index
create index ch24_rentcar_i1 on ch24_rentcar (carid_id);
create index ch24_rentcar_i62 on ch24_rentcar (custid_id);
```

24.5 Application interfaces

The user, who is the owner of the car rental business, needs to provide their credentials to access the application. The application is using Application Express Accounts authentication. The owner's credentials are the same as the WORKSPACE credentials shown on Figure 24.4.

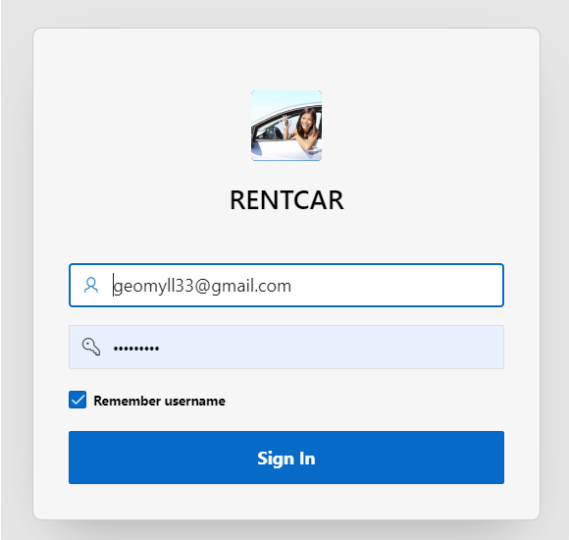


Figure 24.4: Log in to the app.

The application simulates the operation of a car rental shop and includes customer, car, and rental management. It provides detailed information on these entities using a friendly web-based application based on the Oracle APEX platform, as shown on Figure 24.5.

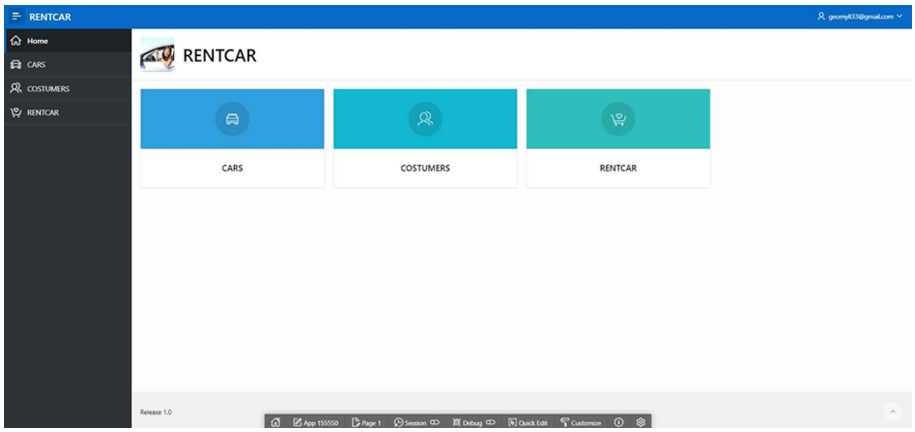


Figure 24.5: Home page of the app.

The template on Figure 24.6 is designed for managing the information related to cars in a car rental shop. The user, who is the owner of the shop, can store and manage all the necessary information about the cars, such as make, model, year, fuel type, and also upload a photo of the car to show its condition.

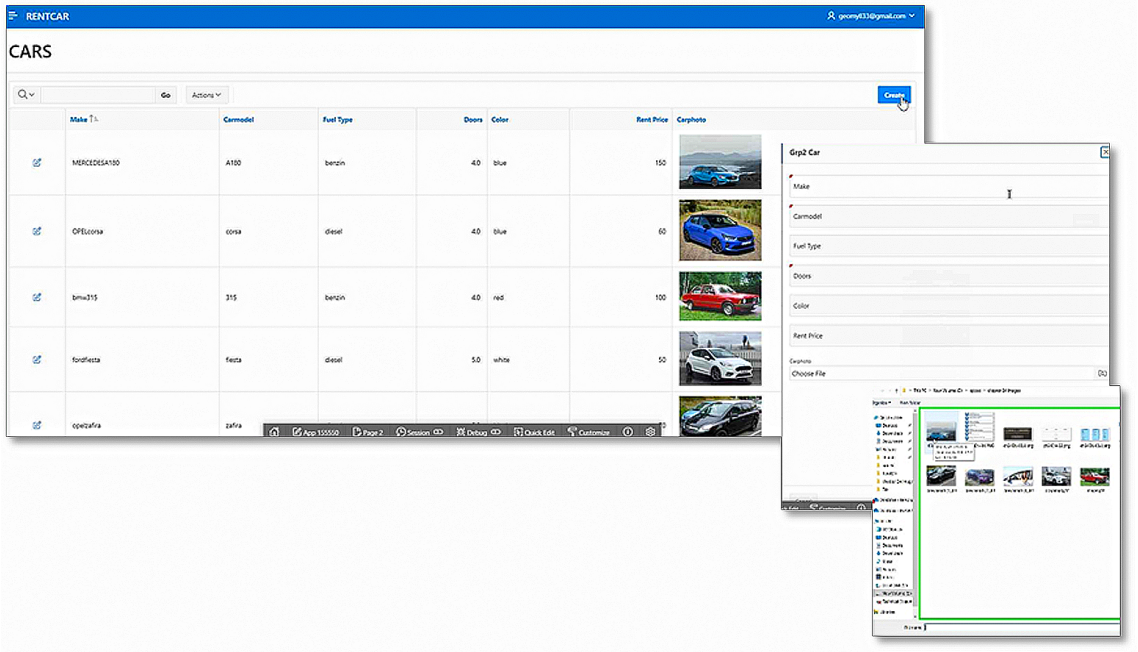


Figure 24.6: Cars template

On the following Figure 24.7, it is explained how to add a column to your car form that will allow showing photos of the cars. To achieve this, you need two auxiliary columns to save the information in your table, one for the mime and one for the file name. The column that will store the photo should be set to storage type BLOB. In our case, the column name is **CARPHOTO**.

In the application page designer, you need to go to the report for the CAR template and configure the **CARPHOTO** column as a file browser. Then change the settings of the storage type to "BLOB column specifier in item source attributes".

Next, you need to go to the form for the CAR template and configure the **CARPHOTO** column as a Display image. Configure the BLOB attributes by connecting the table **GRP2-CARS**, the column **CARPHOTO**, and the primary key **ID**. Finally, the columns **MIMETYPE** and **FILENAME** should be set to hidden.

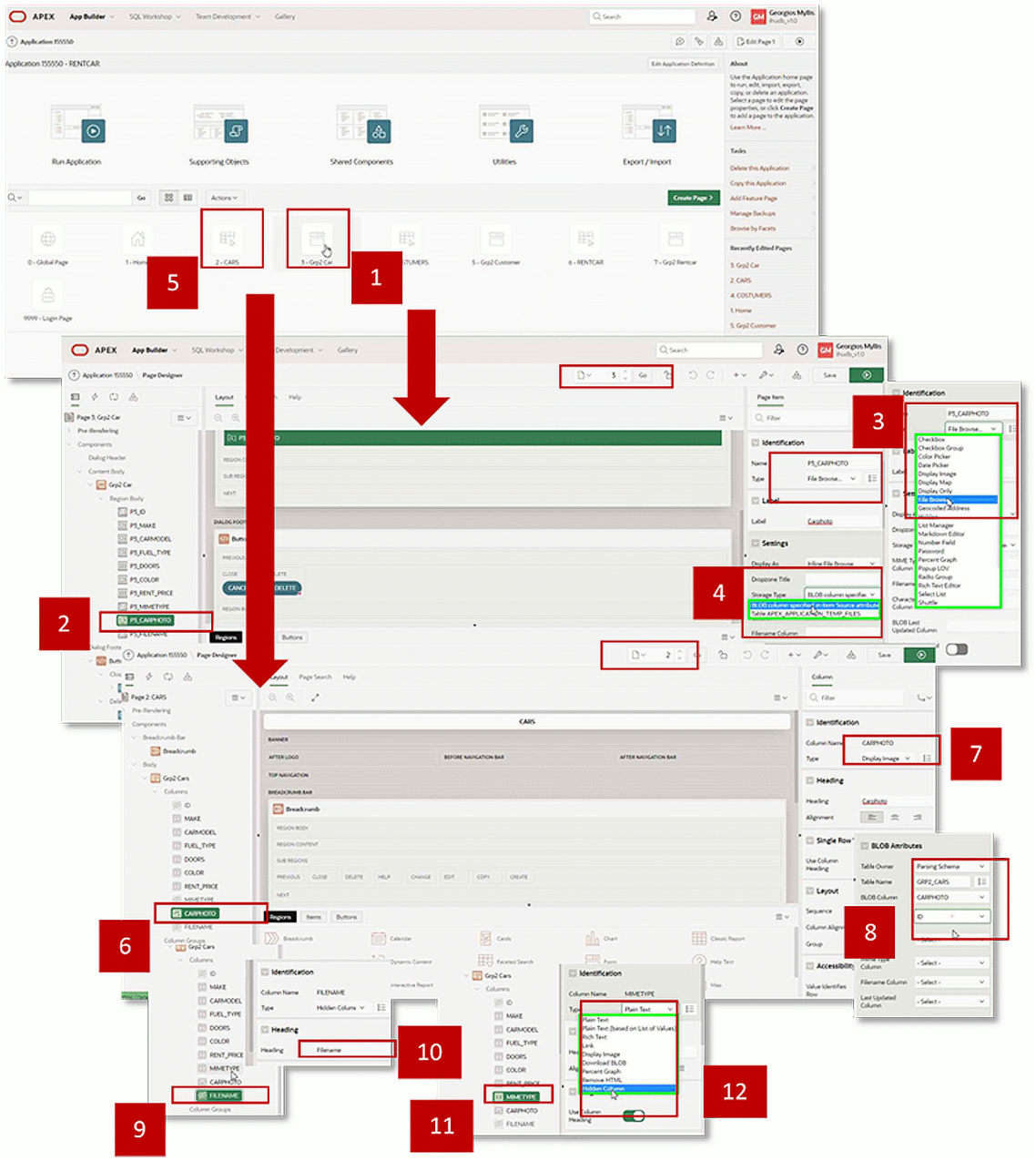


Figure 24.7: How to make a column with photos.

Similarly, data related to customers can be managed (see Figure 24.8).

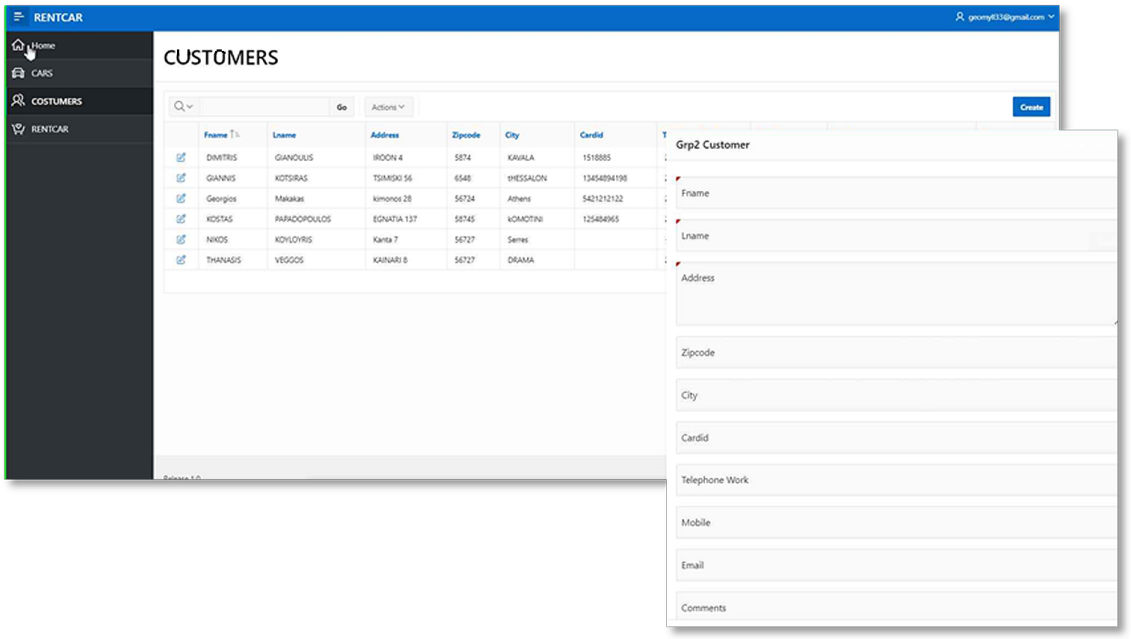


Figure 24.8: Customers data.

In the rental table as shown on Figure 24.9, the user can select a car based on criteria such as maker, model, and year. The rental is then connected to the customer and all the necessary details needed to make a rental such as the car, customer, rental start and end dates, and the total cost will be displayed. The user can also add an additional cost if the car has any damage after it has been returned. Once the rental is complete, the total cost will be calculated and presented in the final report.

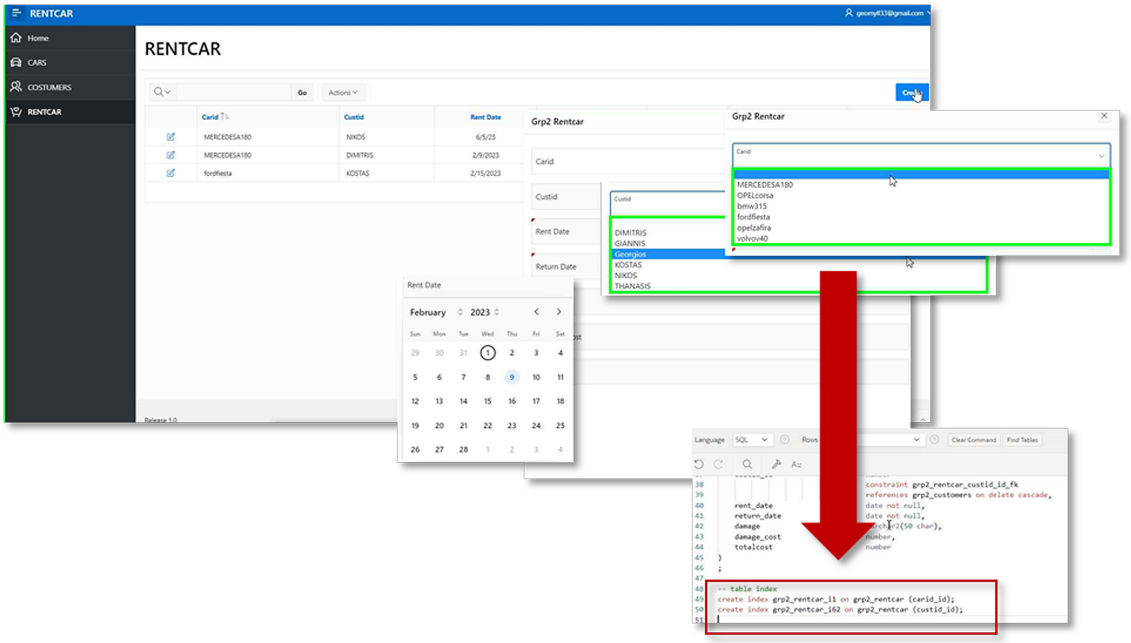


Figure 24.9: Rent car template.

24.6 Supplementary learning material

You can find the following supplementary learning material:

- exported application
- video guides

All supplementary learning material is available on public BeeAPEX project page. Login as a guest user (no password is required). Find textbook in Books section, scripts in folder Part 2 > Chapter24 in the Scripts section and video guides in Collection of video guides. Material for short courses is in Short courses section.

24.6.1 Exported applications

Exported application is packaged. Installation creates tables, index, function, procedure and trigger as well it populates data. De-installation removes all data base objects used in this application.

24.6.2 Video guides

Video guide show every step in application development.

24.7 Questions

1. How can you add an icon in your app logo when you have already created the app?
2. How can you add images to your report pages?

24.8 Answers

1. In your app environment, choose **Shared Components** then choose **User Interface Attributes** click on **Edit** in the pop up window and add the image of your preference.
2. First, you need to set the data type of your column as BLOB. Then, in the Application Page Designer for your report template, configure the column as a file browser and change the Storage Type to **BLOB column specifier in item Source Attributes**. Next, go to the form of this template in the App Builder environment and configure the column as a **Display image**. Then, configure the BLOB attributes by connecting the table, column, and primary key ID that is related to this column.

Bibliography

Articles

[1] Hill Kim. “Altruistic cooperation during foraging by the Ache, and the evolved human predisposition to cooperate”. In: *Human Nature* 13 (Mar. 2002), pages 105–128. DOI: 10.1007/s12110-002-1016-3 (cited on page 185).

Books

[2] Roy Fielding and Richard N. Taylor. *Principled design of the modern Web architecture*. Edited by Mehdi Jazayeri andAlexander L. Wolf Carlo Ghezzi. Association for Computing Machinery, 2010. ISBN: 978-1-58113-206-9. DOI: 10.1145/337180.337228 (cited on pages 104, 105).

[3] Dariusz Jemielniak and Aleksandra Przegalinska. *Collaborative society*. MIT Press, 2020 (cited on page 185).

[4] OMG. *Business Process Model and Notation (BPMN), Version 2.0*. 2011. URL: http://www.omg.org/spec/BPMN/2.0 (cited on pages 262, 263).

[5] OASIS OPEN. *Universal Business Language Version 2.1, OASIS Standard, 04 November 2013*. 2013. URL: http://docs.oasis-open.org/ubl/os-UBL-2.1/UBL-2.1.pdf (cited on page 261).

[6] A. Osterwalder and Y. Pigneur. *Business model generation: a handbook for visionaries, game changers, and challengers*. Volume 1. John Wiley and Sons, 2010 (cited on pages 37, 38).

Index

abstraction level, 50
access control, 86
APEX docker, 44
application builder, 83
application interfaces, 231, 240, 247, 257, 274, 294, 303, 319, 332, 353, 371, 395
application logic, 87
application programming interface, API, 105
application report, 103
attribute, 51
AutoREST, 109

bill-of-material, 309
business process management, 261
business view of the case, 225, 235, 243, 251, 261, 291, 297, 309, 325, 345, 365, 389

calendar, 87
car rental, 389
cardinality, 51
chart, 87
column, 54
computation, 87
create application, 87
create application wizard, 84
create page wizard, 86
CRUD operation, 105, 108

data exchange, 93
data manipulation, 68
data model, 50, 228, 238, 246, 252, 271, 293, 300, 313, 327, 349, 367, 392
Data Modeler, 61
data structure, 50
data type, 51
data workshop, 94, 100
database management system, 49
database system, 49
DB, 50
DB layer, 50, 78
DB schema, 54
DBMS, 49
DBS, 49
DDL, 60
DML, 68
domain, 51
DQL, 70

entity, 51
entity instance, 54
entity relationship diagram, ERD, 50, 51
ER, 50
ER diagram, 50, 51
export data, 100
export wizard, 100

first normal form, 1NF, 59
Flows for APEX, 264
foreign key, 56

how to benefit from gallery of applications and plug-ins, 197
how to collaborate in team, 185
how to exchange data in APEX, 93
how to generate first draft of application, 113
how to manage forms, 157
how to manage menus, 181
how to manage packaged and multilingual applications, 205
how to manage reports, 135
how to navigate in APEX, 81
how to prepare a database, 49
how to start Oracle APEX, 35
HTTP request, 105, 108
HTTP response, 106
https://apex.oracle.com, 43

import data, 94
interactive report, 87
intranet, 225
item, 87

list, 87
list of values, LOV, 210
load wizard, 96

logical data model, 50, 228, 238, 246, 252, 271, 293, 300, 314, 327, 350, 368, 393
logical model, 50

multilingual, 217
multiple languages, 217

normalization, 59

object browser, 67, 69, 101
office hours scheduling, 345
on-premise, 43
Oracle Academy, 47
ORACLE APEX, 81
Oracle Cloud Infrastructure, 46
Oracle REST Data Service, ORDS, 105, 106

packaged application, 205
page, 87, 90
page creation, 86
page designer, 89
page type, 87
PL/SQL, 206
problem definition, 225, 235, 243, 251, 264, 291, 297, 309, 325, 346, 365, 389

query builder, 75
Quick SQL, 63, 69, 120, 205
QuickSQL and Script SQL Script, 329

RDB, 54
RDB schema, 54, 58
RDB schema generation, 55
RDBS schema management, 60
region, 87
relation, 54
relational data model, 230, 239, 246, 256, 271, 294, 300, 314, 328, 350, 368, 393
relational DB schema, 54
relational model, 50, 54
relational model generation, 56
relationship, 51
representational state transfer, REST, 104
resource handler, 108
resource module, 108
resource template, 108
REST, 104
RESTful access, 106
RESTful service, 104

sample and starter apps, 197
second normal form, 2NF, 59
sequence, 273
skill level, 42
small innovation system, 251
SQL aggregation, 74
SQL ALTER TABLE, 67
SQL COUNT, 74
SQL CREATE TABLE, 66
SQL data definition language, 65
SQL data definition language, SQL-DDL, 60
SQL data manipulation language, SQL-DML, 68
SQL data query language, SQL-DQL, 70
SQL DELETE, 69
SQL Developer Data Modeler, 50
SQL DROP TABLE, 66
SQL INSERT, 68
SQL JOIN, 73
SQL ORDER BY, 71, 74
SQL Script, 121
SQL SELECT, 71, 102
SQL SUM, 74
SQL TO_CHAR(), 72
SQL UPDATE, 68
SQL WHERE clause, 68, 72
SQL workshop, 67
SQL-DDL, 60, 68, 205
SQL-DQL, 70
stateless REST, 105
stored function, 273
structured query language, SQL, 60

table, 54
telecommunication services, 365
third normal form, 3NF, 60

use case diagram, 226, 236, 244, 252, 265, 293, 298, 310, 327, 347, 366, 390
use case, UC, 39
Use cases, 310
use cases, 226, 236, 244, 251, 265, 292, 298, 325, 346, 365, 390
user authorisation, 243
user feedback, 86

validation, 87
Virtual Box Appliance, 44

web application, 49, 89
web application development process, 81
workflow model, 268