Web-Application Modelling
Tutorial | ORACLE APEX

This tutorial shall show you how to model and generate a simple sample Web application ("Hotel-App") with ORACLE.

Throughout this tutorial, you will perform the following steps:

1) Create an online ORACLE APEX workspace
2) Create tables and insert data
3) Create a first Web Application
4) Add different additional Pages
5) Add manual linking

7) Access the generated web application

NOTE:

We are going to use **ORACLE APEX https://apex.oracle.com/ in the Cloud**.
The functioning of your application will be dependent on the service of ORACLE.

For this we are generating an **online workspace**. Thus, development and accessing will only be possible with an existing internet connection.

Please note that the **workspace will be purged in case of inactivity** over a certain period. Make sure to reactivate the workspace in time.

The Johannes Kepler University is member of the **ORACLE Academic program https://academy.oracle.com/en/oa-web-overview.html**.

In the course of that you are able to access the **ORACLE APEX training material** available.
https://apex.oracle.com/en/learn/

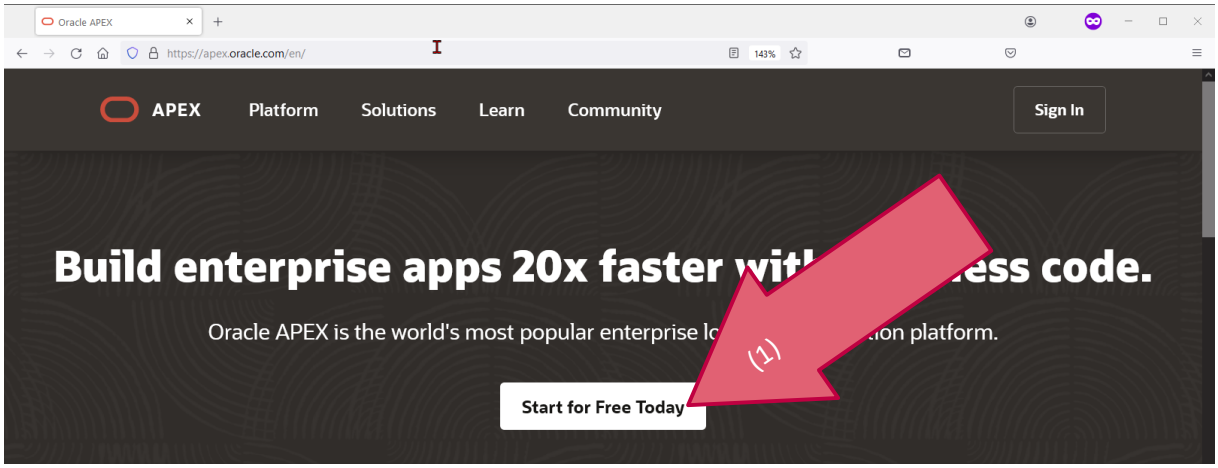
Thus, ORACLE APEX is a **low code development platform**.
While generators will generate most of the code necessary, however, to realize certain functionality **some ORACLE SQL / PLSQL knowledge will be necessary**.
https://www.oracle.com/database/technologies/appdev/sql.html
https://www.oracle.com/database/technologies/appdev/plsql.html

1) Generate online WORKSPACE

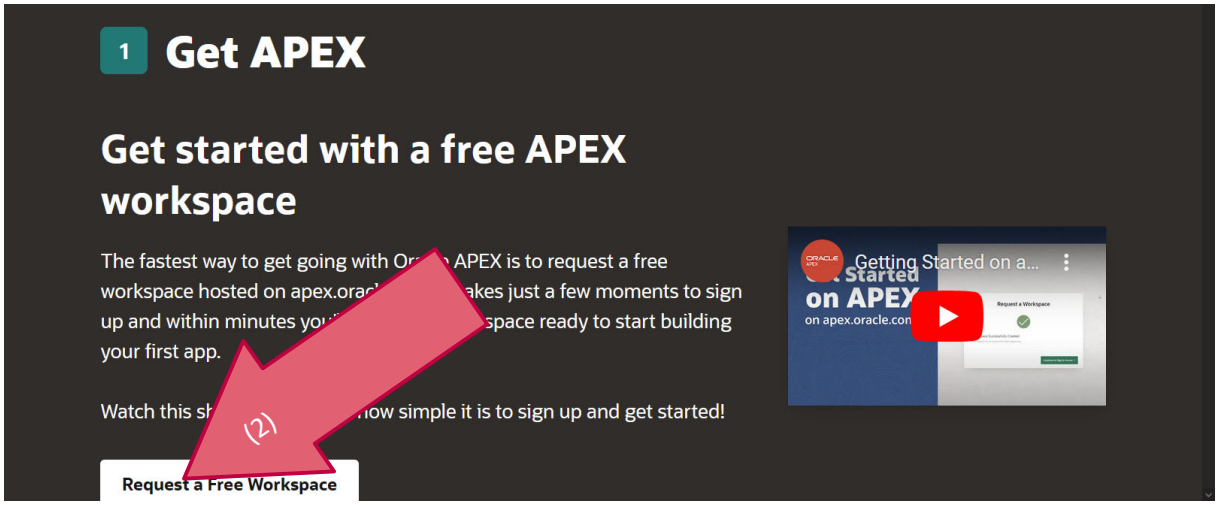
For specifying the Web application as well as for accessing it we will use a web-browser throughout.

First we need to start with creating a new ORACLE APEX workspace.

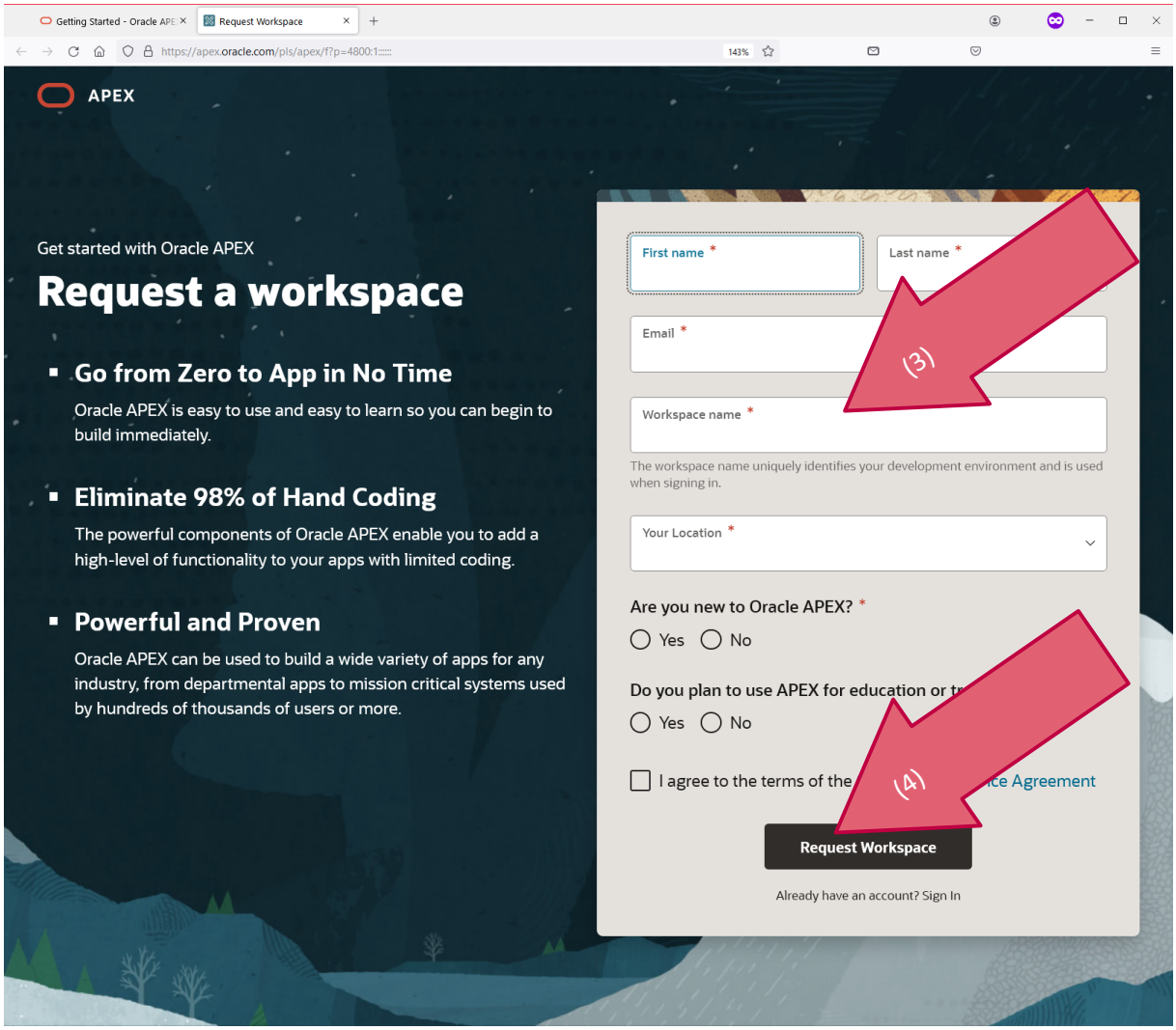
1. ***Navigate to: https://apex.oracle.com/***



2. Request a free workspace



3. Fill out the from



We need to provide your information and agree to the ORACLE APEX Service Agreement.

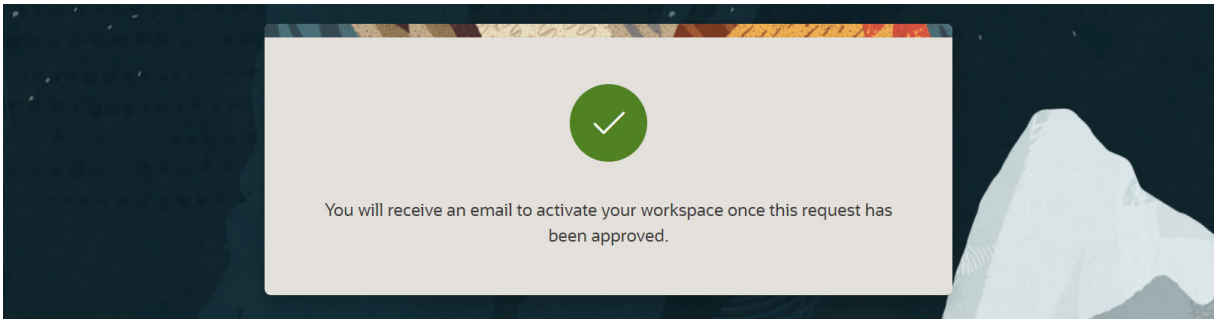
NOTE:
Workspace names need to be unique throughout the ORACLE APEX system. Thus, you might want to use something as:

MIA2023_<Lastname>_workspace1

to make it unique.

You are able to add more than one workspace when needed.

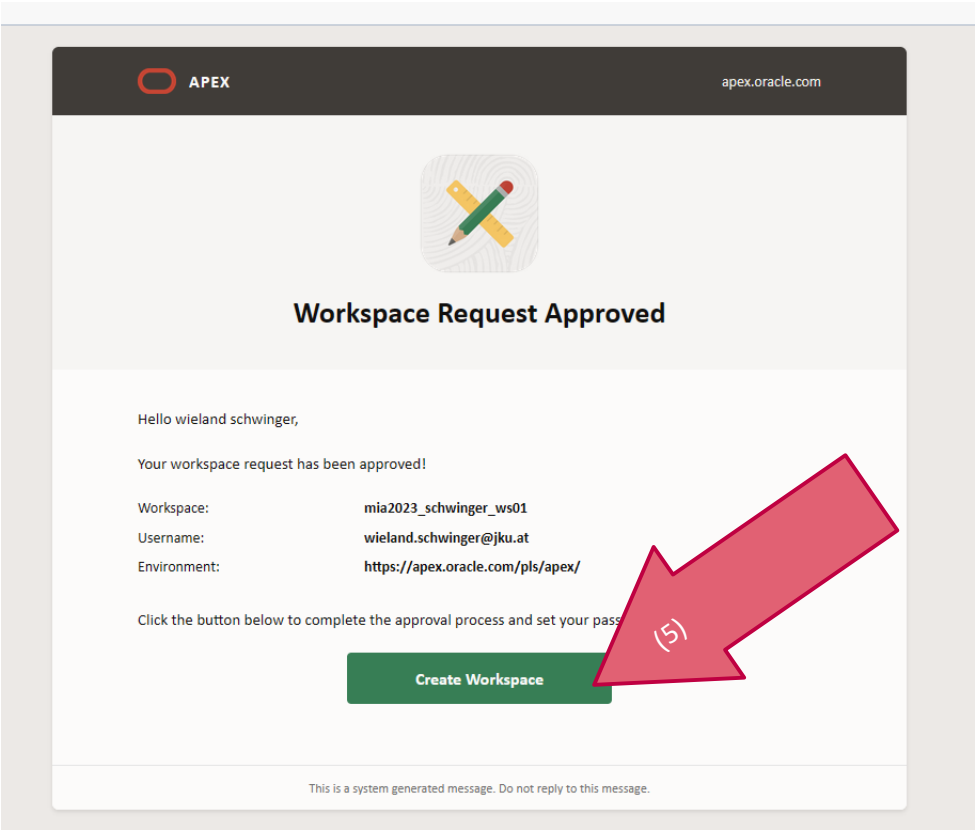
4. “Request Workspace”



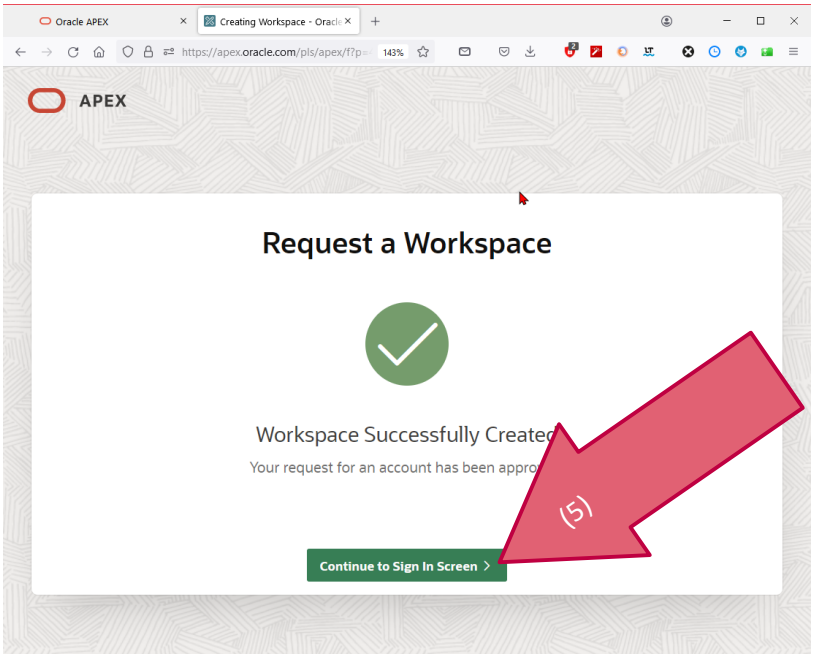
After you press the button “Request Workspace” the request is transmitted to ORACLE.

5. Confirm request

In turn you will receive a confirmation email. Follow the link in the email.



Finally the workspace is actually created in the ORACLE APEX Cloud.

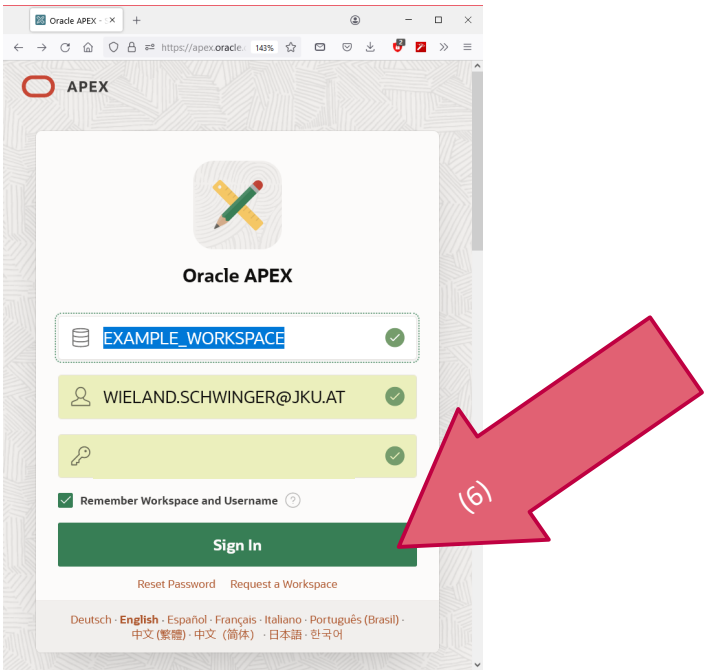


6. Login to ORACLE APEX Workspace

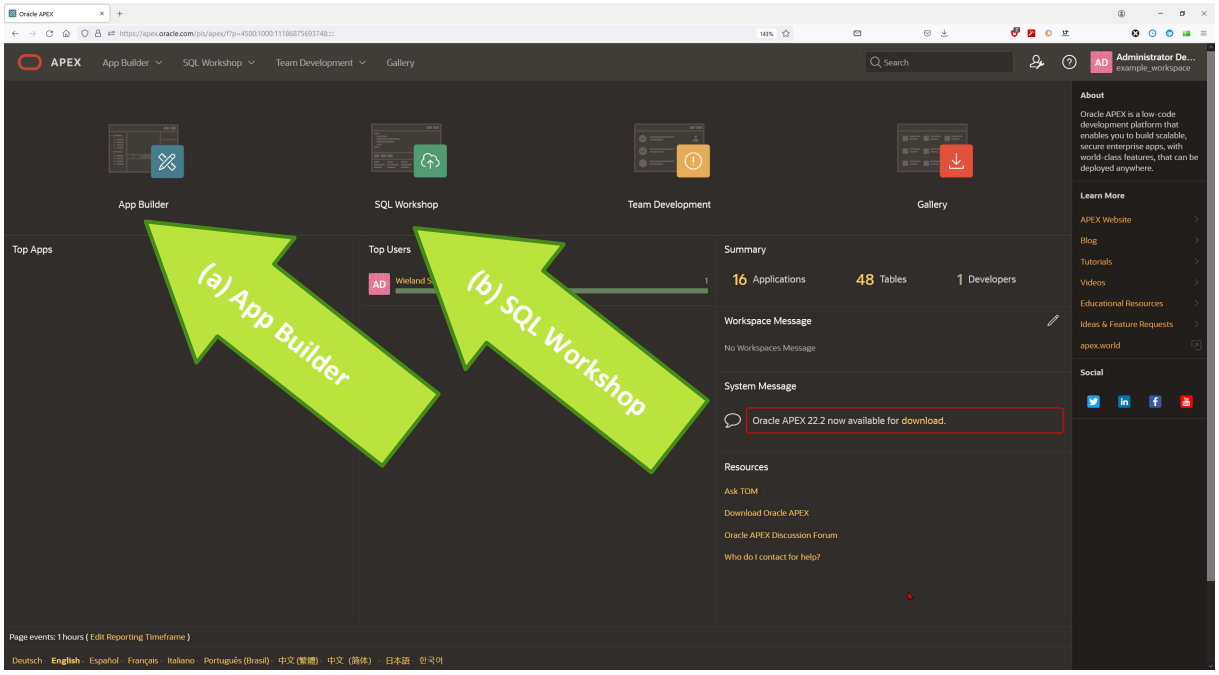
Now that you have successfully created the workspace you are able to sign in.

NOTE:
On the first sign in you need to set your password for your login.

Use your **workspace** and your **email** and **password** for signing in.



You are redirected to ORACLE APEX. Here two tools are of particular importance for us:
(a) App Builder to create and specify Web applications and
(b) SQL Workshop to create and fill the underlying tables.

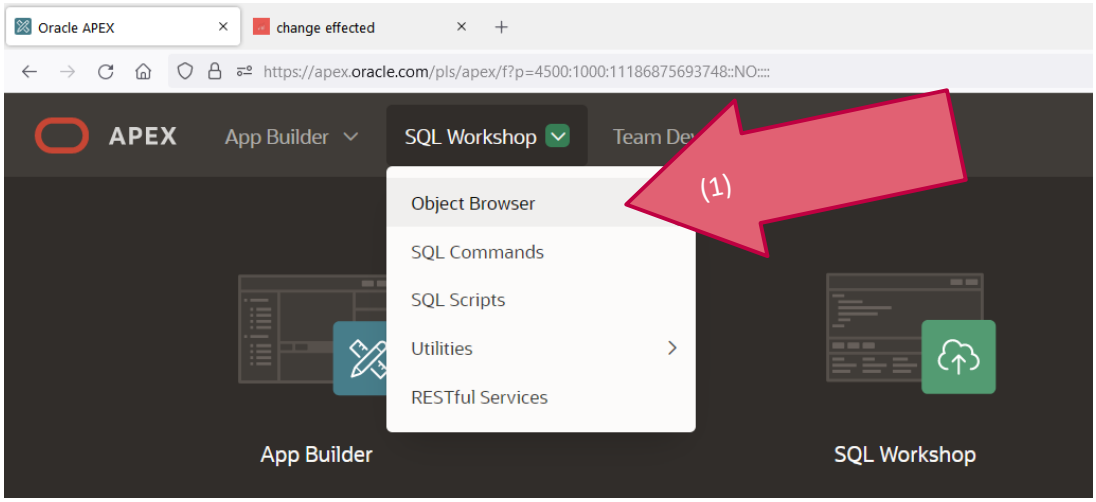


2) Generate DATABASE TABLES

For APEX apps the data is stored in ORACLE **database tables**.

These can be generated / managed through various ways. Consult the additional training materials for familiarizing your self with the different options.
Within this tutorial we will focus on the **ORACLE Object Browser** within the **ORACLE Workshop** only.

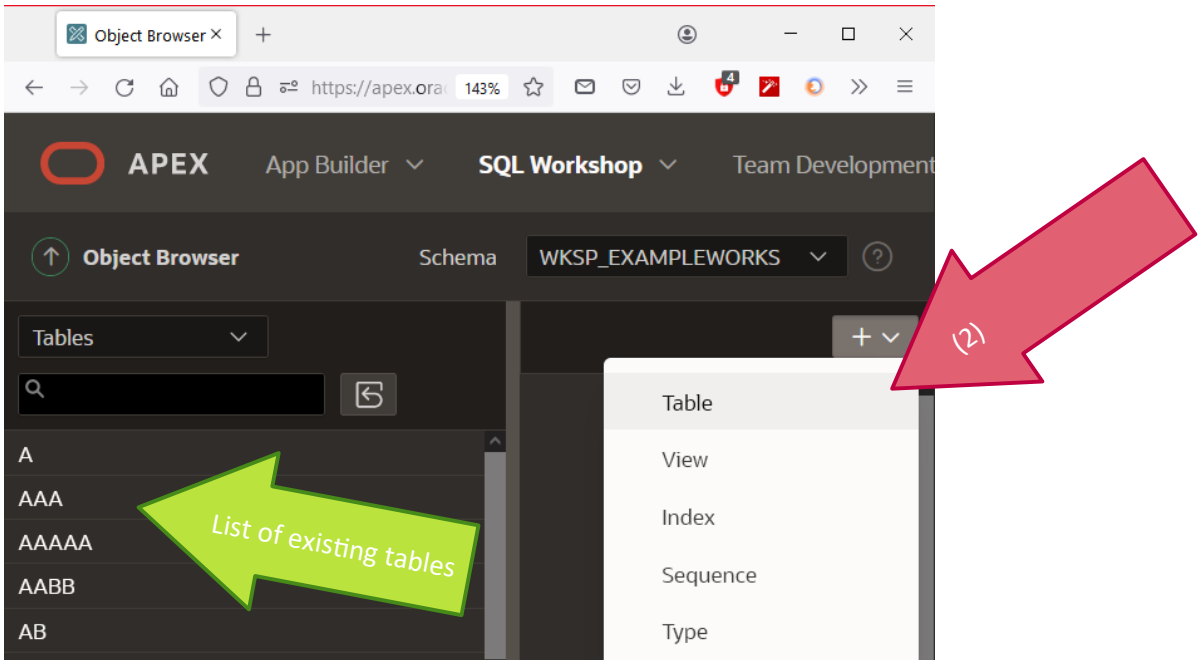
1. SQL Workshop → Object Browser



2. Create new Table

On the left side pane you see already existing tables.

Use the **+** symbol to create a new table.



3. Specify Name and Columns

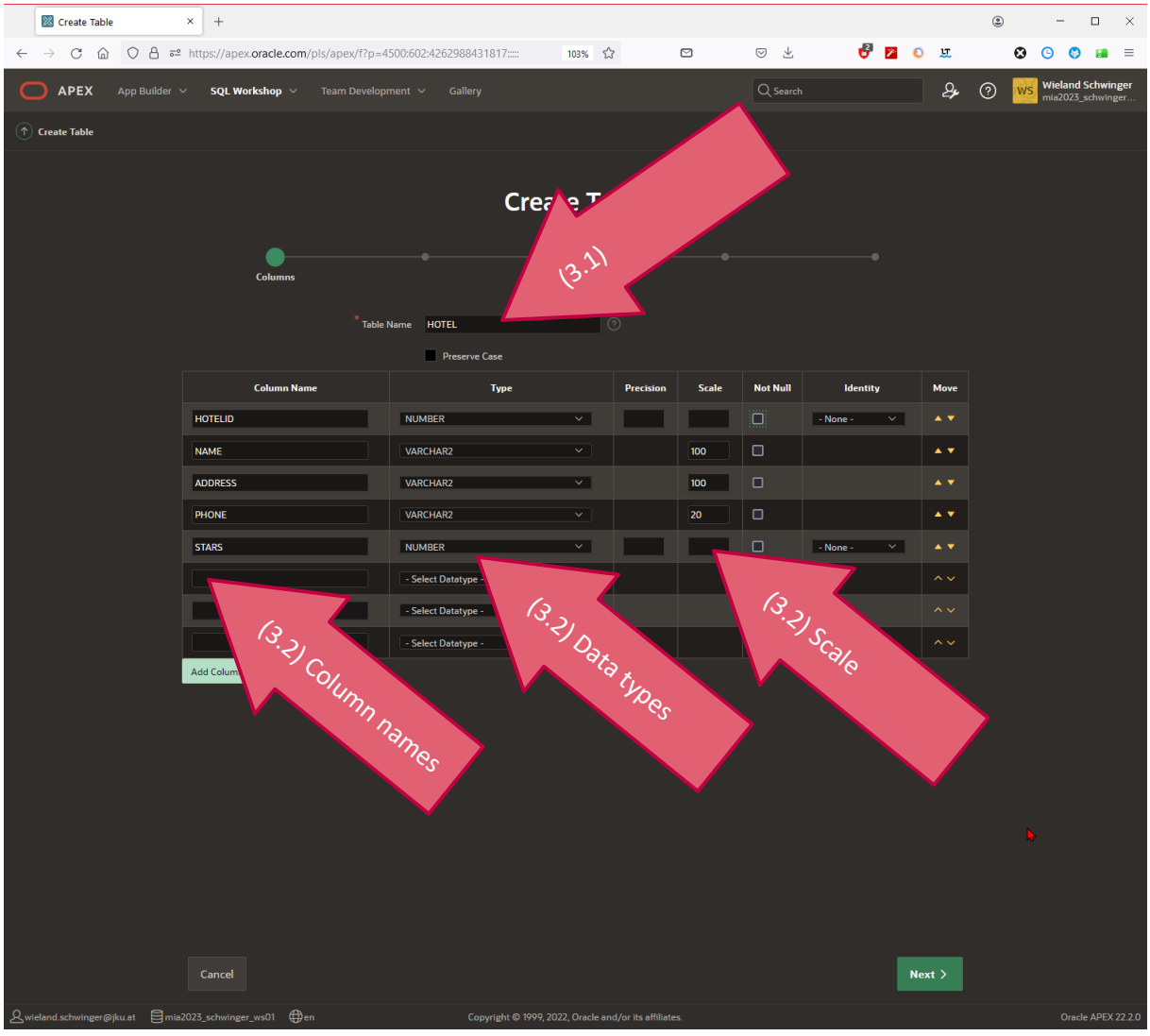
3.1. Specify Name of Table

- **Name** the Table **“Hotel”**.

3.2. Add the Columns

- **Add** the following **columns** to the table:
  - **hotelID** NUMBER
  - **name** VARCHAR2 100
  - **address** VARCHAR2 100
  - **phone** VARCHAR2 20
  - **stars** NUMBER

It is required to specify the ORALCE specific physical data types. Thereby, NUMBER indicating a numerical and VARCHAR2 with the according length of characters a textual data type.



NOTE:
We will not specify the Identity for our "HOTELID" as we will specify the population of the primary key in the next step.

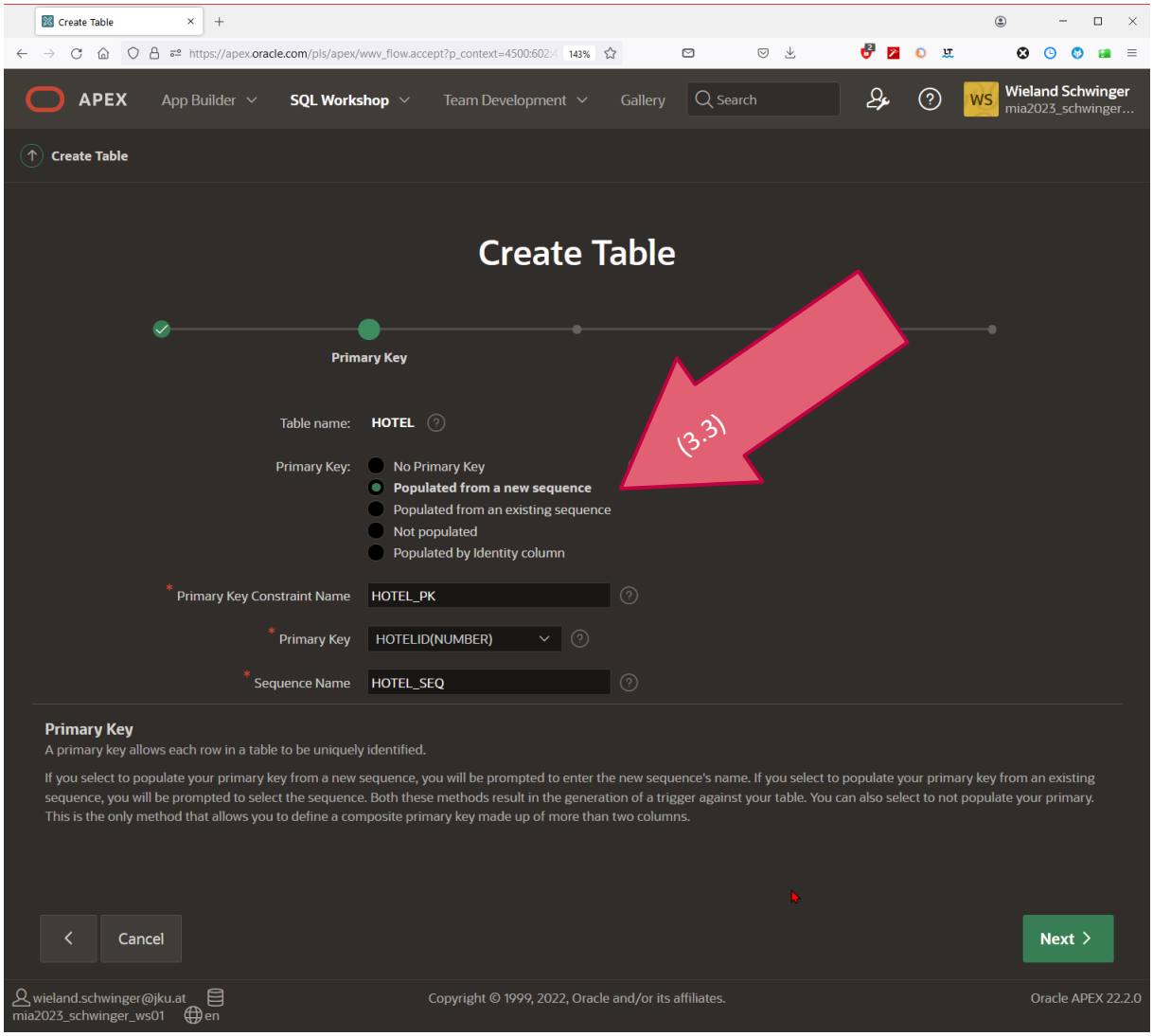
3.3. Specify Primary Key

- Select that the *Primary Key* to be **"Populated from a new sequence"**. This will automatically assign a new value for a new entry in that table.

- Select the **"Primary Key"** to be the **"HOTELID"** from the drop down list.

- You may leave the **"Primary Key Constraint Name"** as is. If you change the name of the *Primary Key Constraint* make sure to name it uniquely in the workspace.

- Likewise you may leave the **"Sequence Name"**, being the source of the next number to be used for the primary key, as is.

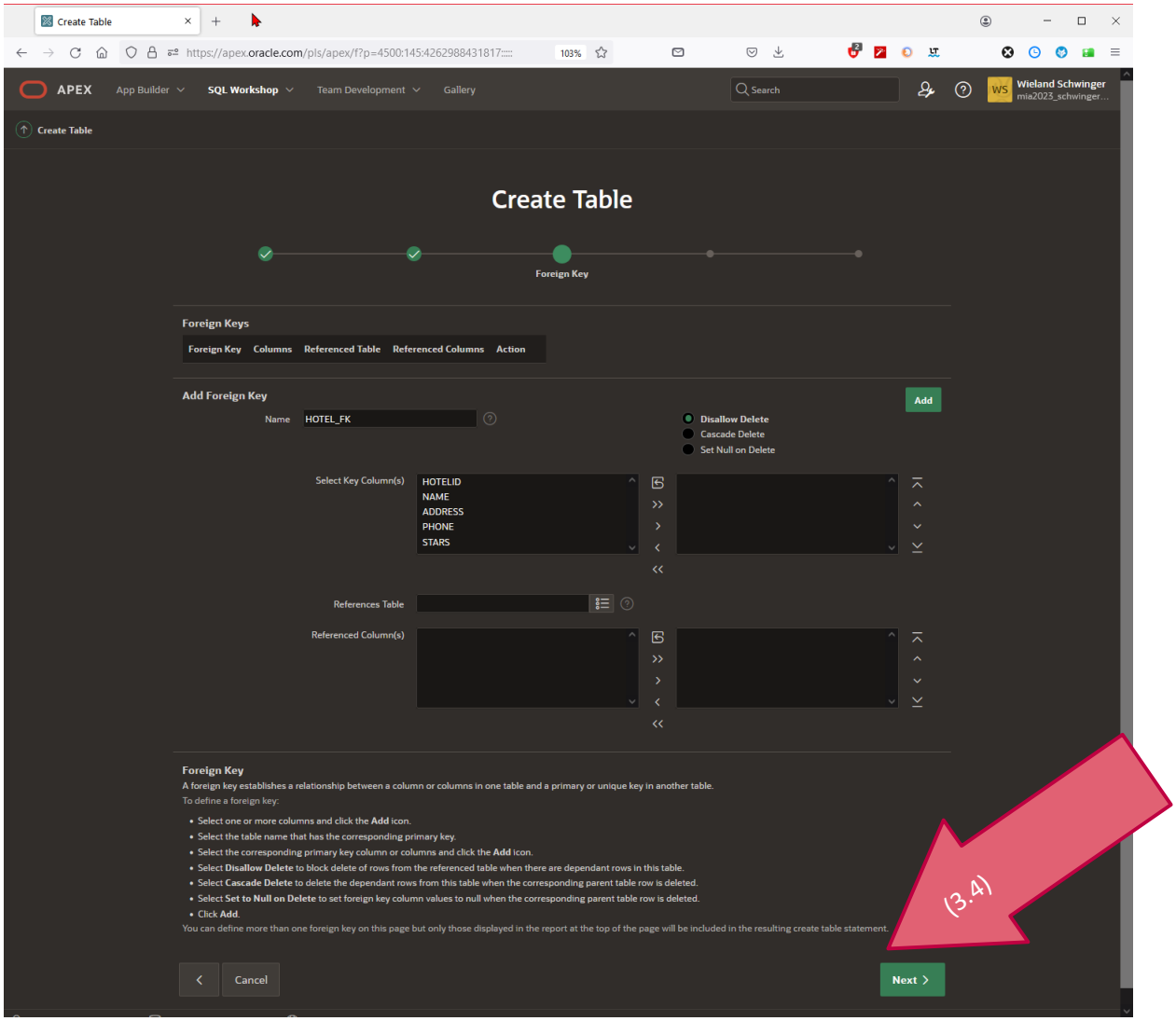


3.4. Specify Foreign Keys

Here you would specify that a column of the table actually represents a **Foreign Key** to another existing Table along with the *Delete Semantics*.

At this stage, we will not specify any Foreign Key. However, we will need to alter the specification of the table later on to include a foreign key.

Just continue with **"Next"**.

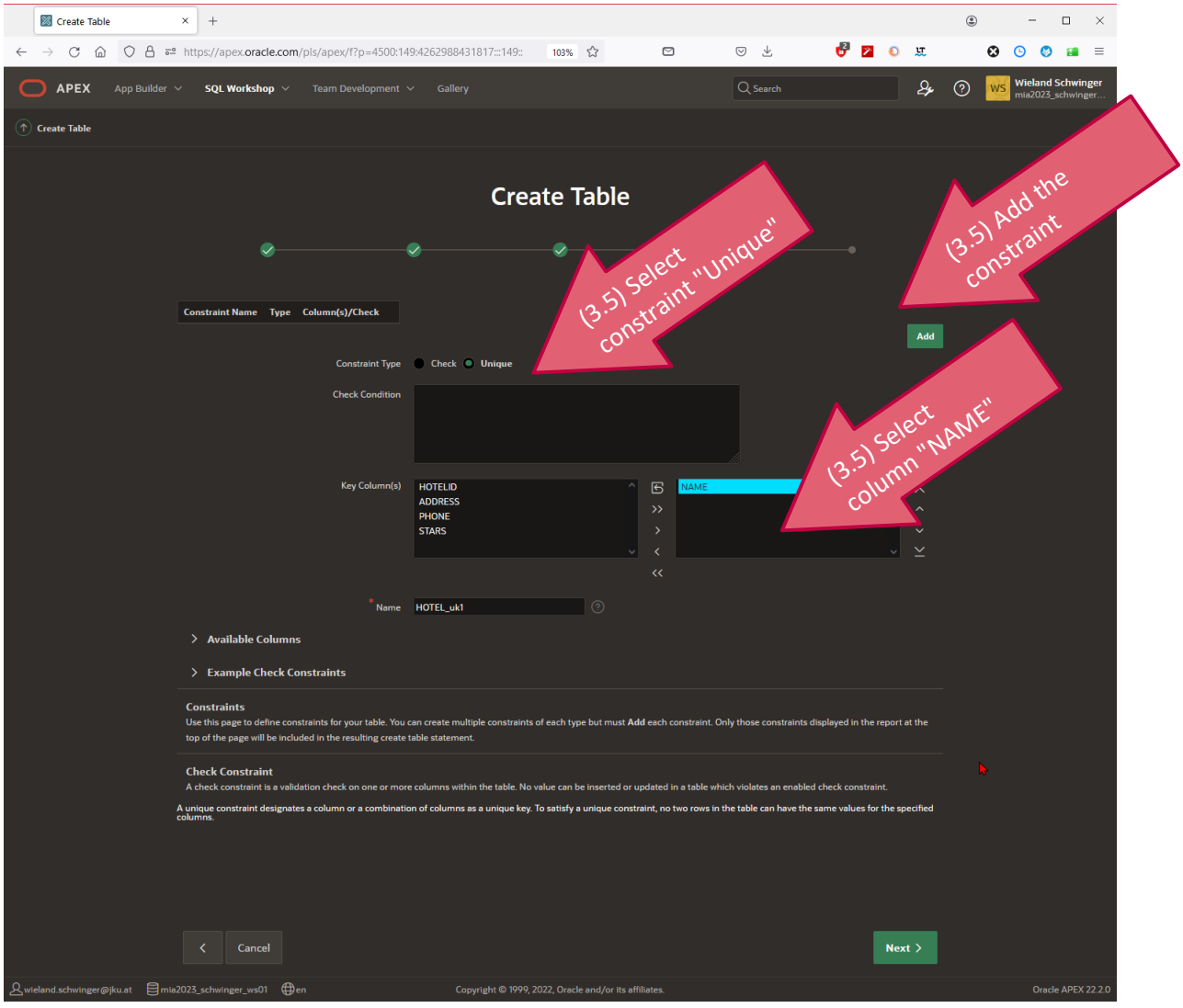


3.5. Specify Further Constraints

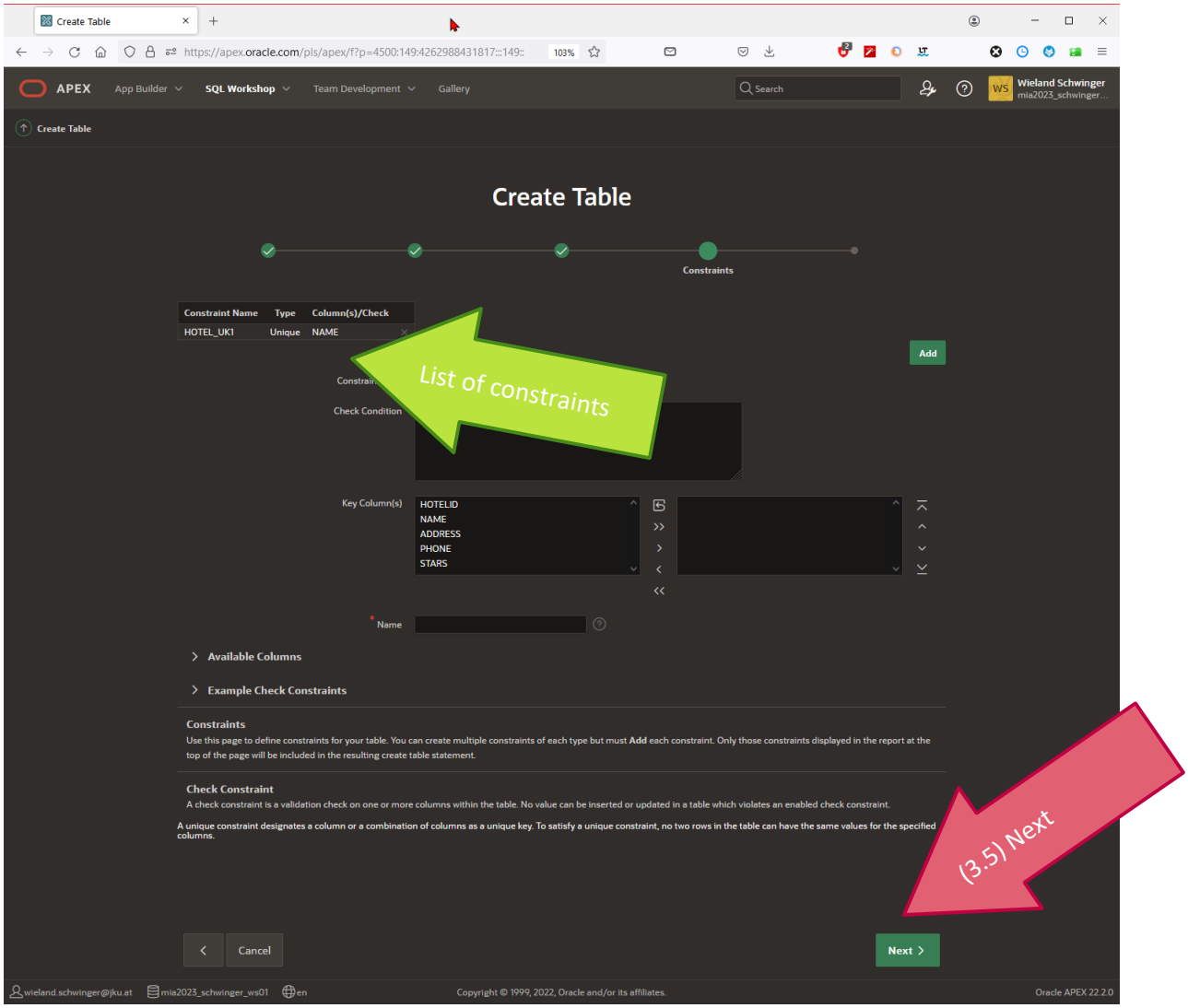
Here you would specify further check constraints on the values of columns.

- We may want to set a **“Unique” constraint** for the column **“Name”**.
  - For this select the **“Name”** out of the list of “Key Column(s)” using the **>** operation to move it to the right panel.
  - Select the checkbox in front of **“Unique”**.
  - **Add** the constraint through the **"Add"-button**.

The **name** of the Unique-Constraint can be left as is again.



The constraint is added to the table. This can be seen as the Constraint is now listed for the table.



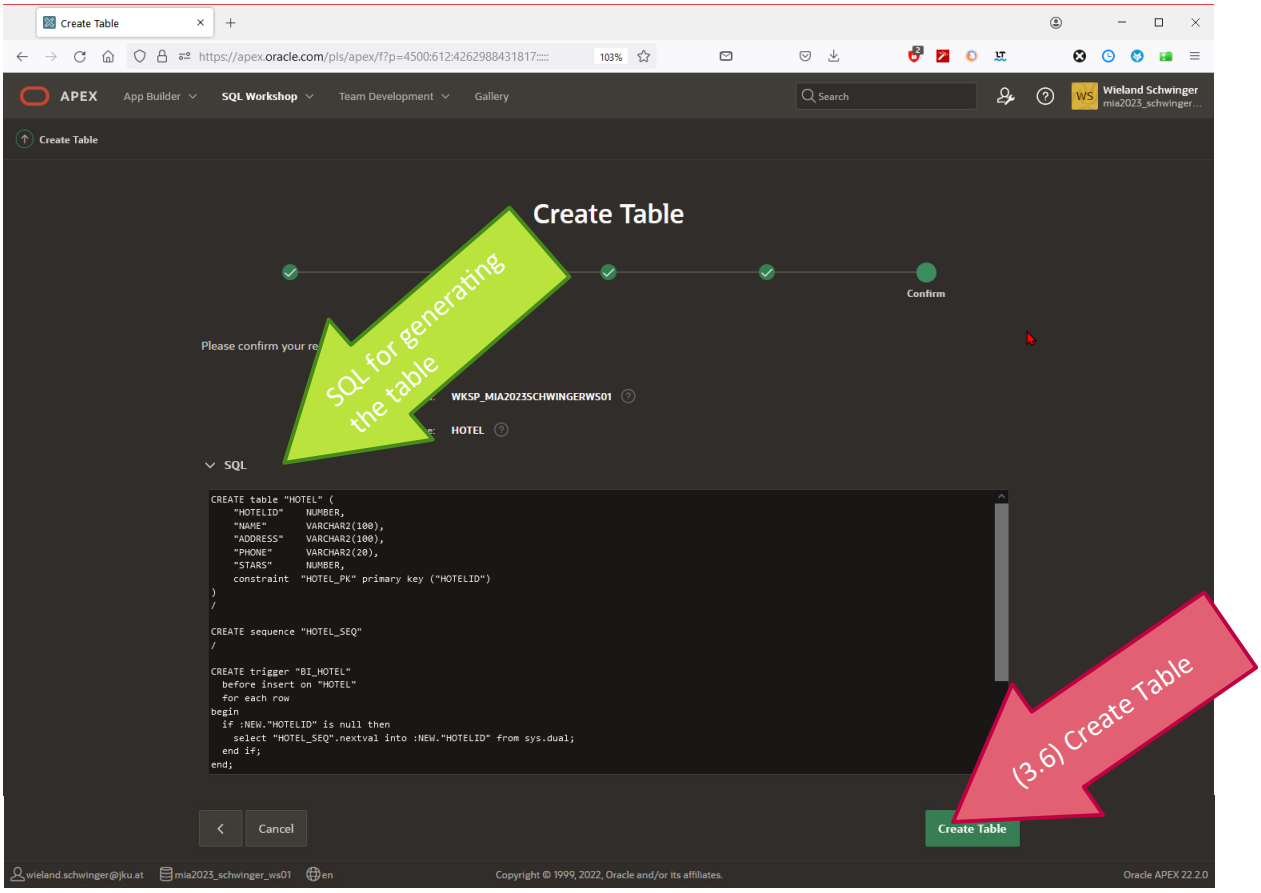
To delete a constraint use the **x** at the end of the listed constraint.

Use **Next >** to continue.

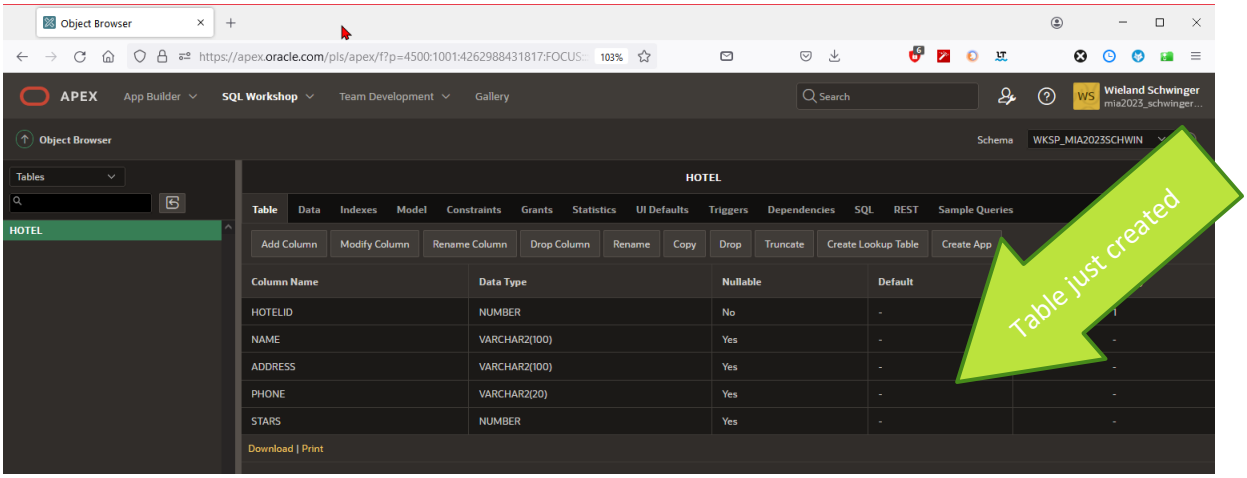
3.6. Confirm Table Creation

You can now investigate the SQL generated to create the table by opening the **SQL**-Section.
The complete SQL-Code for creating the table is shown, which includes the CREATE-TABLE-Statement as well as the CREATE of a SEQUENCE and the according TRIGGER being initiated when a row is inserted as well as altering the table to include the unique constraint on the column.

Use **Create Table** to actually create the table.



The created table will now be available in the **Object Browser.**



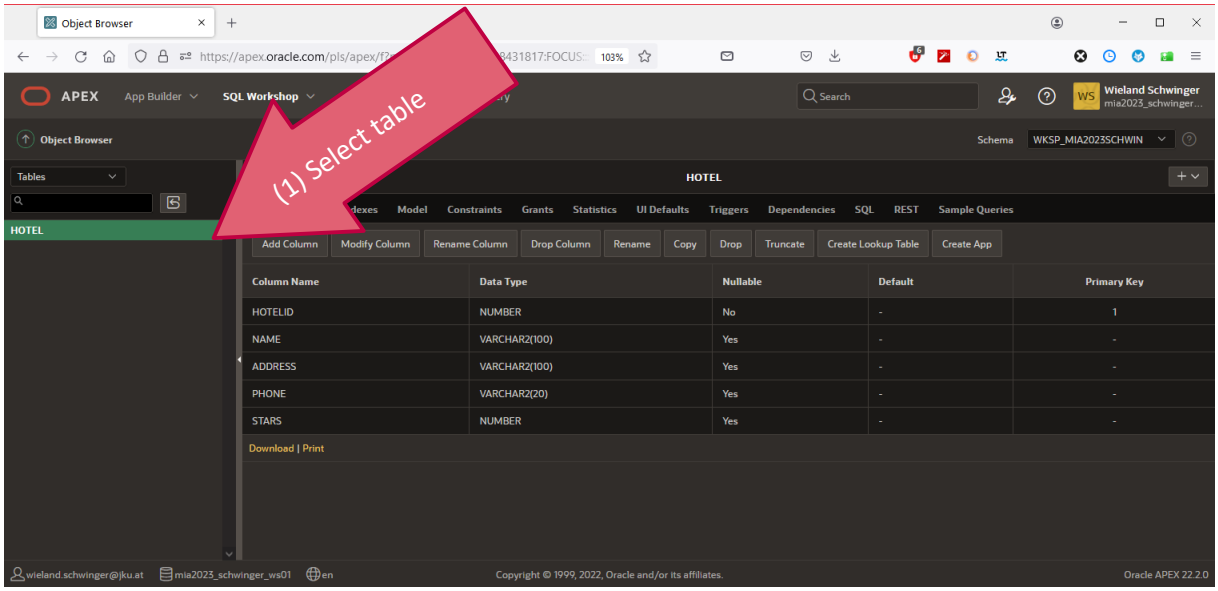
3) Insert DATA

Data can be **inserted** to an existing table through the **Object Browser** of the **SQL Workshop**.

1. Select Table

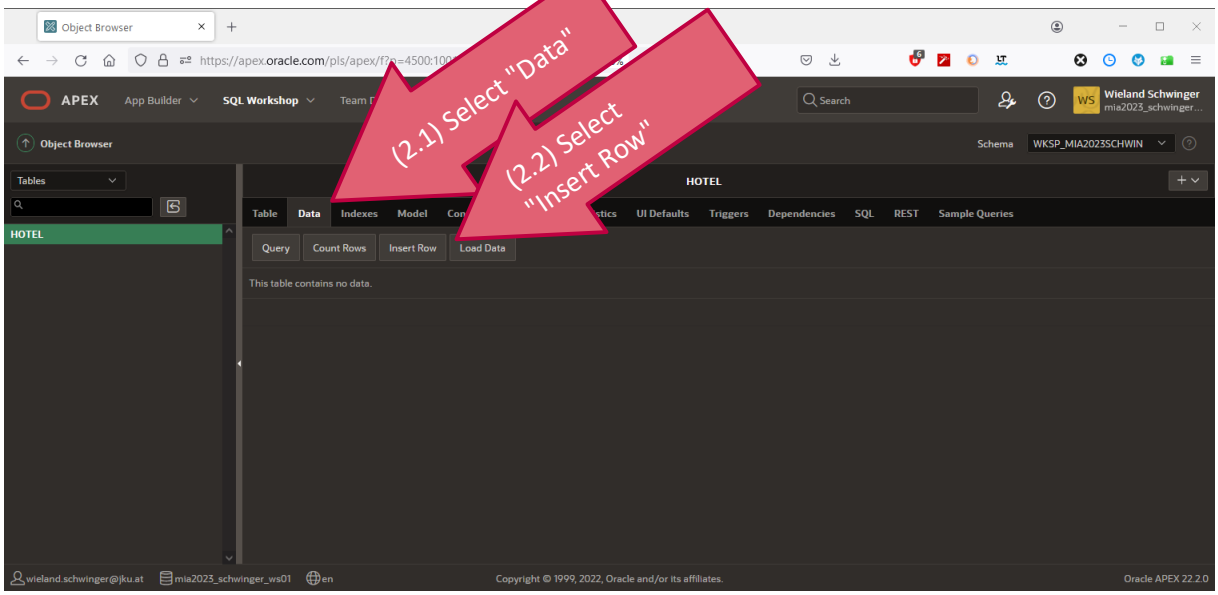
In the **SQL Workshop** chose the **Object Browser**.

Select the table in the left side panel. You may utilize the search function.



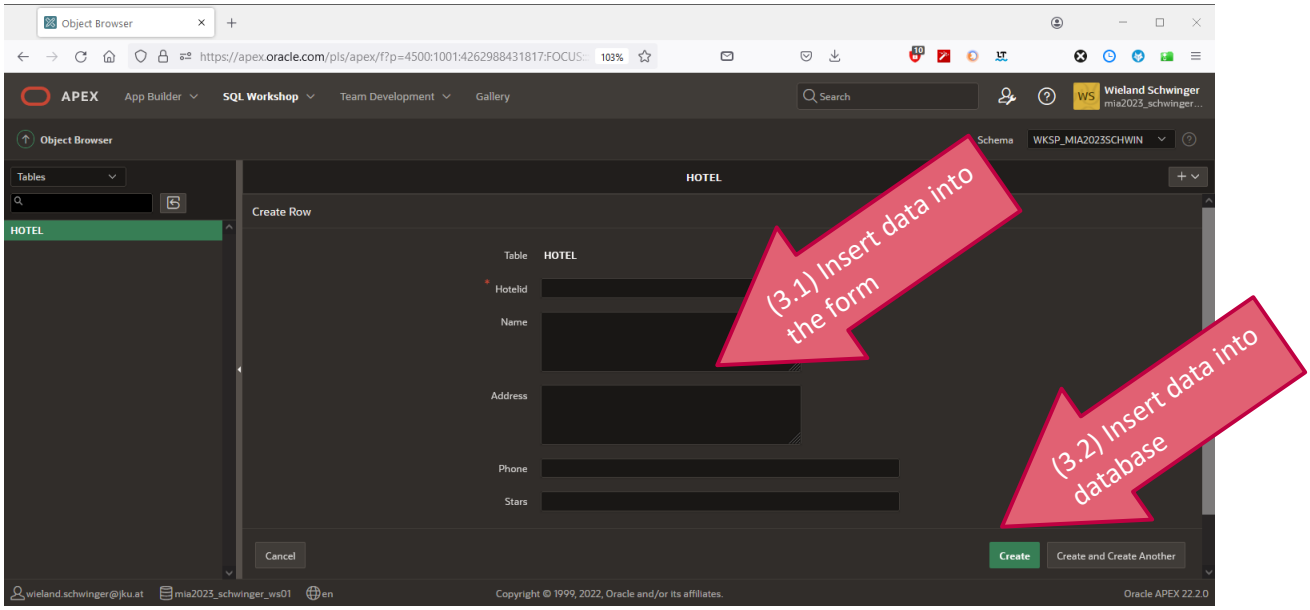
2. Select "Data"

- **2.1** Select the **"Data"-Tab**.
- **2.2** Select **"Insert Row"-Button**.

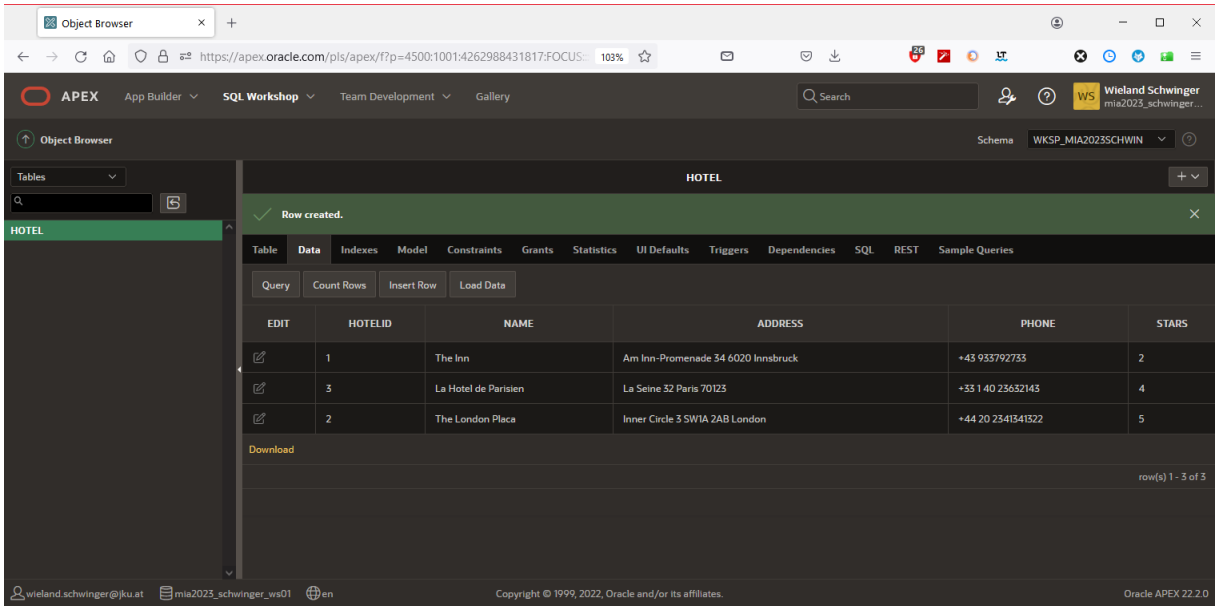


3. Insert an Entry

- **3.1** A **form** is offered to **insert data** into the table.
- **3.2** Use **"Create"-button** to actually insert the data in to the table. Use **"Create and Create Another"-button** for inserting the data into the database and return directly to that empty form for another entry.



Insert some entries into table "Hotel".



Finally, your table shall contain some entries.

4) Creating First WEB APP

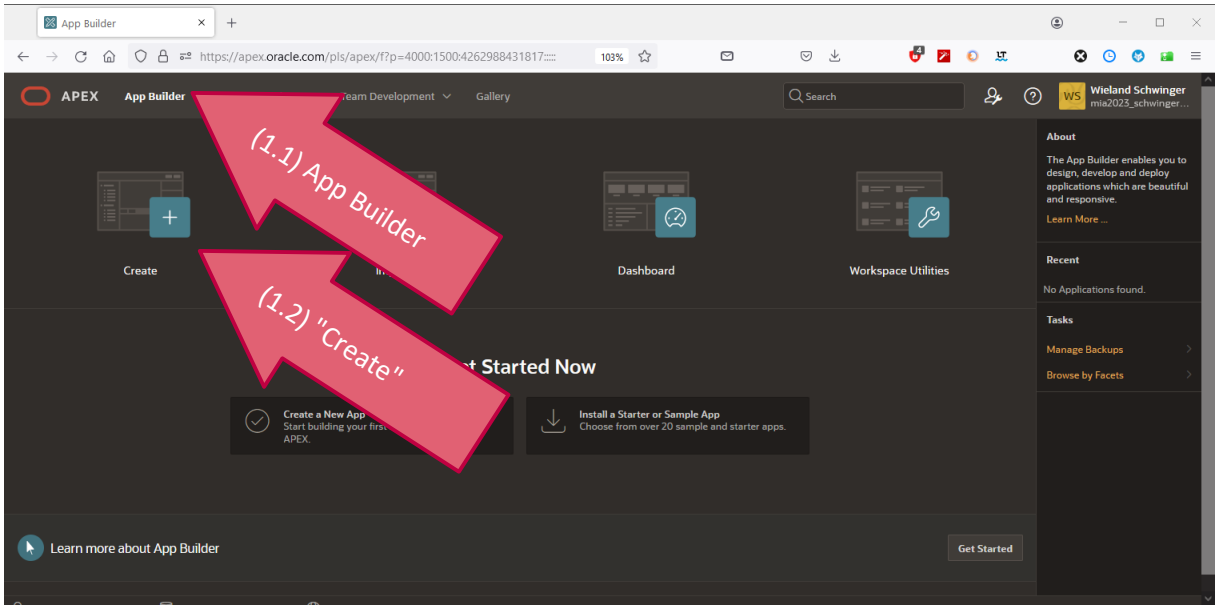
ORACLE APEX Web applications (Web app) comprise **pages** and operate on **tables**. The code for the web apps is generated through the APEX web interface and stored in your workspace.

The Web app is created once. While pages for that Web app can be added and altered later on.

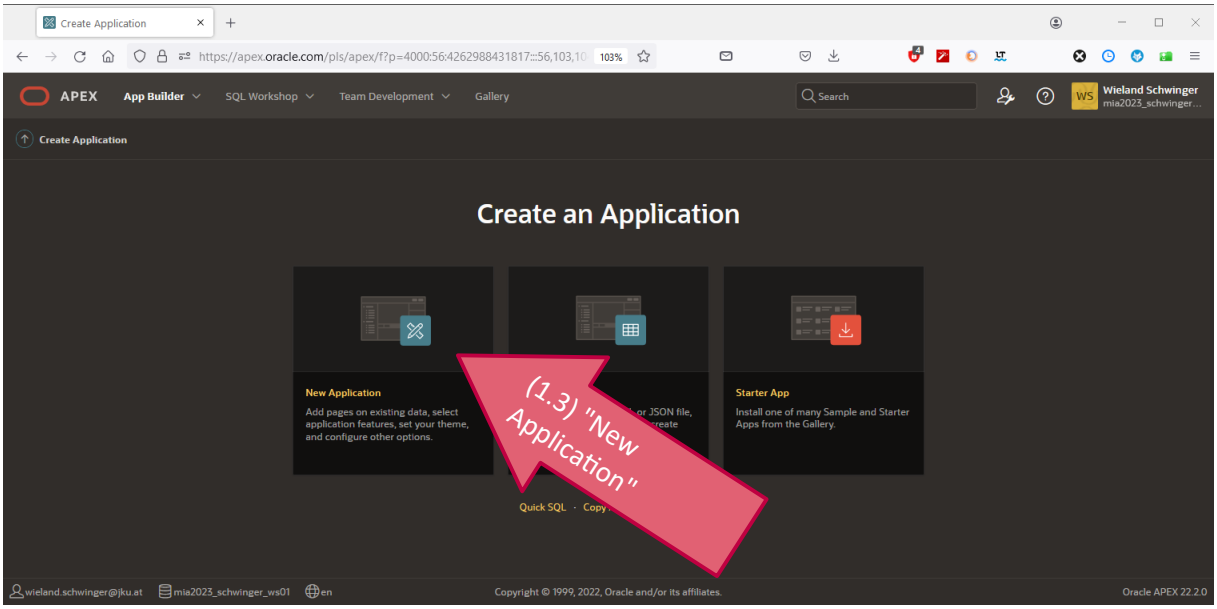
We thus need to create the Web app first. While there are different ways to start in creating a Web app in APEX (see additional training material) we will resort to creating a Web app from scratch.

1. Create Web app

- We use the **(1.1) ORACLE APEX App Builder** using the **(1.2) "Create"-functionality**.



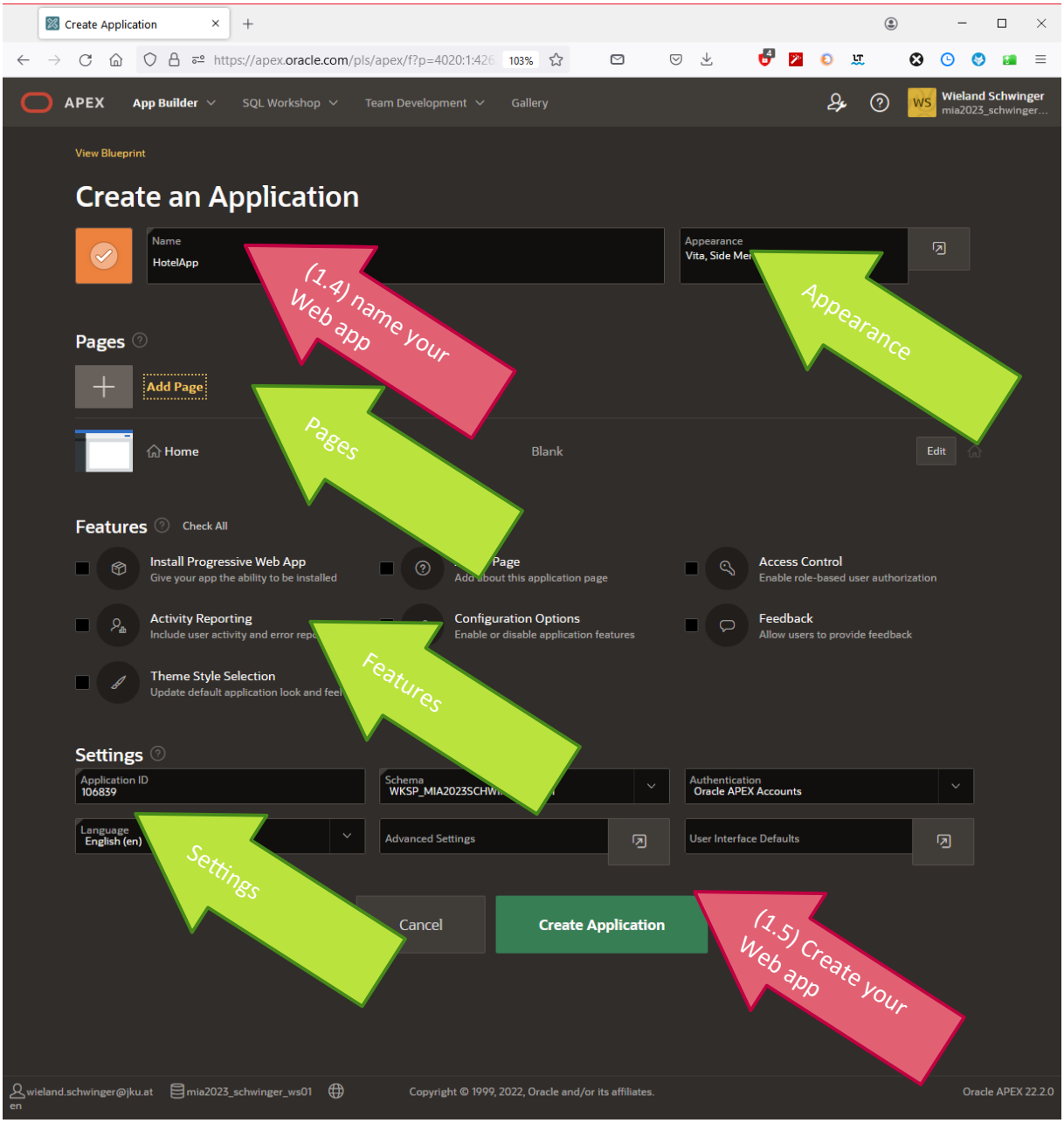
- Chose to create a **(1.3) "New Application"**.



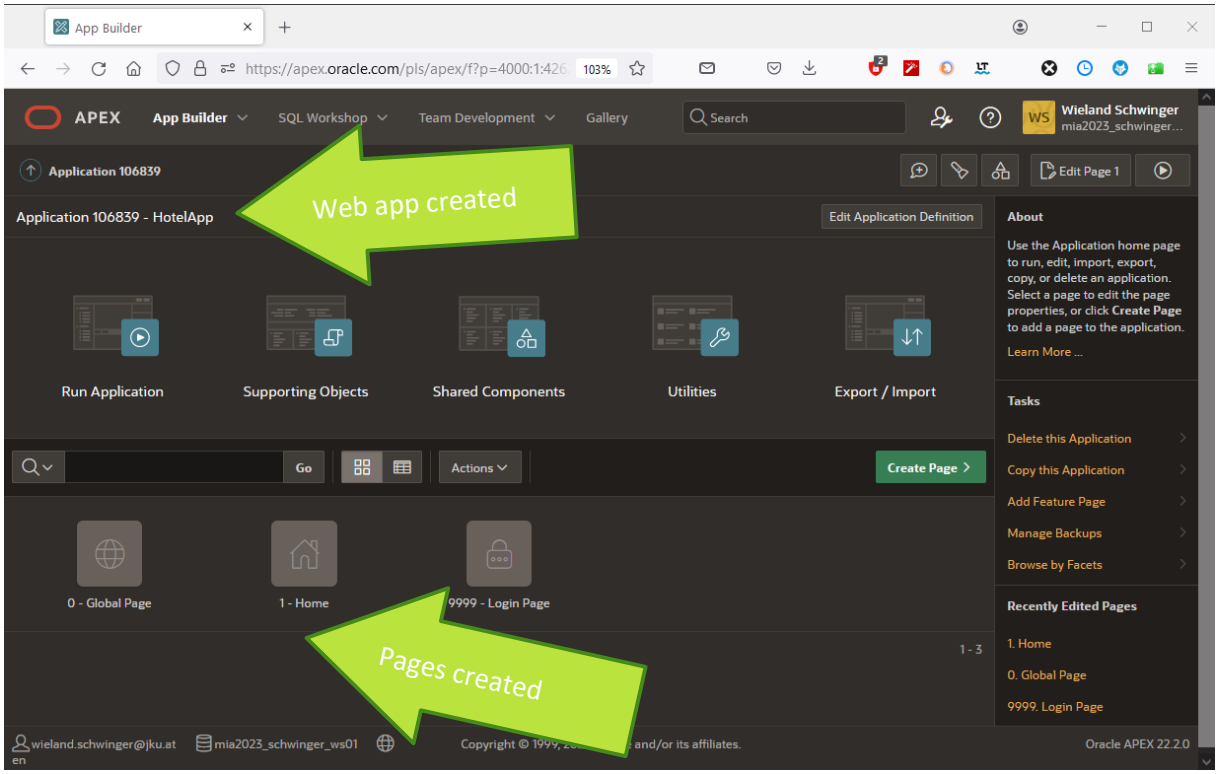
- Next, you need to specify the name of the Web app **(1.4).**

- In the course of this dialogue you may also decide on the **appearance** of the Web app. You may alter the appearance if you like.
- Furthermore, it is already here possible to add **pages** to the Web app even before it has been created. We will specify pages later. Note, however, that a Home-page has already been specified for the Web app.
- Further **features** can be "installed" during the creation of the Web app, like foreseeing a Progressive Web app to render on multiple platforms.
- Additionally, certain settings can be made for the Web app.

- Please consult further training material for details on those advanced features.
- We will generate our first Web app without those advanced features through **"Create Application"**.



After creation you will be redirected to the **Specification Functionality** of the **App Builder** with the newly created Web app selected.



You can see that the Web app contains already some pre-generated pages. These pages somewhat correlate to pages or page impression. However, not all the "pages" listed are self-contained pages which can be accessed directly through the Web browser as some might represent e.g. modal overlay "windows".

Next we will add our first "real" page to our Web application.

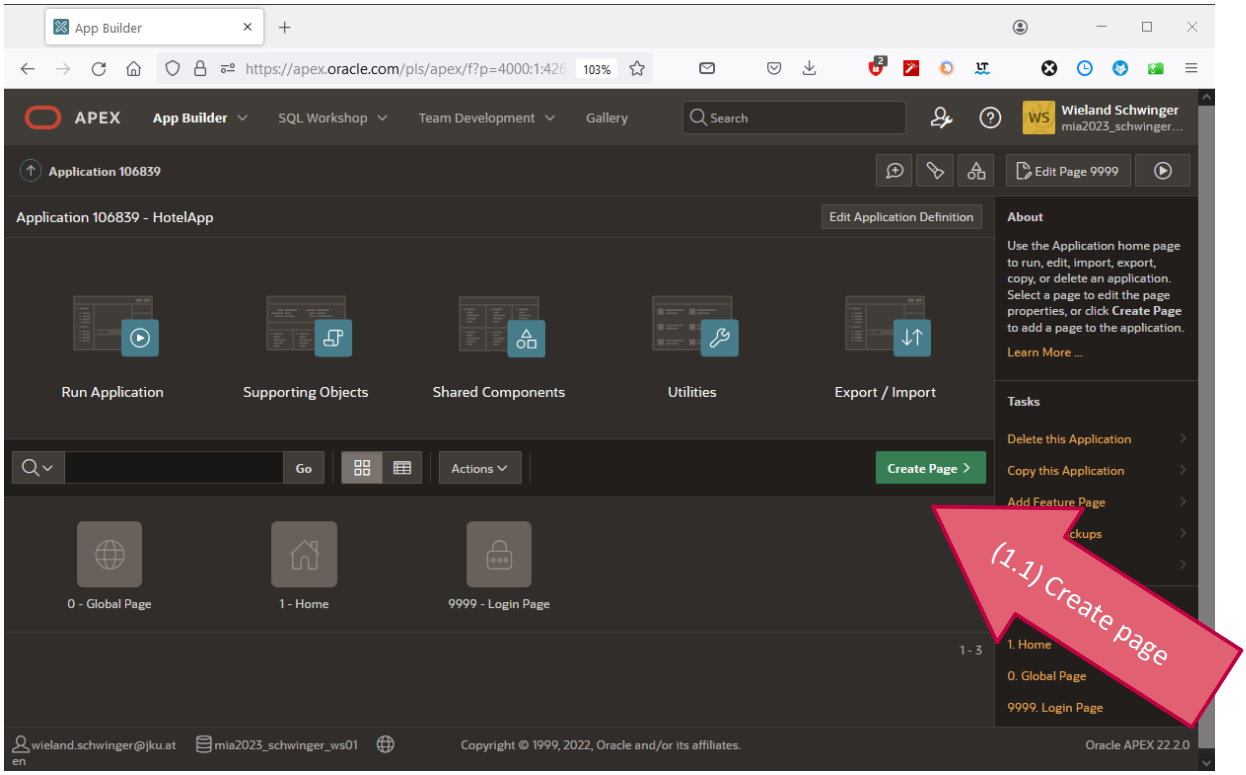
5) Adding First PAGE

Pages can be added to a Web application as desired. In general, pages operate on **tables** to e.g. display their data.

APEX offers a wizard to create pages. During that wizard you may specify certain aspects of the page, most importantly the type of page. This specification is than used to generate a standard representation of that page. That generated page specification can later be changed. However, it is of course advised to start with the appropriate page type.

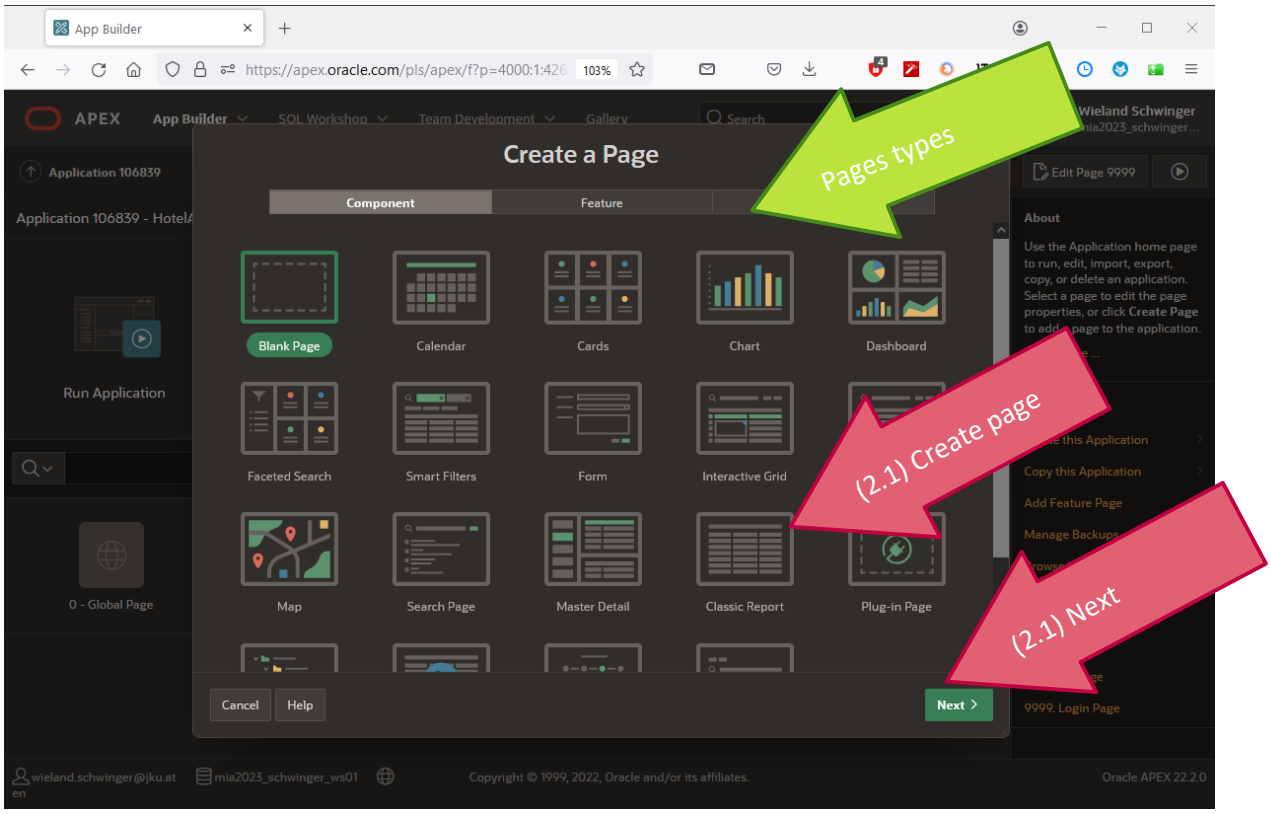
1. Create a page

- We use the **(1.1) "Create Page"-button** to start the page creation wizard.



2. Specify Page Type

APEX offers a series of "built-in" page types to choose from during page creation.



These page types come with standard functionality which is automatically generated with the page. However, the general idea of most is that a page is operating on one table and its data.

The most relevant page types are:

- **"Classic Report":** This can be viewed as a page generating a straight forward list of data from one table. If you just want to display all/some data of a table than this is the one to choose.
- **"Interactive Report"**: This is similar to a "Classic Report" but additionally it offers interaction capabilities along with the ability to alter data in the table.
- **"Form"**: It renders like a form allowing to insert or update a single entry in the table.
- **"Smart Filters"**: Allows for filtering data.

Please consult the additional training material for further details on page types.

For our first page we will select a **(2.1)** **"Classic Report"** to generate a straight forward list of Hotels with **(2.2)** **"Next >"**.

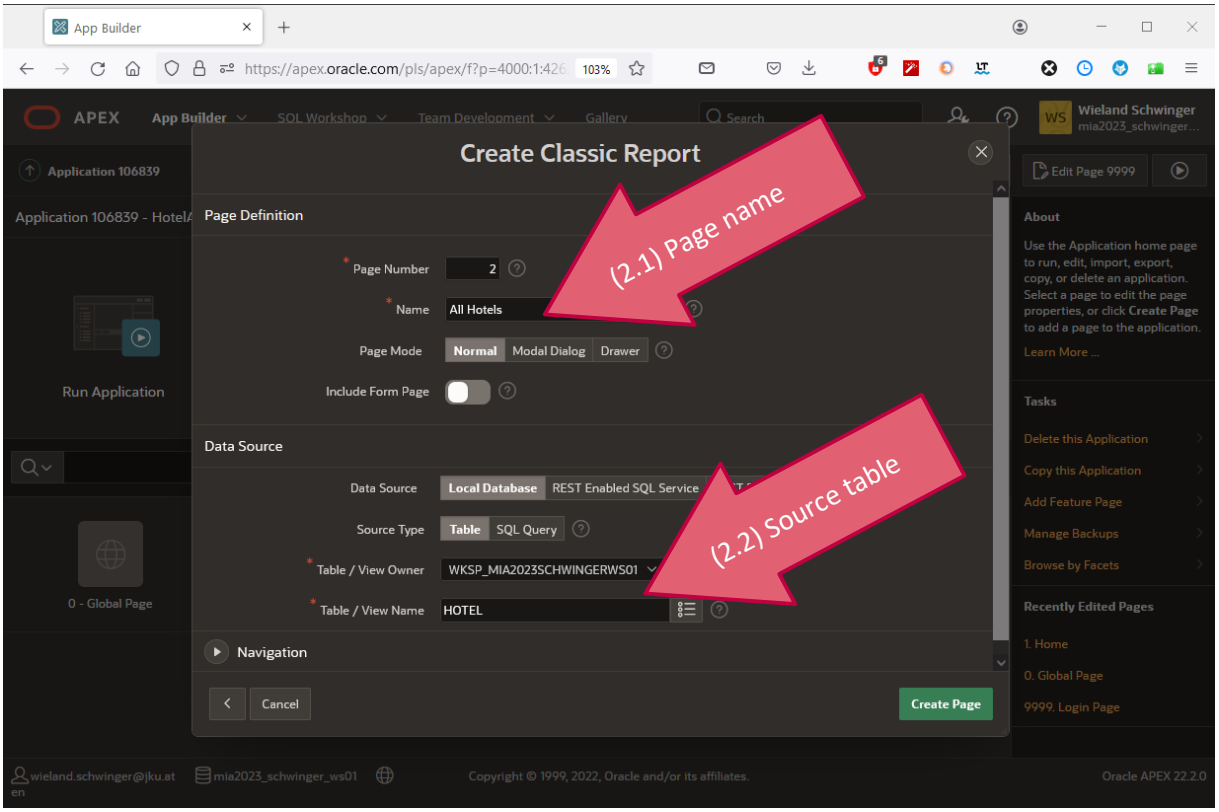
2. Page Definition and Source

Page Definition.
Every page has a **name**. However, internally pages are referenced by their **page number** which must be unique. Furthermore, pages have a "page mode". You can think of it as a "normal" page being a stand alone page vs. e.g. a "Modal Dialogue" is rendered as an "overlay" over an existing page impression, just like a well known modal window.

- We give a page a meaningful **name** but leave the **page numbering** as is.
- Page mode is **normal**.
- To start minimalistic, we **don't select "Include Form Page"** now.
  With "Include Form Page" we are able to specify whether additionally also a form page is generated to insert, update and delete data for the list. Note, that you need to specify this functionality already here to have the functionality generated automatically. Later you need to specify certain aspects manually if you want to achieve insert, update and delete functionality.

Data Source.
Most page types operate on a **table** as source for the data. This is specified under **"Data Source"**.

- To get the data from a local database select **"Local Database"** as **"Data Source"**.
  For the other options please consult the additional training material regarding REST.
- Since we want to directly select the data from a table, use **"Table"** as "**Source Type**".
- The **"Table / View Owner"** specifies the "workspace"; we **leave it as is**.
- Most relevant is to specify the **table** the report is operating on. You can either directly type the table name or use the **listing**-function.



Navigation.

- The **Navigation** section allows to specify how navigation to that page – through e.g. the Web apps menu and breadcrumbs – is set up. We leave these navigation settings as is.

Finally, we create the page with **"Create Page"**.

After creation of the page we are redirected to the **Page Specification**. This Page Specification allows to alter the standard specification of the page. It can be also accessed from overview on the Web app as already seen before.

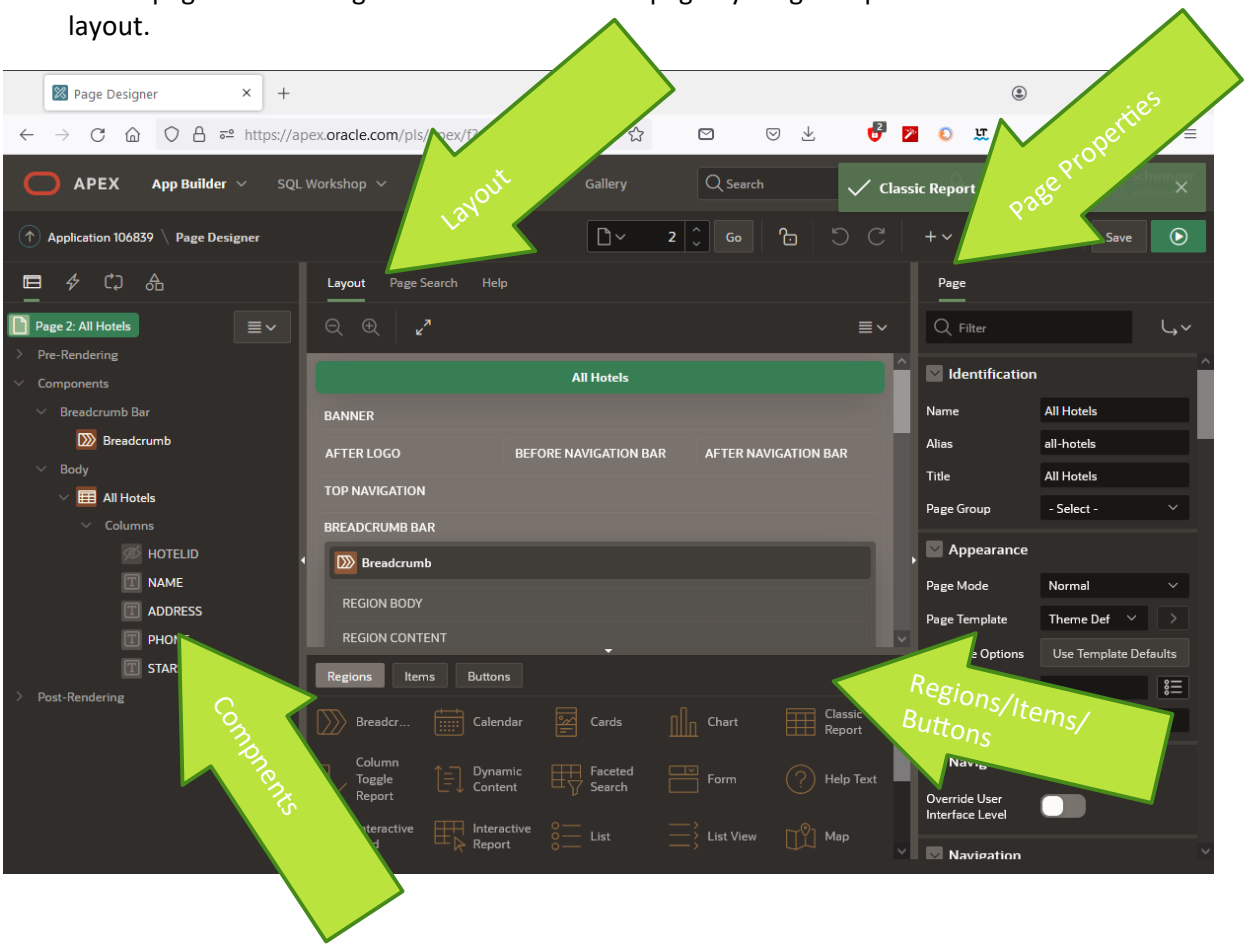
The Page Specification screen offers various panels. The most important functionalities are:

- **Components:** This left-hand pane displays the components of the page.
  Importantly, within "Body" (representing the region in the page where the main "content" of the page is going to be displayed), you can see the "Classic Report"-region, i.e. our "List of all hotels"). The region is the component referring to the table "HOTEL".
  Accordingly, under that region you can see, and configure the columns for that region, i.e. the columns of the underlying table.

- **(Page) Properties:** Here you see and can alter various aspects (e.g. appearance, navigation) of that page as currently the complete page is selected.
  However, this right hand panel is changing as you select different components of the page allowing for component specific configurations.

- **Layout:** It comprises an abstract view on the layout (composition) of the page. It specifies herein the page skeleton certain components are placed and thus displayed.

- **Regions / Items / Buttons:** These are the available elements which can be used and placed on the page. It allows e.g. to add a Button to the page by drag&drop that element into the layout.



For now we stick with the automatically generated page.

6) Running the Web App / Running the PAGE

You can **"run"** directly out of APEX. It basically opens a page / the homepage of your Web app in the Cloud in your browser.

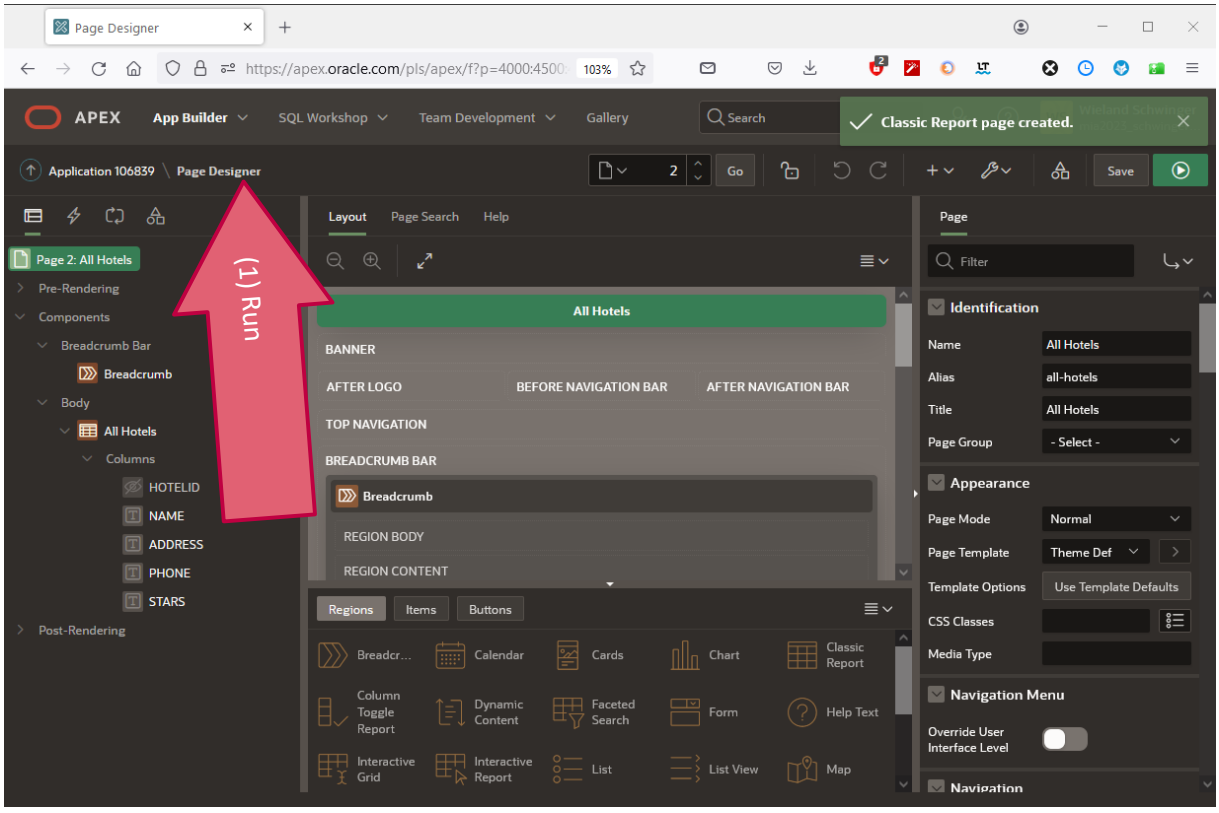
There are various ways to "run" your Web app.

- One way is to start from the **Overview on the Web app** by **"Run Application"** (see previous) where you can "run" your Web app from the home page or select a specific page to "run" your Web app (cf. ▶-Symbol for a page).
- Alternatively, you can "run" your Web app, i.e. render that specific page, from the **Page Specification** (cf. below).

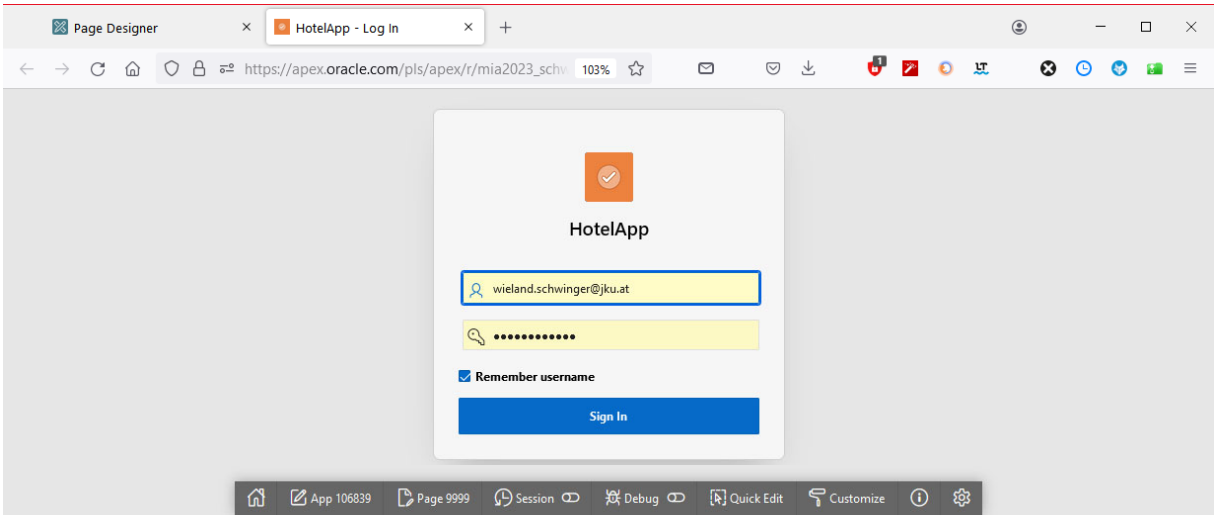
For more information on how to "run" a Web app please refer to the additional training material.

6.1 Run Page

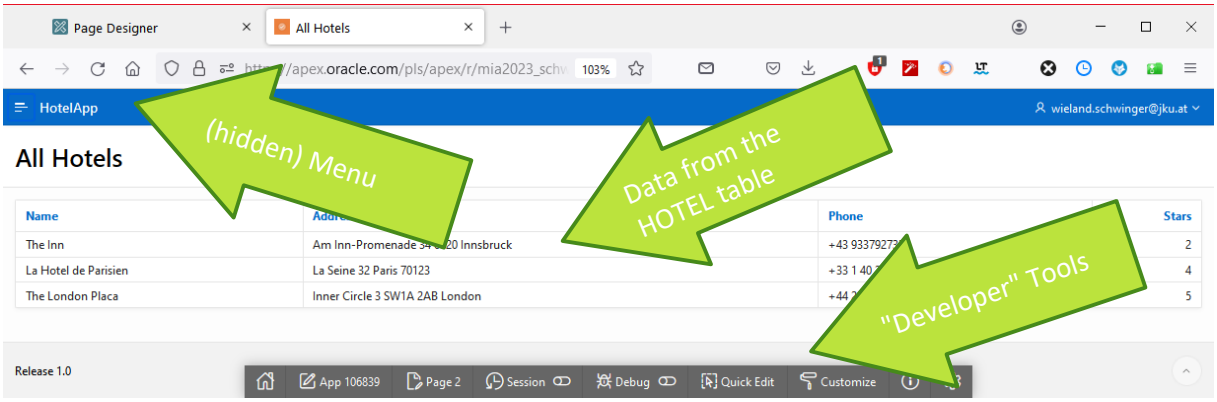
- To simply continue on with our tutorial, we use the ▶-Symbol on the **Page Specification** to start your Web app.



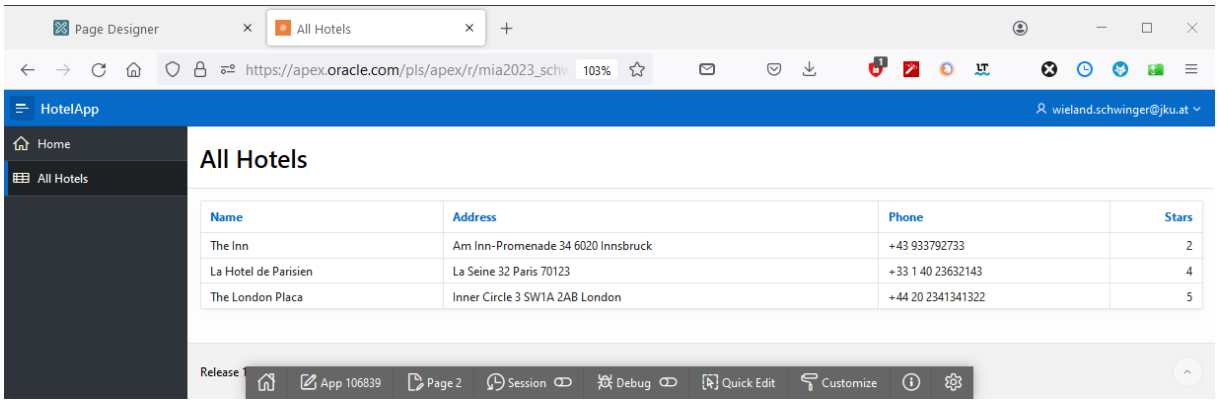
Accessing the page, as standard, requires to login. Use your ORACLE APEX login for this.



After login you are seeing your first Web app generated with ORACLE APEX.



You can "open" the **menu** of the Web app where you can see that the page of "All Hotels" is accessible through the menu for any page of the Web app.

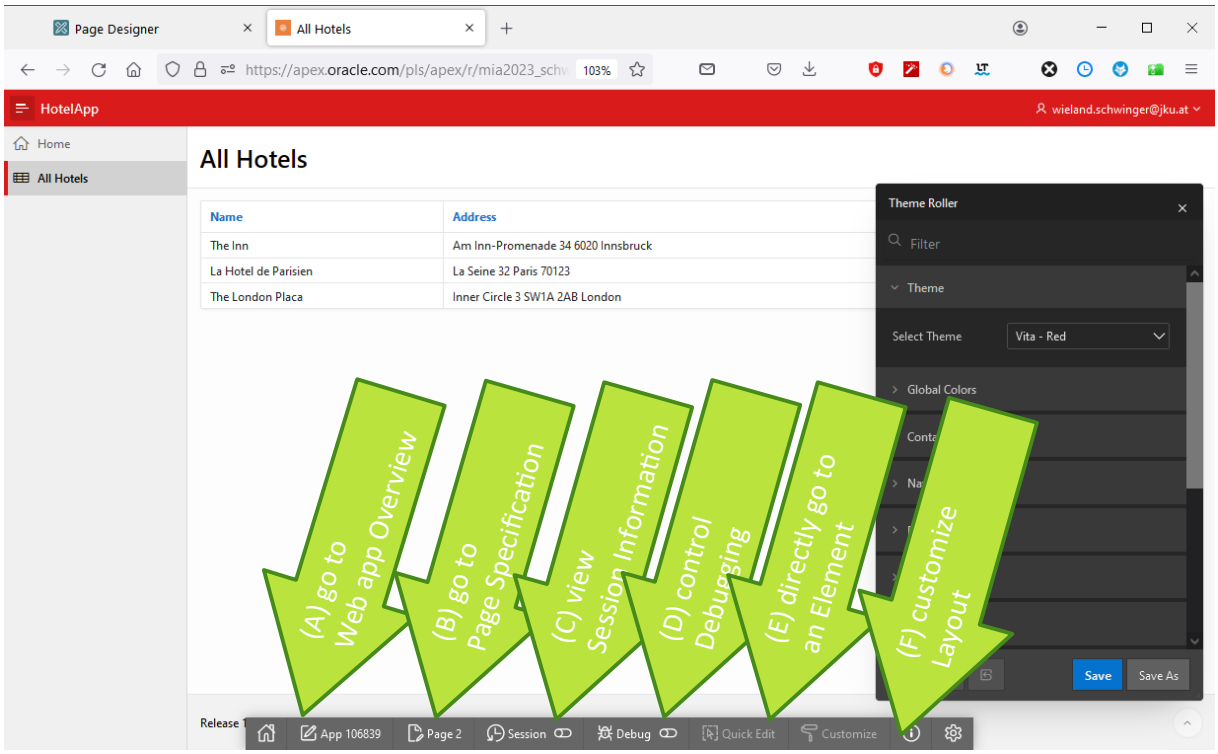


7) "Developer" Options on a PAGE

On the page you just displayed you see a kind of **"Developer Tools"**-bar. This allows you to invoke conveniently certain functionalities.

- **(A)** With **"App <AppNo.>"** you can go to the **Overview on the Web app** allowing you e.g. to add more pages.
- **(B)** With **"Page <PageNo.>"** you can go to the **Page Specification** allowing you e.g. to add or remove the information to be displayed on the page.
- **(C)** With **"Session"** you can see and alter session-local information which is helpful for e.g. seeing the state of variables in the page.
- **(D)** With **"Session"** you can turn on debugging which is helpful for e.g. debugging dynamic aspects of your page.
- **(E)** With **"Quick Edit"** you can select certain parts of the page and jump directly to the **Page Specification** of that specific element on the page.
- **(F)** With **"Customize"** you can easily change visual aspects of the page, e.g. the colouring.

During our initial development changing between the Page Specification and the rendered Page through the **"Developer Tool"-pane** might be very convenient.

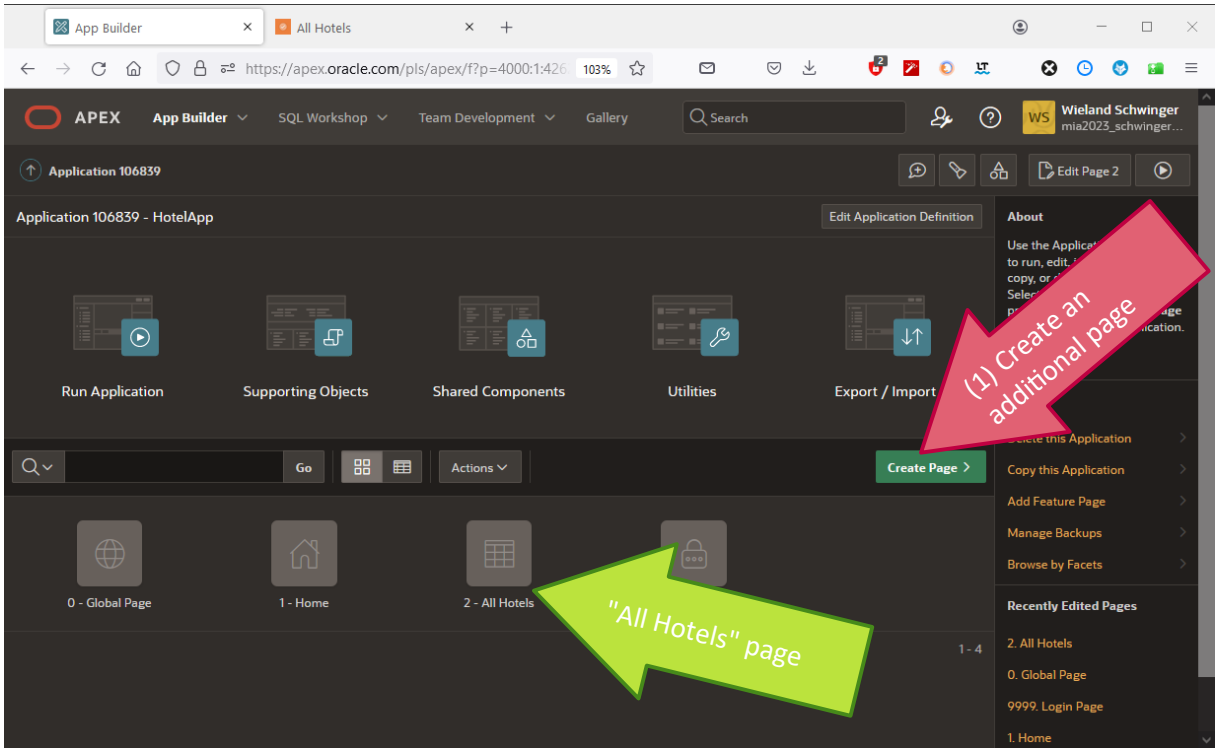


8) PAGE with Interaction

To add functionality to our Web App to be able to insert/update/delete Hotels, we generate an additional page for Hotels but this time with interaction.

1) Add an Additional Page

- Go to the **Overview of the Web App**, e.g. by using the **"Developer Tool".**
- Create an additional page (cf. **"Create Page"**).



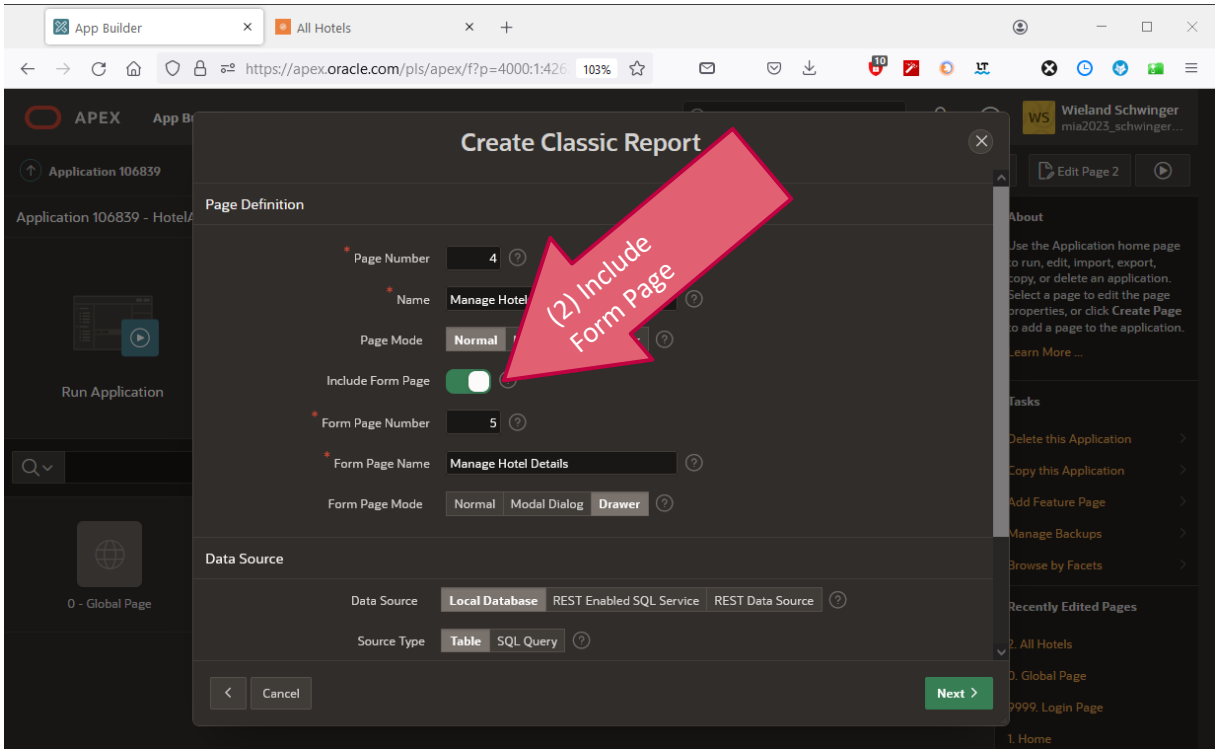
2) Create Classic Report with Interaction

- Use the wizard as shown before to generate a **"Classic Report"** based on the table **HOTEL**.

- **However**, this time …

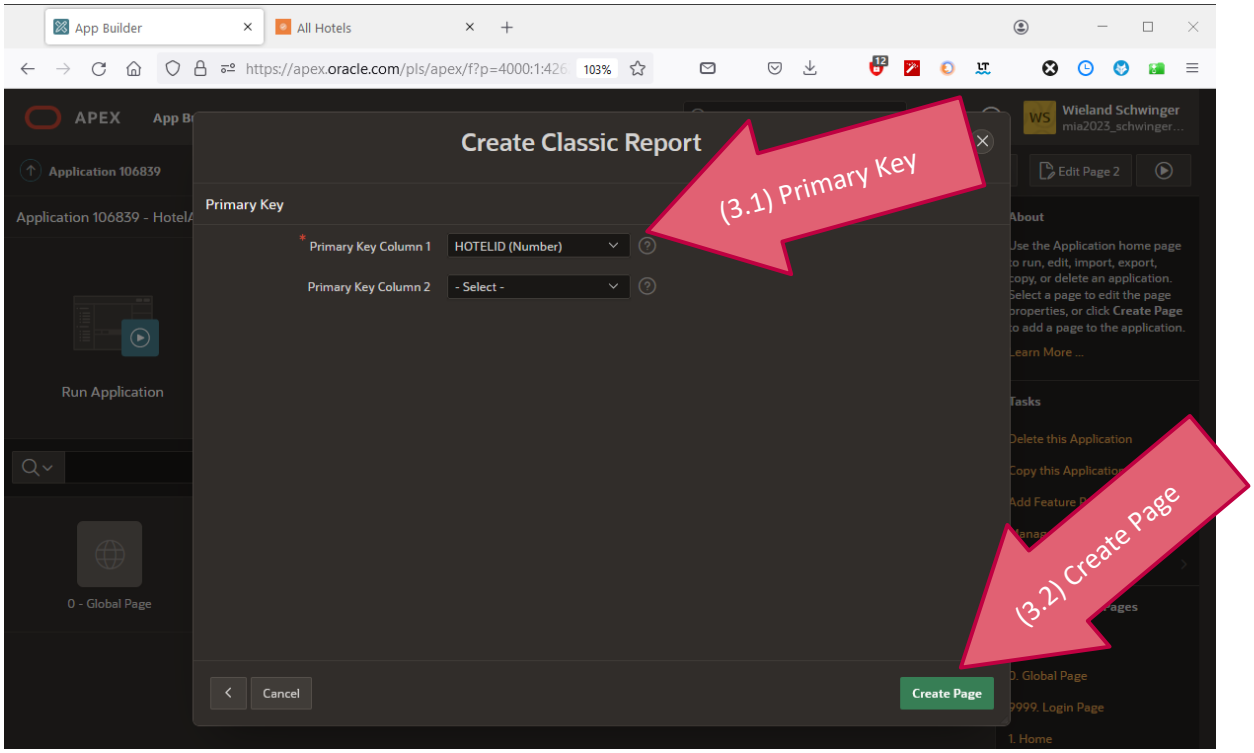
  - **Name** the page **"Manage Hotels"**
  - Set **"Include Form Page"**
  - **Name** the **"Form Page Name"** **"Manage Hotel Details"**
  - Leave the **"Form Page Mode"** set to **"Drawer"**

- As **Data Source** again specify the **local database table "Hotel"**.



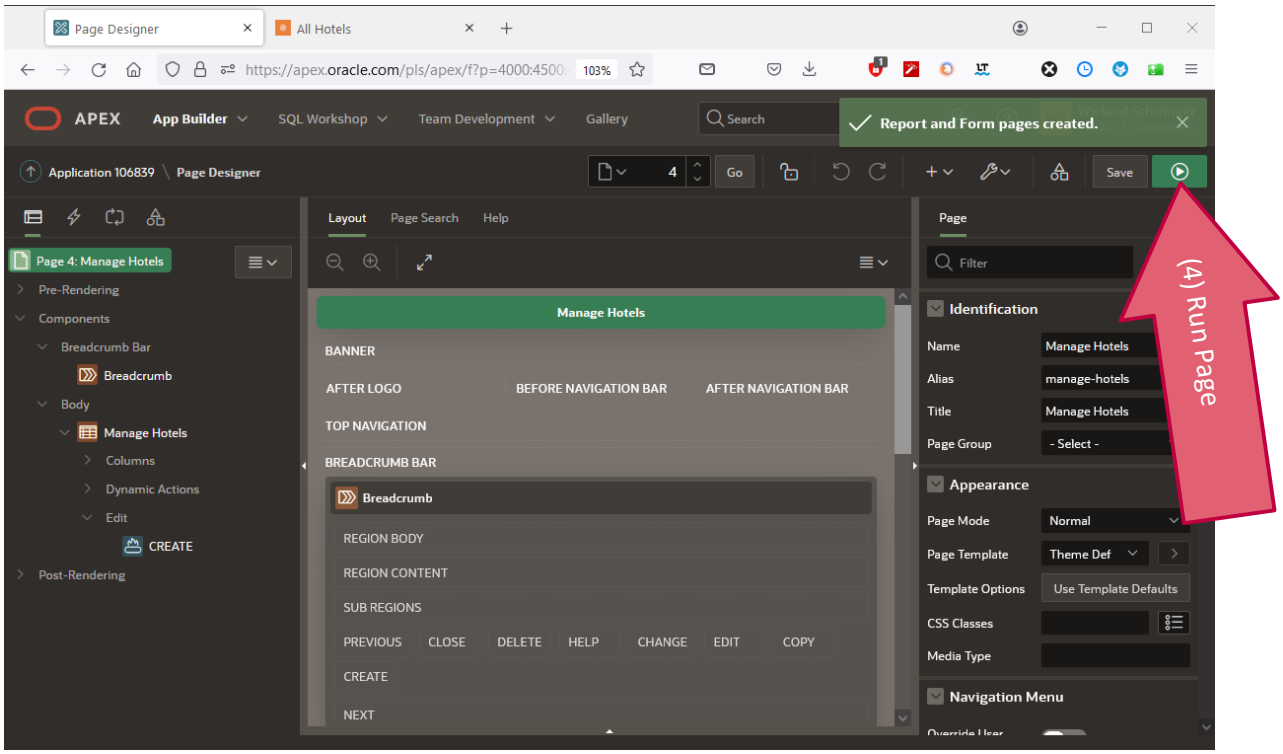
3) Specify Primary Key for Interaction

- The default **primary key** of the **"Hotel"** table is already filled in as **"Primary Key Column 1"** – we leave it like it is **(3.1)**.
- **Create** the page **(3.2)**.



3) Run Page

We land at the **Page Specification** from where we can directly **"run"** the page.

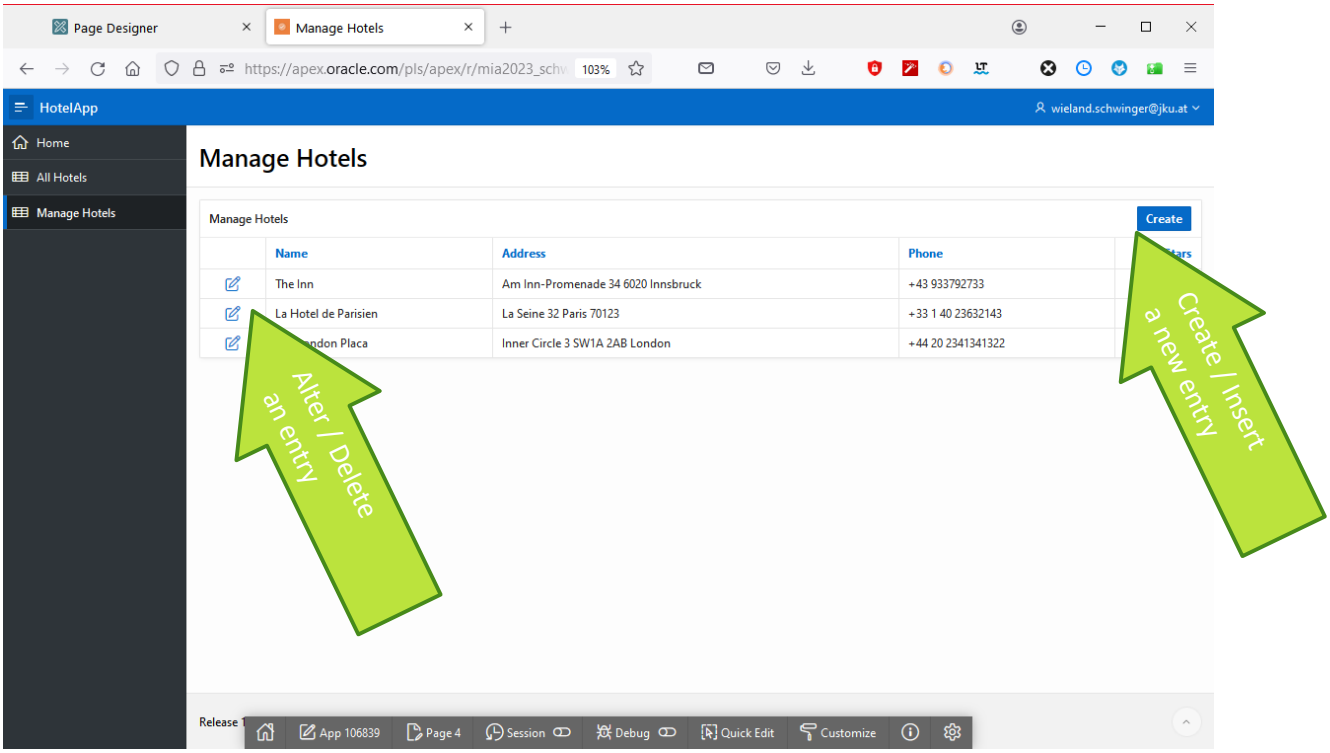


4) Generated Page with Interaction

When you run the page you can see that **additional Elements** have been created for our new "Manage Hotels" page.

The generated page now contains a **"Create"**-button which allows your Web app now to create/insert data into the "Hotel" table. For altering / deleting a hotel from the "Hotel" table an additional icon (**"Edit Icon"**) is rendered in front of every row in the list.

When you select the "Edit Icon" in front of an entry the edit dialogue is opened, since we have specified **"Drawer"** it opens a form to interact with from the side on our page.



NOTE:
Depending on your window size the "Delete" / "Cancel" button of the Drawer might be covered by the "Development Tools"-bar.

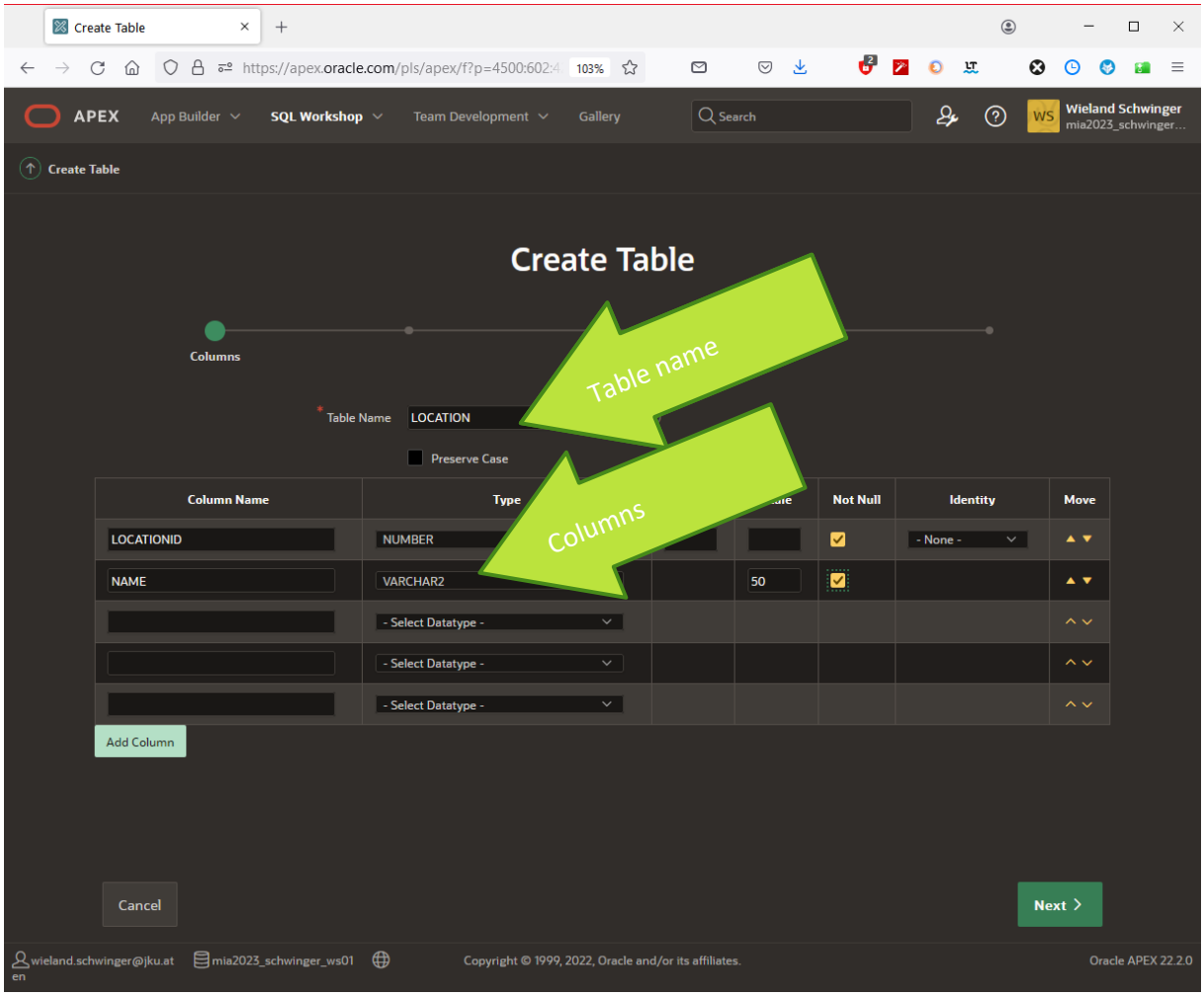
9) Relating DATA

After having created a single independent table we now look how to realize **relationships** at the **Data Level**. Since ORACLE APEX roots in relational databases, the Data Level is realized in terms of the **relational model**.

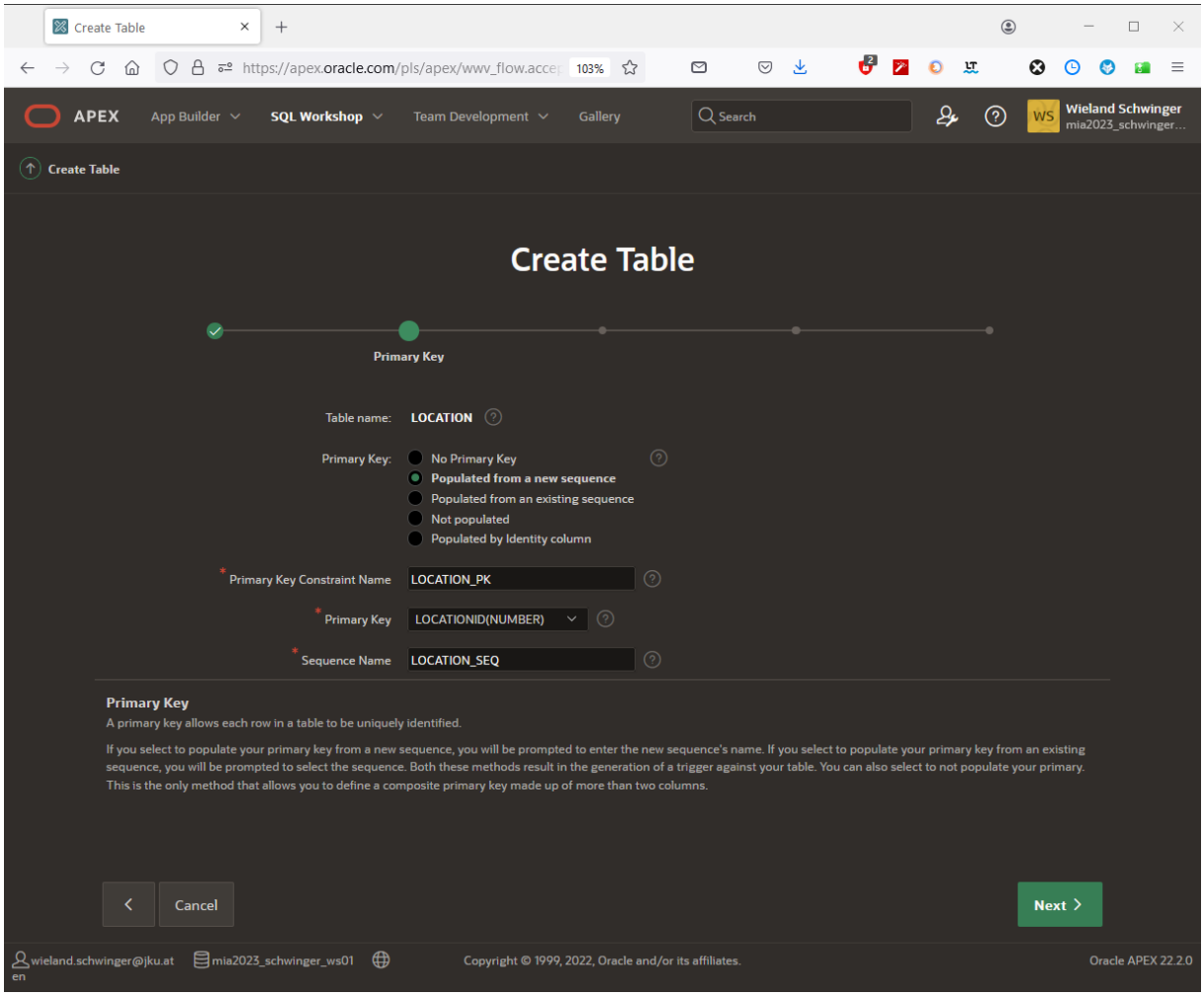
Thus, we need to learn how to deal with **1:n-Relationships** – that are realized through Foreign Keys at the Data Level in ORACLE APEX and **n:m-Relationships** – that are realized through mapping tables.

1) Adding a Table

- Go to the **Object Browser** in the **SQL Workshop**.
- **Create an additional table (cf. "+").**
- Use the **Table Creation Wizard** to create a table **"Location"** as described above.
  - Table name: **Locations**
  - Columns:
    - **LocationID** NUMBER NOT NULL
    - **Name** VARCHAR2 50 NOT NULL

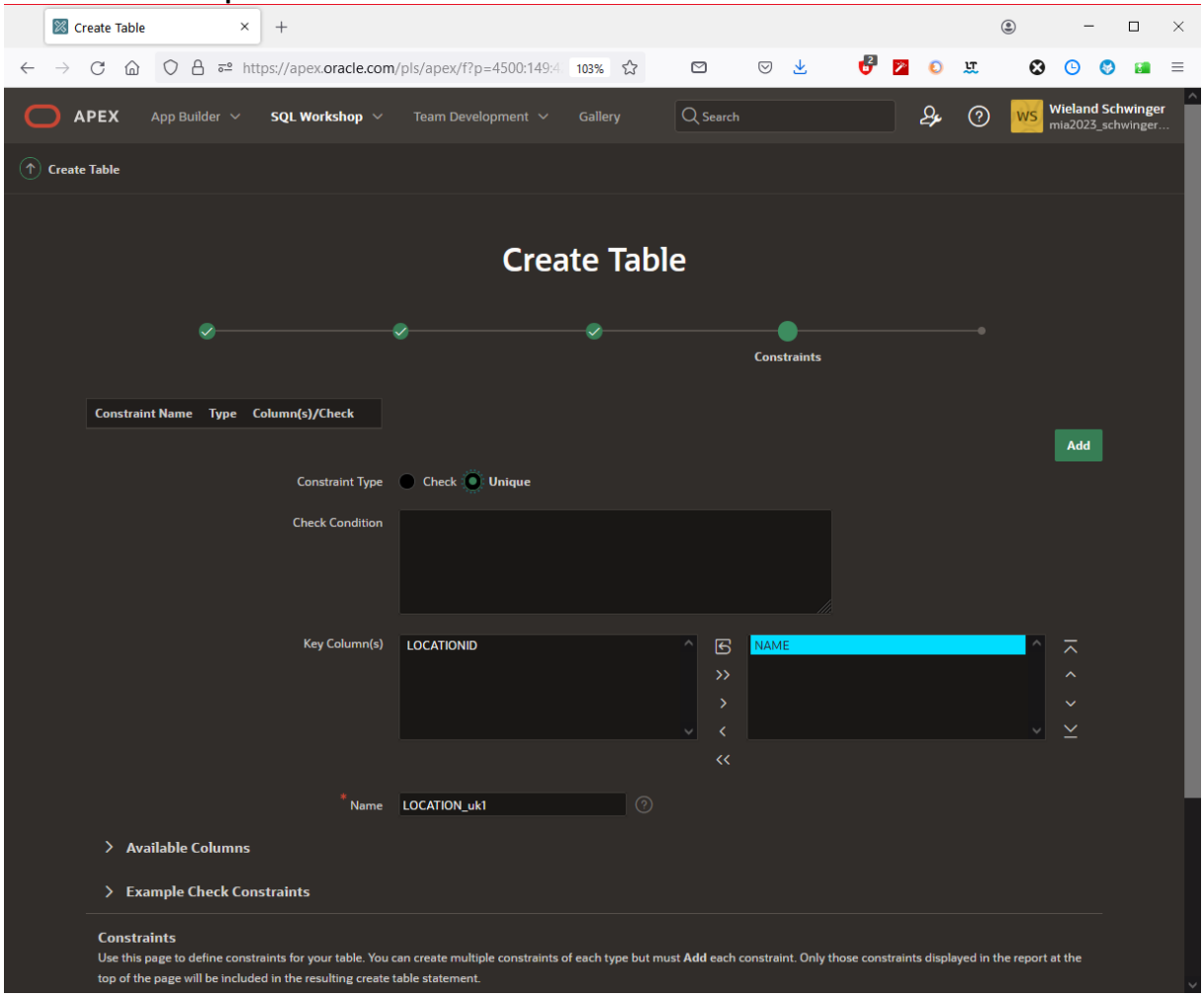


- Set the **Primary Key** to be the **LocationID** being populated from a **sequence** as shown before.



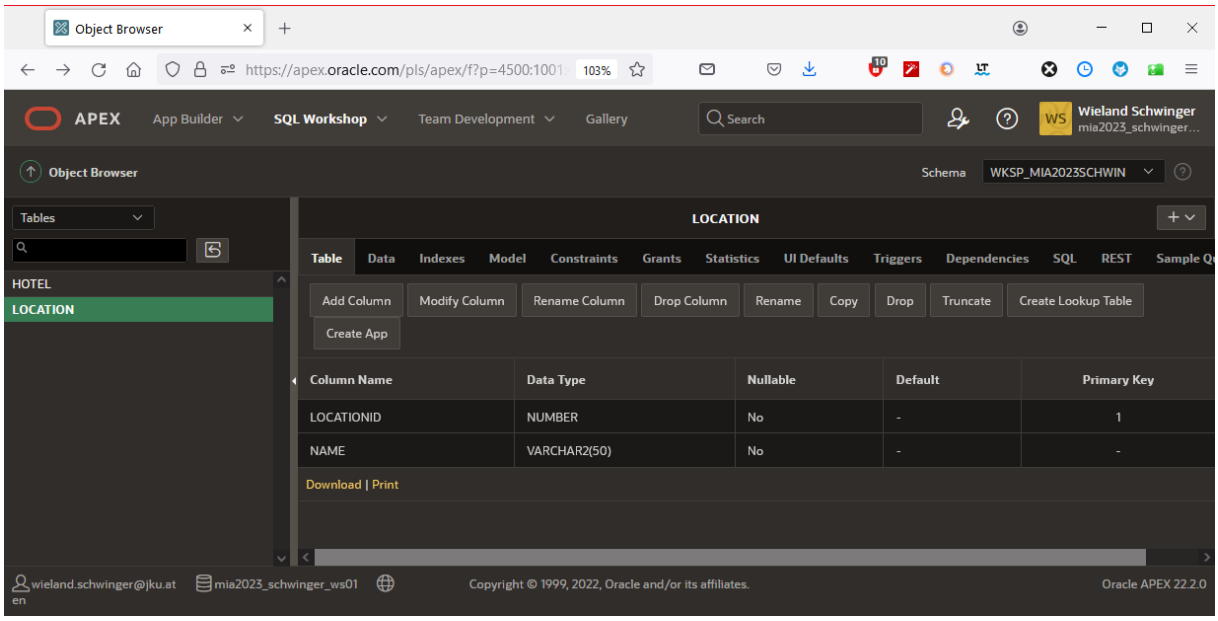
- No Foreign Key needs to be set for that table. Just go **"Next >"**.

- **Add** a **Unique Constraint** on the **"Name"** column.

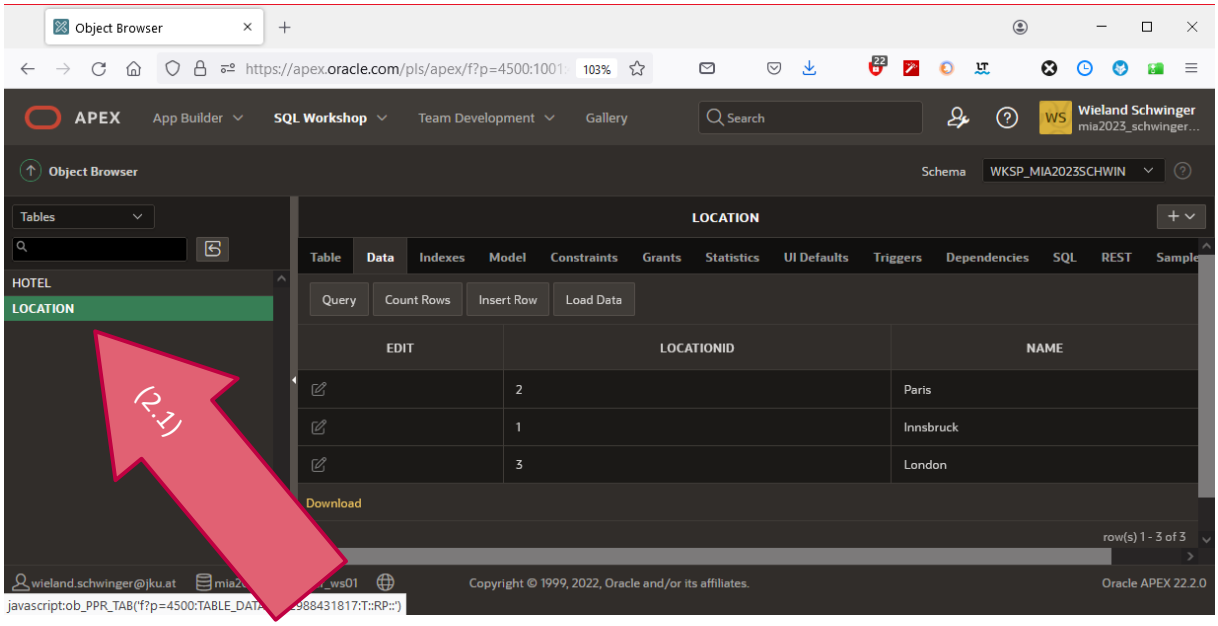


2) Adding Data to Table

- Go to the **Object Browser** in the **SQL Workshop**.
- **Select** the **"Location"** Table **(2.1)**.

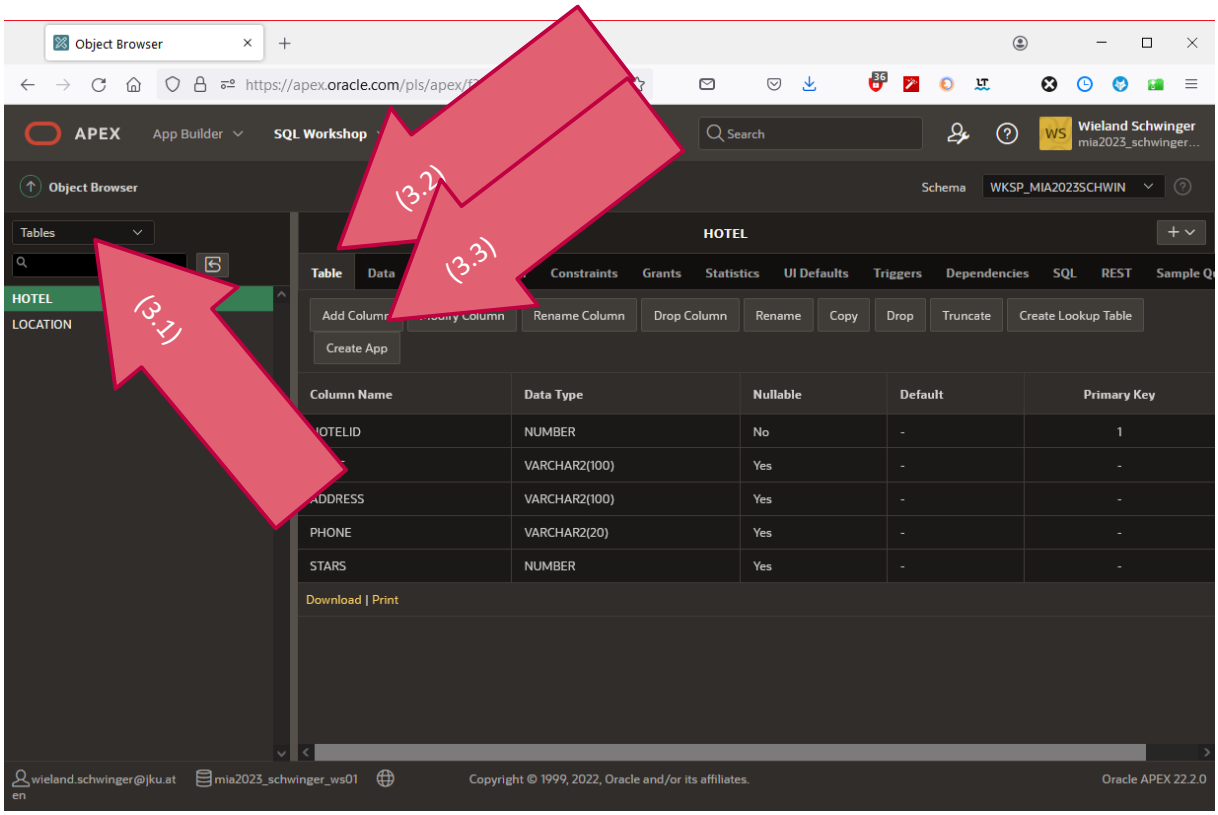


- Add some entries to the table.



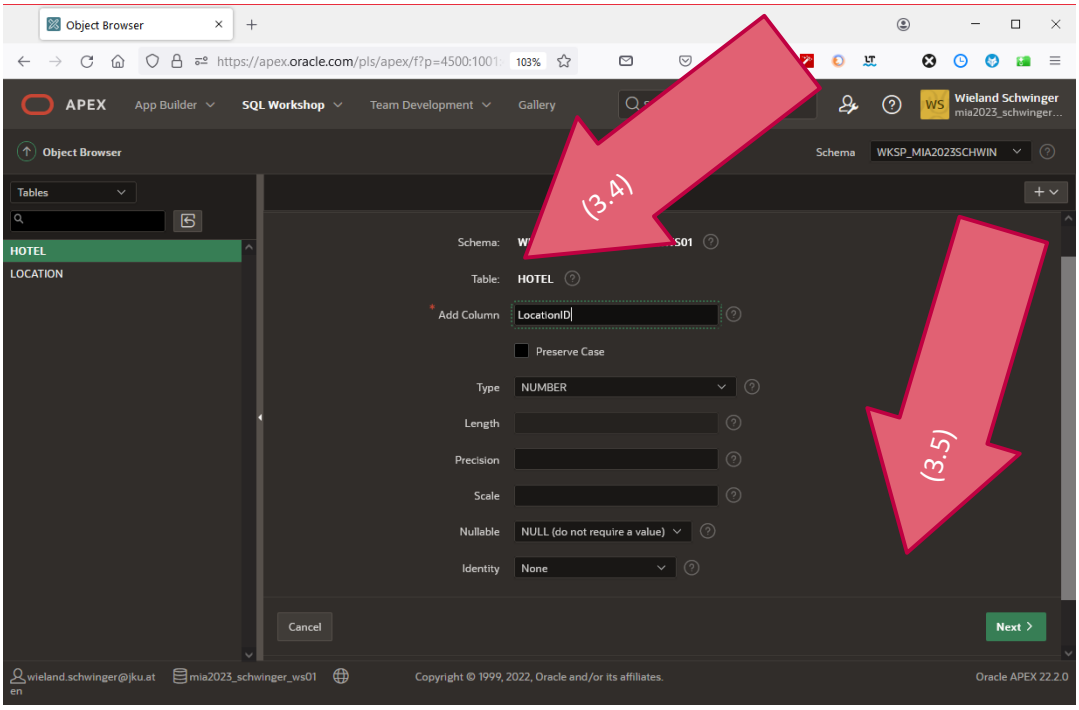
3) Adding the Reference Column to the table "Hotel"

- Go to the **Object Browser** in the **SQL Workshop**.
- **Select now** the **"Hotel"** Table **(3.1)**.
- We add now a column to the table to refer to the location a hotel is located at. For this select the **"Table"-tab (3.2)** and use **"Add Column" (3.3)**.

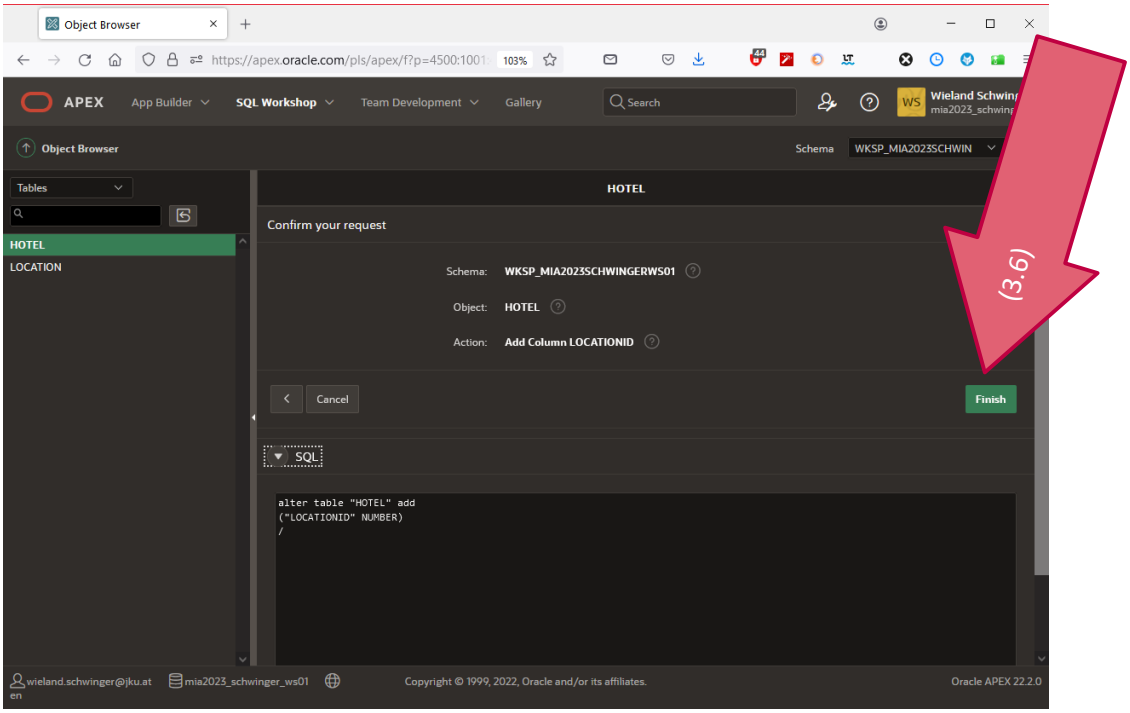


- Add the column **(3.4)**:
  - `LocationID  NUMBER`

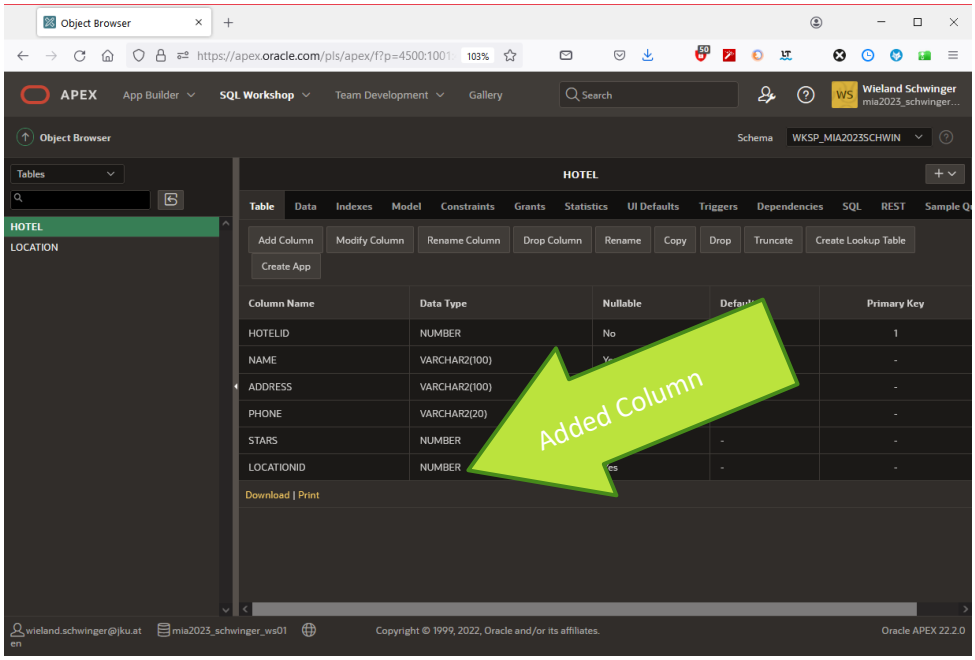
- **"Next >" (3.5)**



- **"Finish" (3.6)**



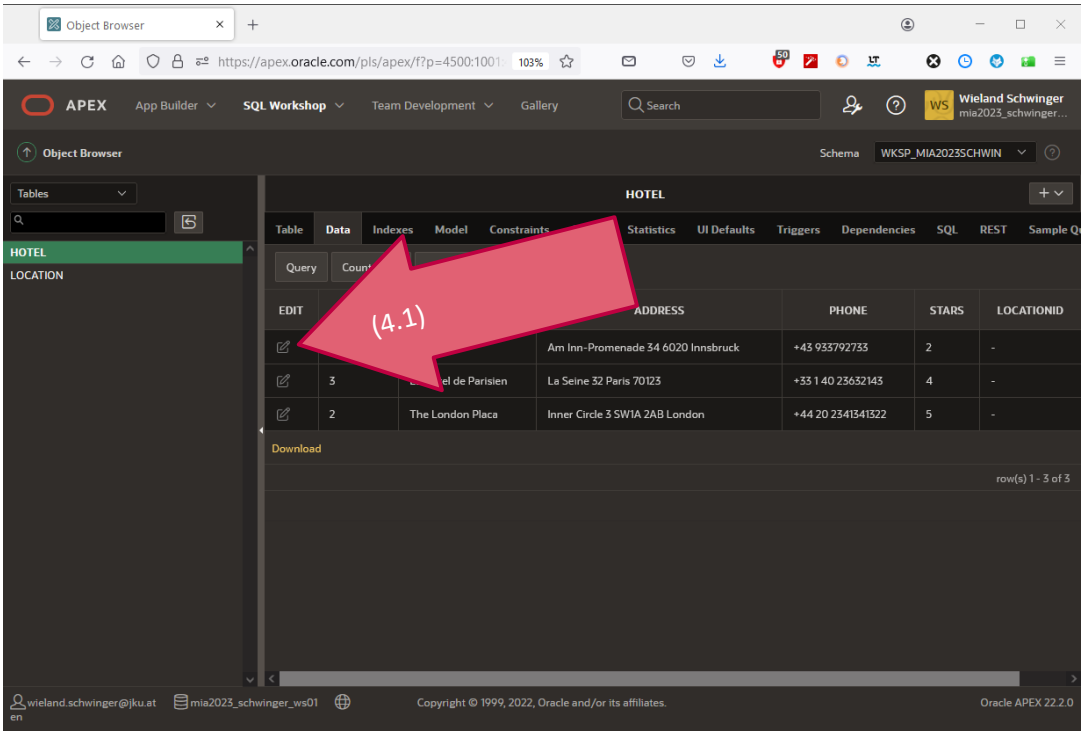
The table has now an additional column.

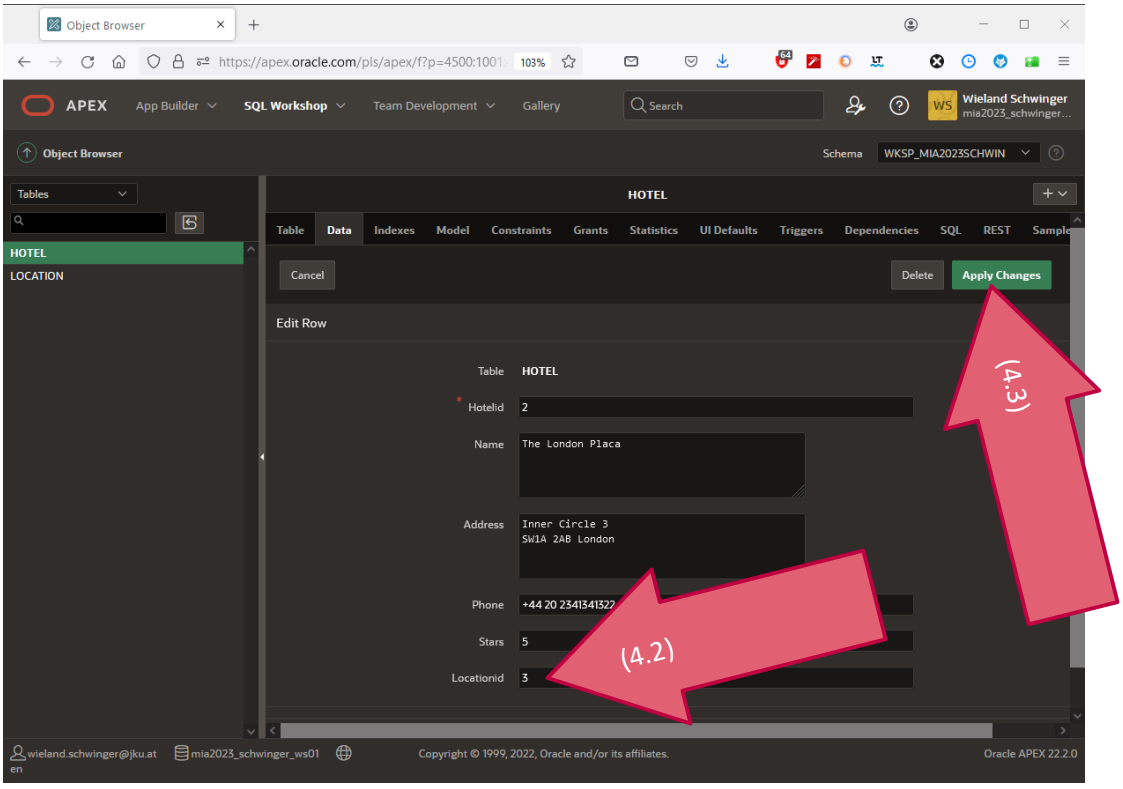


4) Filling the Column

Before we are able to specify that every hotel needs to refer to a location and accordingly add that constraint to the "Hotel" table, we need to fill the data for the column.

- **SQL Workshop → Object Browser → select "Hotel"**
- **select "Data"**-tab.
- **For each row** in edit that entry by clicking on the **"edit"-symbol (4.1), insert a valid LocationID-value (4.2)** and **"Apply Changes" (4.3)**.

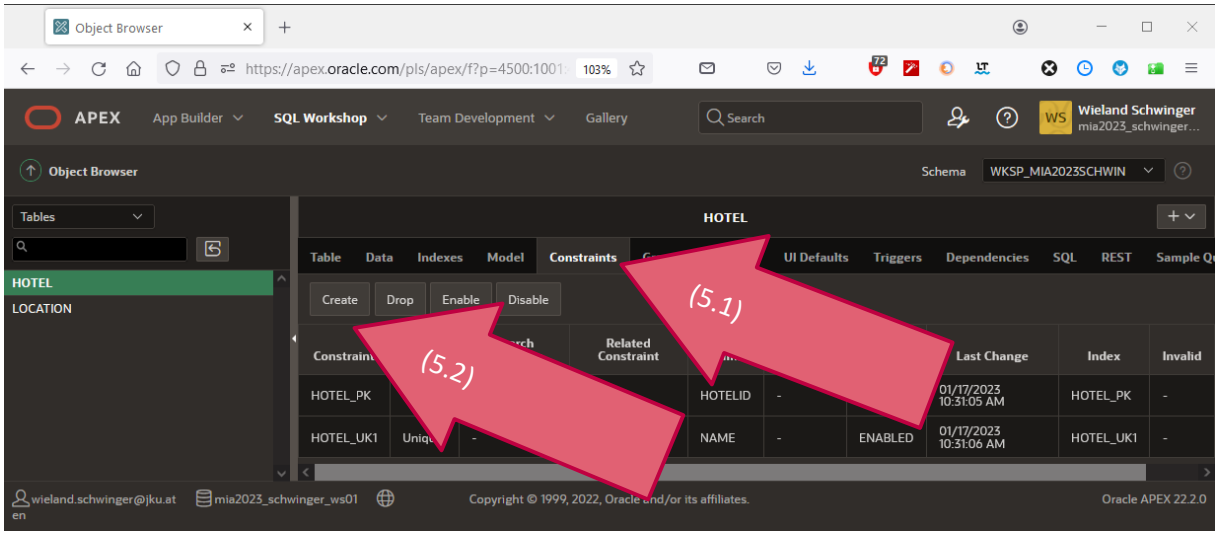




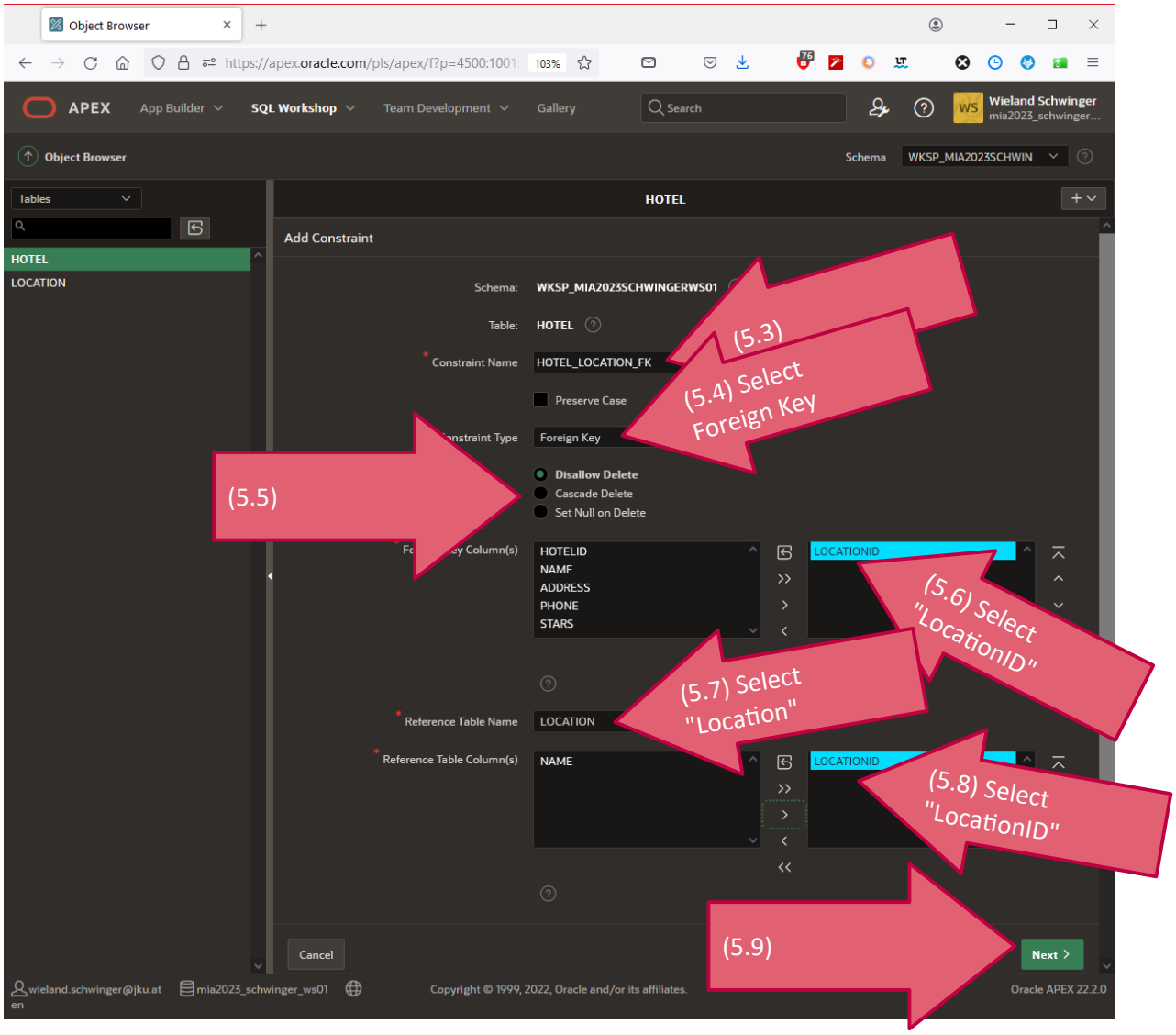
5) Add Foreign Key Constraint

Finally, we need to specify that the LocationID column of the "Hotel" table is actually a Foreign Key column.

- **SQL Workshop → Object Browser → select "Hotel"**
- **select "Constraint"**-tab **(5.1)**.
- **select "Create"** to create a **new Constraint**.

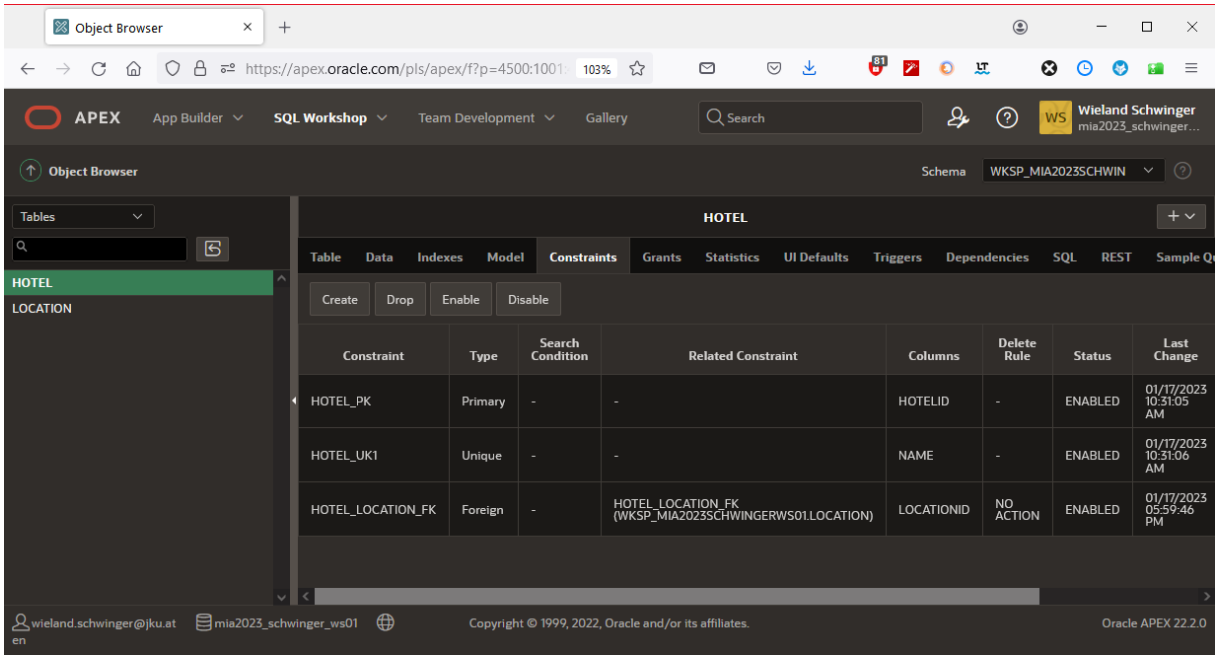


- **Name** the **Foreign Key Constraint** accordingly **(5.3)**.
- Set **Constraint Type** to **"Foreign Key" (5.4)**.
- Leave the **Deletion Semantics** to **"Disallow Delete" (5.5)**.
- For the **Foreign Key Column(s)** select **"LocationID" (5.6)** by using **>-Symbol**. This is the Column from which is referenced.
- Select for the **Referenced Table Name** the **"Location"**-table. This is the table to which is referenced **(5.7)**.
- Under **Reference Table Column(s)** select **"LocationID"** by using **>-Symbol**. This is the column to which is referenced **(5.8)**.
-



- **Next >** **(5.9)**

- After **Finish** on the next screen, the **Foreign Key Constraint** is added.



NOTE:
Of course it would have been easier if we would have anticipated the Foreign Key Column from the very beginning.

In case we would have created the table "Location" (the table to be referenced) prior to creating the table "Hotel" (the table referencing) we could have defined the Foreign Key Constraint already during the creation of the table "Hotel".

The principle is the same for tables realizing relationships. In that case both tables to be related should be created first before the table realizing the n:m-relationship is created.

However, it demonstrates how to proceed if a table to be referenced has not yet been created or we need to make changes to the tables later on.

10) Completing DATA Model

After having created a single independent table and related it to another table through a foreign key we now want to complete our sample data model. Thereby, we will see how **n:m-Relationships** – that are realized through mapping tables in the Data Level.

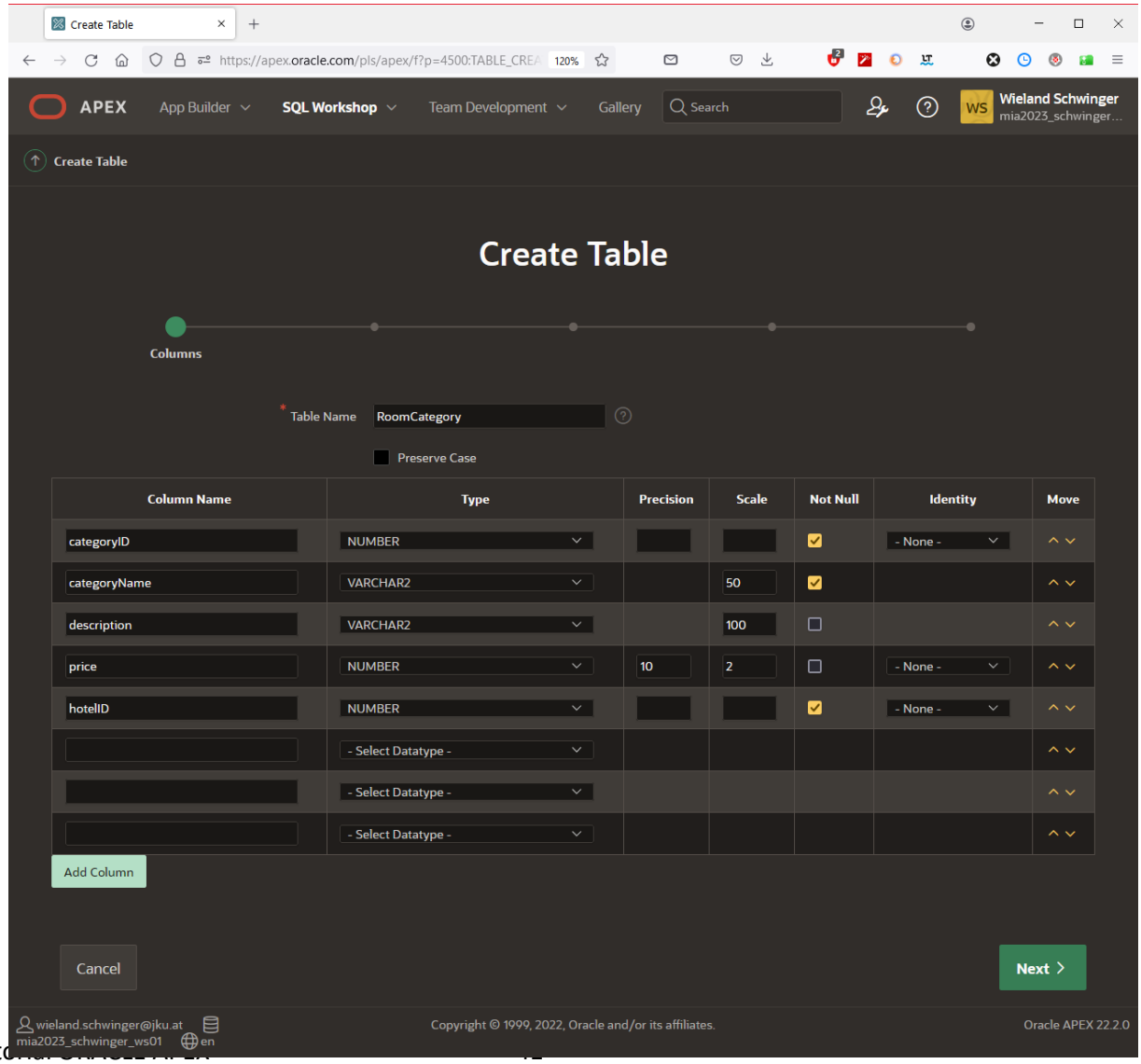
We need to create the following tables using the **Table Creation Wizard** of APEX **Oracle Browser**:

1) Add Additional Table "RoomCategory"

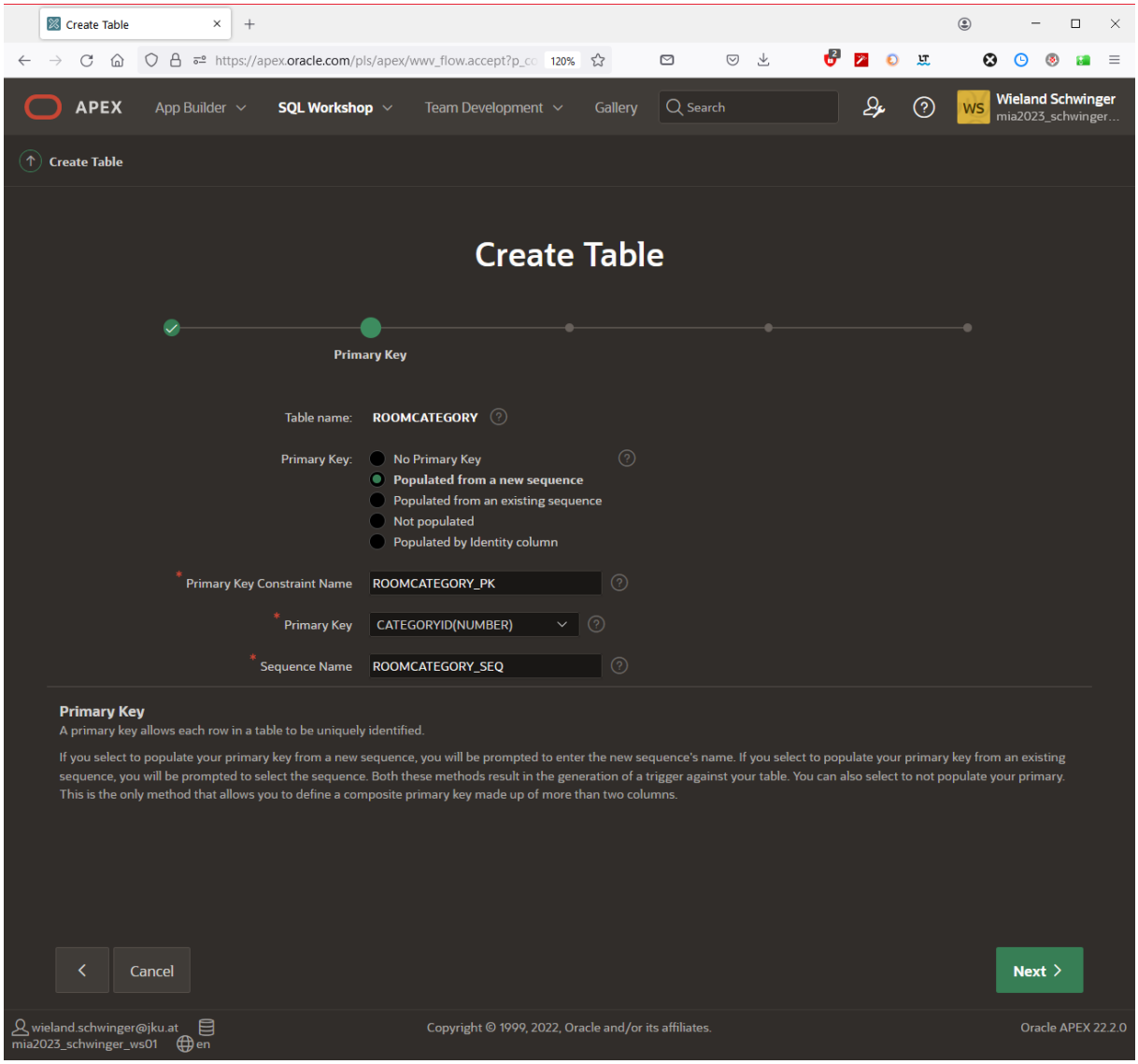
We need to add some additional tables. Use the **Table Creation Wizard**.

- **Create Table and Attributes**
  - Table name: **RoomCategory**
  - Columns:
    - **categoryID** NUMBER NOT NULL
    - **categoryName** VARCHAR2 50 NOT NULL
    - **description** VARCHAR2 200
    - **price** NUMBER 10 (Precision) 2 (Scale)
    - **hotelID** NUMBER NOT NULL

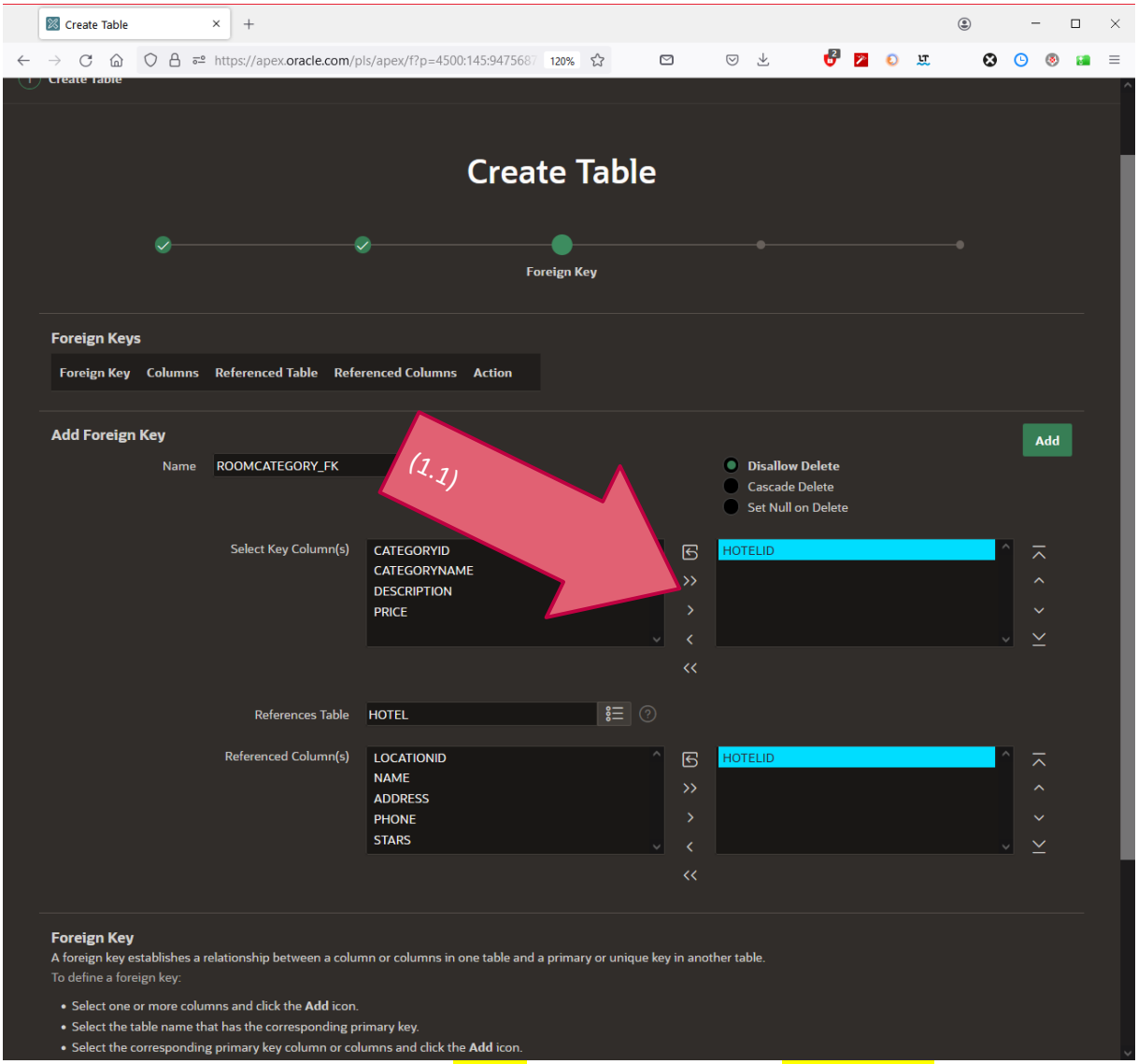
  Note that, **hotelID** will serve to reference the Hotel as a foreign key.



- Set the **categoryID** as **"PrimaryKey"** to be populated from a new sequence.



- **Set hotelID as Foreign Key.**
  - **Use the >-Symbol (1.1)** to select the column of the table which holds the foreign key under **"Select Key Column"**.
  - **Use the >-Symbol to select the column of the table being referred to (table "Hotel") which contains the referred to primary key under "Select Referenced Table"** and **"Referenced Column(s)"** respectively**.**



- We **skip** any further **constraints** (**Next >**)and **create** the **Table** (**Create Table**).

2) Add Additional Table "Guest" and "Reservation"

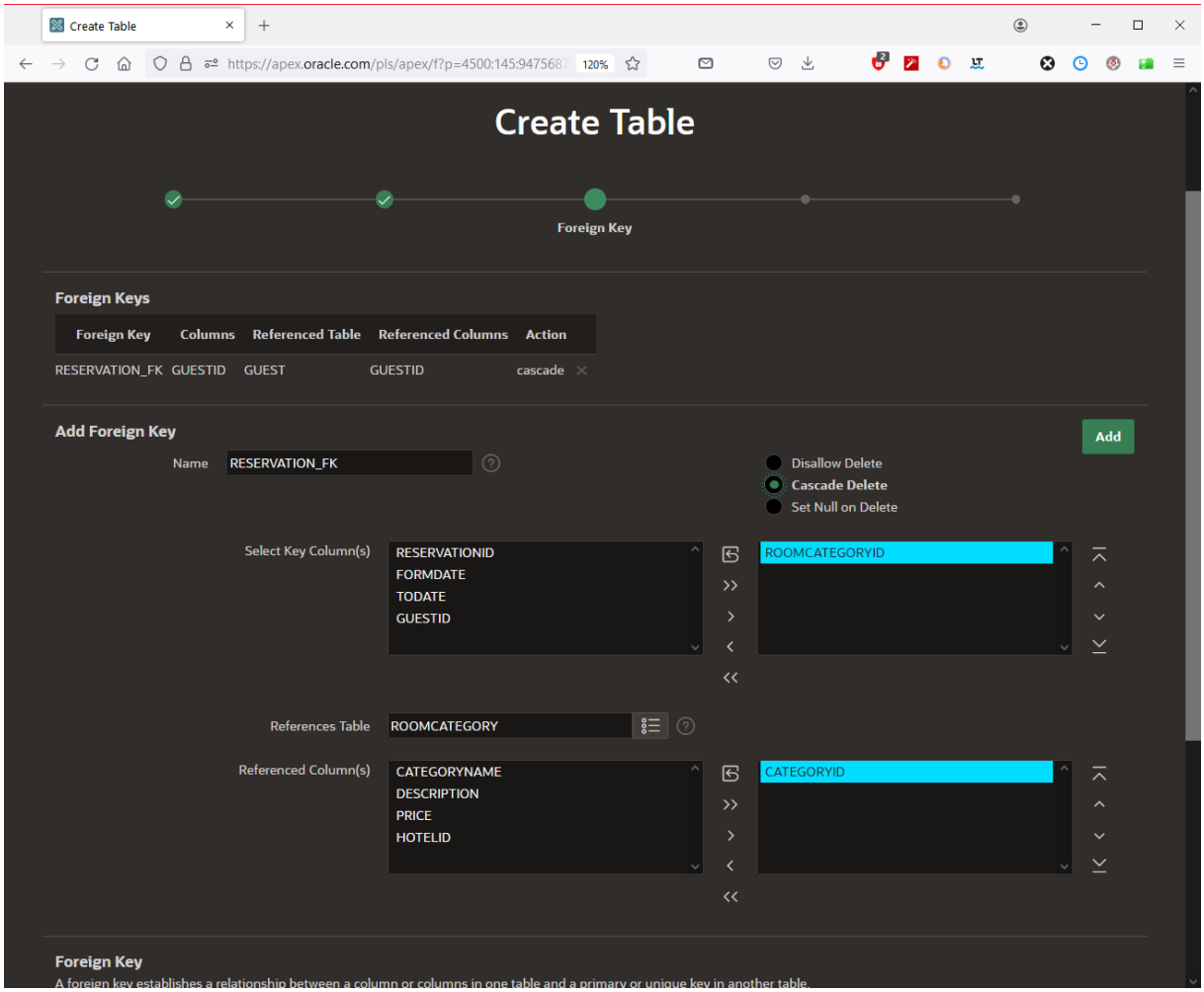
- In the same way as before create two additional tables. First, and this is important, create the table **"Guest"**. After that create the Table **"Reservation"**.
  Use the **Table Creation Wizard** again.

  - Table name: **Guest**
  - Columns:
    - **guestID** NUMBER NOT NULL
    - **surName** VARCHAR2 100
    - **firstName** VARCHAR2 100
    - **phone** VARCHAR2 30
    - **eMail** VARCHAR2 100

- Now create the table **"Reservation"**.

  - Table name: **Reservation**
  - Columns:
    - **reservationID** NUMBER NOT NULL
    - **fromDate** DATE
    - **toDate** DATE
    - **categoryID** NUMBER
    - **guestID** NUMBER

  - We want to specify two foreign keys.
    - **Note** that, we want to set both foreign keys as **"Cascade Delete"**.
      This specifies that if either, a guest or a room category is deleted, thus the related reservations of that guest or of that room category are deleted as well.
    - Specify that, **categoryID** is a foreign key referring to table **RoomCategory** and the column **categoryID** like before in the tutorial.
    - Do the same for **guestID** to refer to the table Guest and the **guestID** column in that table. **Reservation** thus allows to have a singe **Guest** having multiple reservations of one and the same **RoomCategory**. In that sense Reservation acts like a table realizing (with additional attributes) a **n:m relationship** between **Guest** and **RoomCategory**.



3) Add table FacilityType

- We want to add a table FacilityType representing the different facilities, e.g. Spa, Pool, Playground, Parking, WiFi.

  - Table name: **FacilityType**
  - Columns:
    - **facilityTypeID** NUMBER NOT NULL
    - **facilityName** VARCHAR2 100 NOT NULL
    - **facilityDescription** VARCHAR2 1000

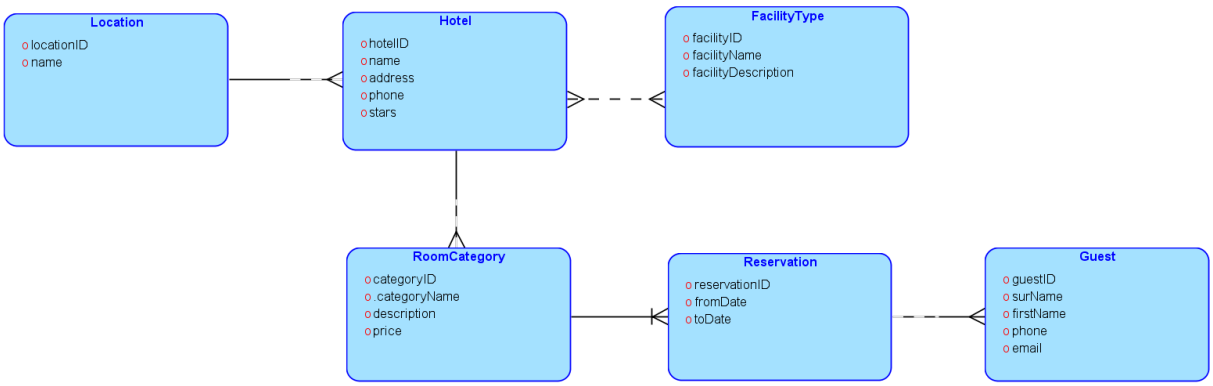
  - Make **FacilityTypeID** the **primary key** and have it populated as learned before.

4) Adding n:m Relationship

- Add mapping table between Hotel and **FacilityType** to allow for a **n:m Relationship** between the two. We will call that "Mapping"-table for realizing the n:m Relationship **"HotelFacilities"**.
  - Table name: **HotelFacilities**
  - Columns:
    - **hotelFacilityTypeID** NUMBER NOT NULL
    - **hotelID** NUMBER NOT NULL
    - **facilityTypeID** NUMBER NOT NULL
  - Again specify **hotelFacilityTypeID** a **primary key**.
  - As before for **Reservation**, specify that, **hotelID** is a foreign key referring to table **Hotel** and the column **hotelD.**
  - Do the same for **facilityTypeID** to refer to the table **FacilityType** and the **facilityTypeID** column in that table.
  - For both foreign keys again specify **"Cascade Delete"**

5) Overview on Data Layer

Here a final overview on the Data Layer using an ER-Diagram.



Note that, for realizing **1:n-Relationships** we need to have the respective Foreign Keys as additional attribute in a table. For realizing **n:m-relationships** we need to create respective mapping tables containing the two foreign key – from each of the involved tables.

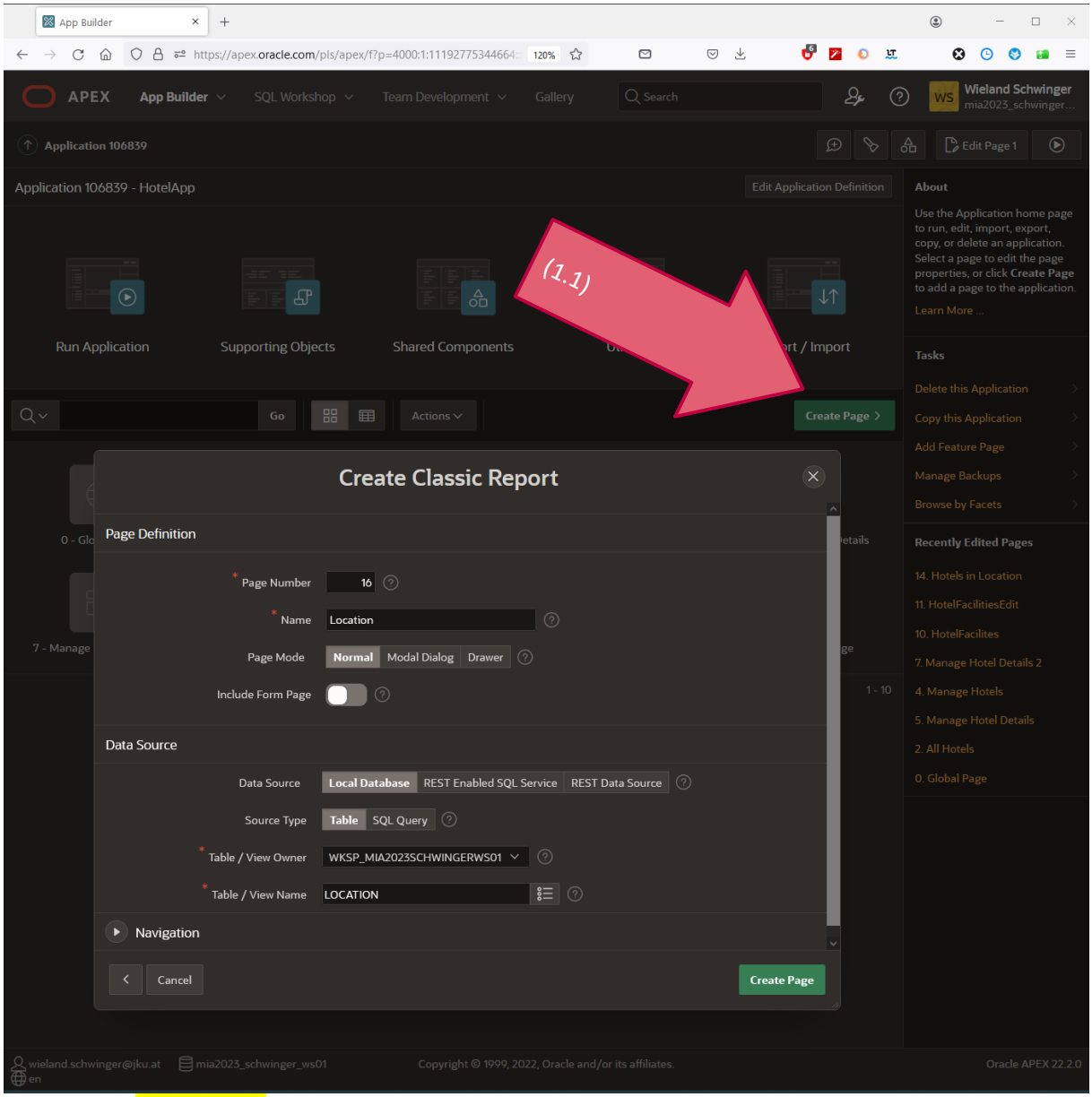
11) Passing a Parameter though a Link

Turning to the Web Layer and creating pages with APEX again, we now want to expand on the hypertext of our tutorial app. Thereby, we will learn how to create a link for an entry of a list and pass values to a subsequent page in the hypertext.

We use the **App Builder** of APEX for this. Choose your created application.

1) Create a "Location" Page

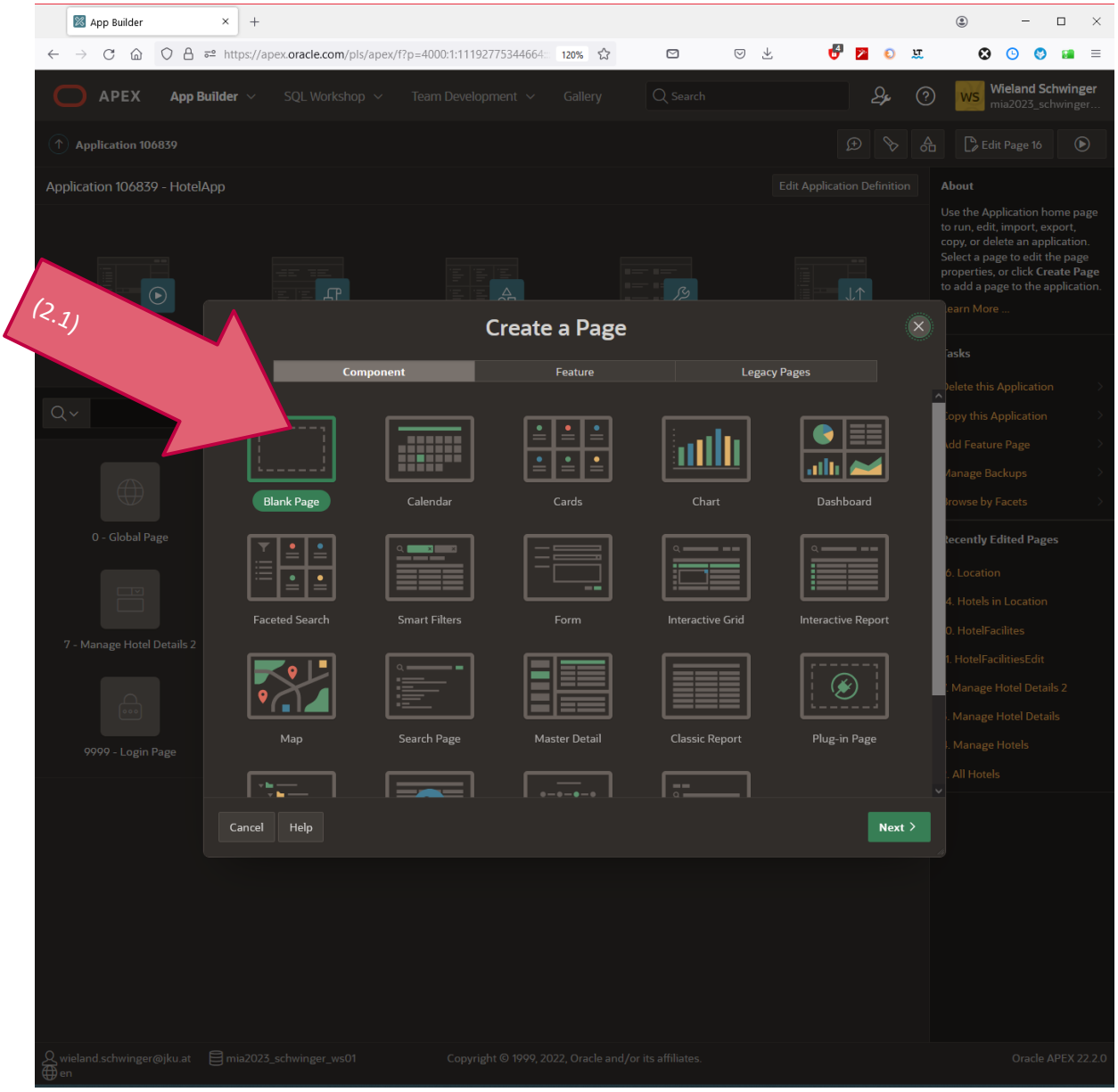
- Use the **CreatePage**-Button **(1.1)** to start creating a new page. This will act as the page from which we like to transfer some information onto the next page.
- Use the **"Classical Report"** page style again.
- Name the page **"Locations"** and use as **Data Source** the **"Location"** table.



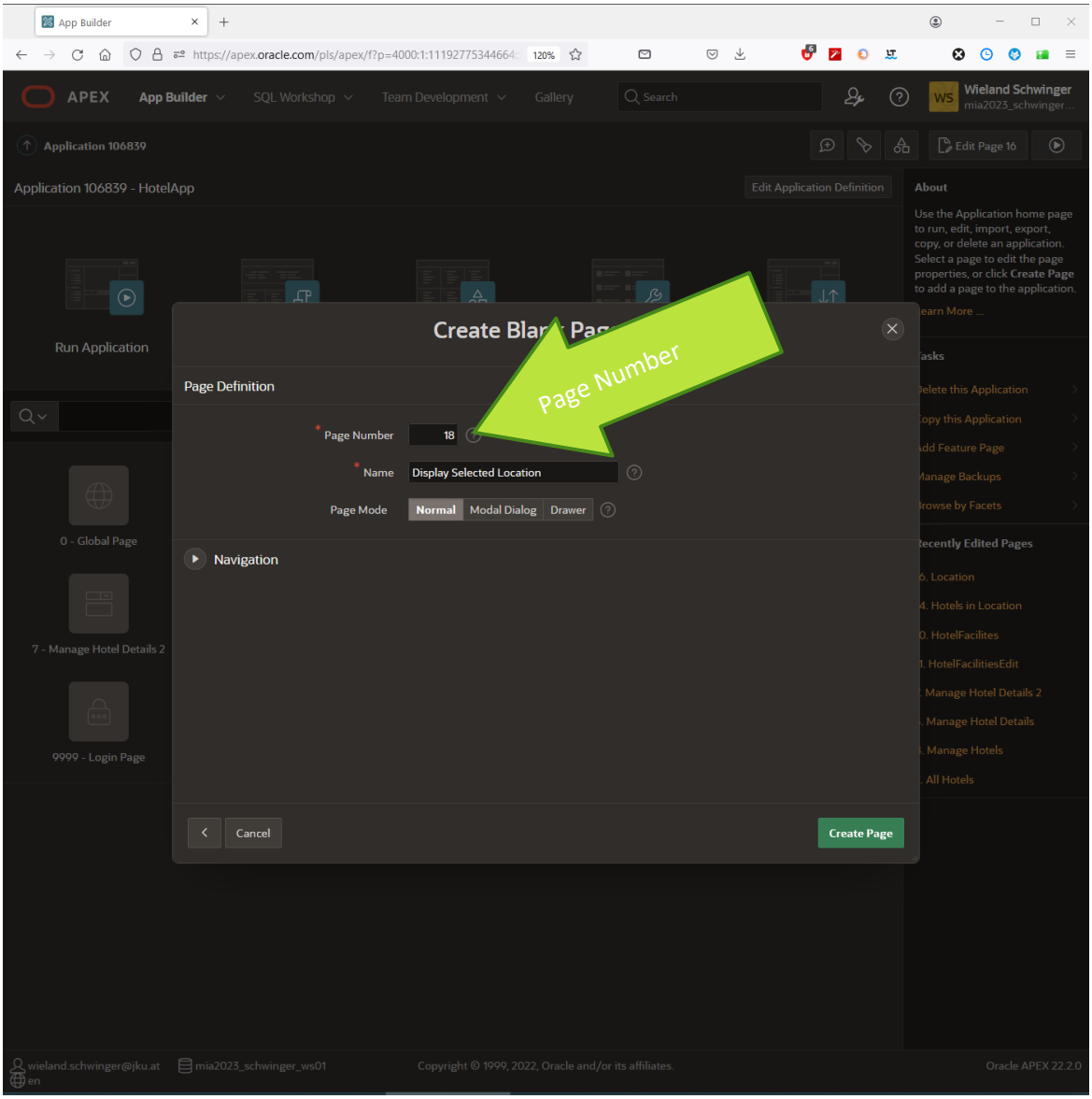
- Use **Create Page** to finalize page creation.

2) Create an empty Page

- For demonstration purposes we will create a simple empty page (**"Blank Page"**) to show the information that is passed through a link to it.
- Return back to the overview of pages of our app in the **App Builder**.
- Use the **CreatePage**-Button to start creating a new page.
- This time use a **"Blank Page"** page style again.



- Name the Page **"Display Selected Location"**.
  You may want to keep track of the **Page Number** of the newly created page at this point. Note that the page numbers are assigned automatically by APEX thus they may vary depending on your previous page creations.



- Use **Create Page** to finalize page creation.

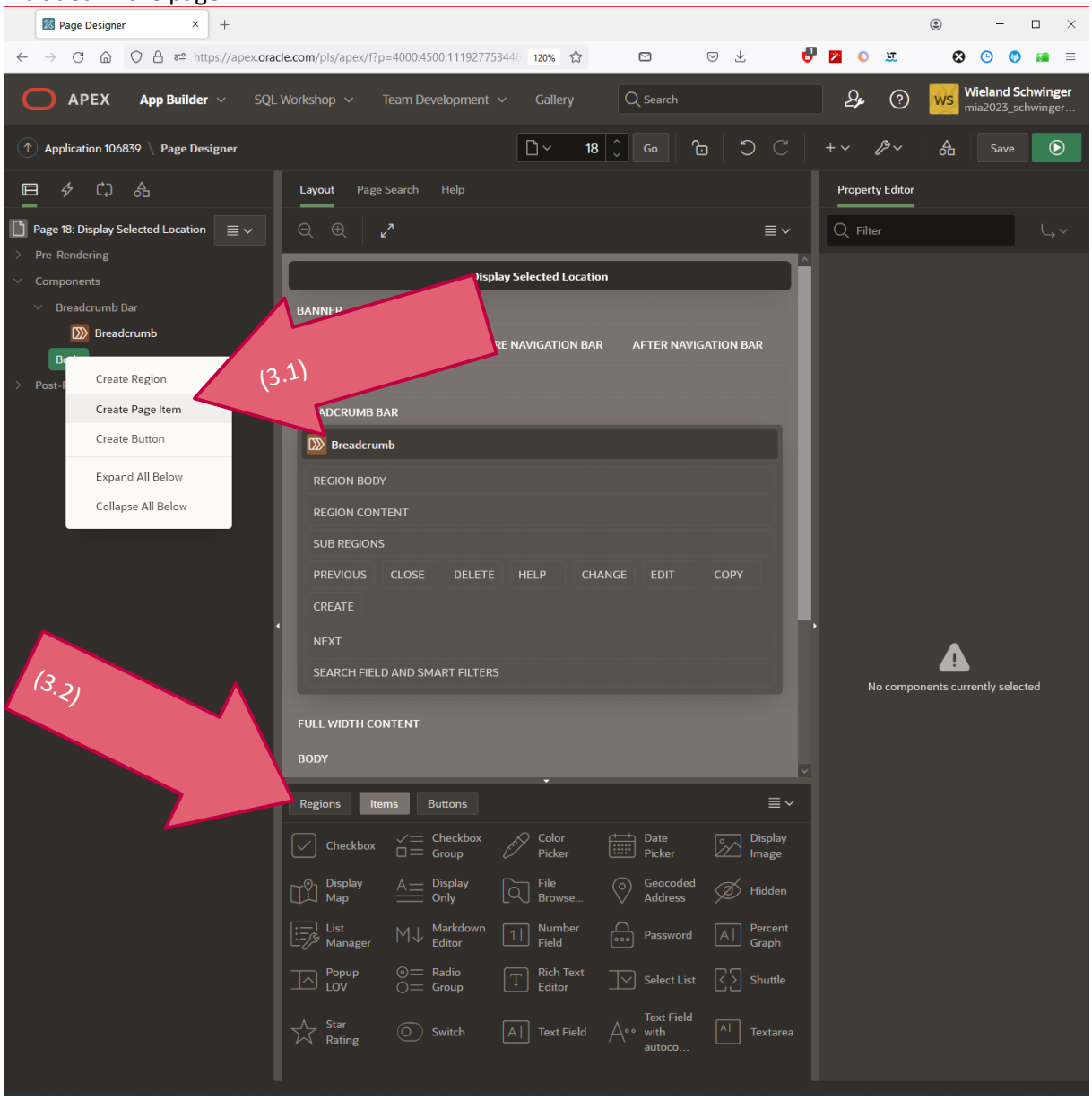
3) Add a Text Field

- In the next step, we want to manually include a **text field** (**=item**) to display some information on that page.

- In the left hand side pane select the **"Body"** (**= the main content area of the page**).
- Use a **right-mouse-buttom-click** to get to the contextual menu.
- Select **Create Page Item (3.1)**. This will create a new text field (=item of type text field).

Alternatively, you can use the **lower middle area (3.2)** of the **Page Designer** to **drag-n-drop** an **item** (or a **region** or **button**) to the desired region in the page.

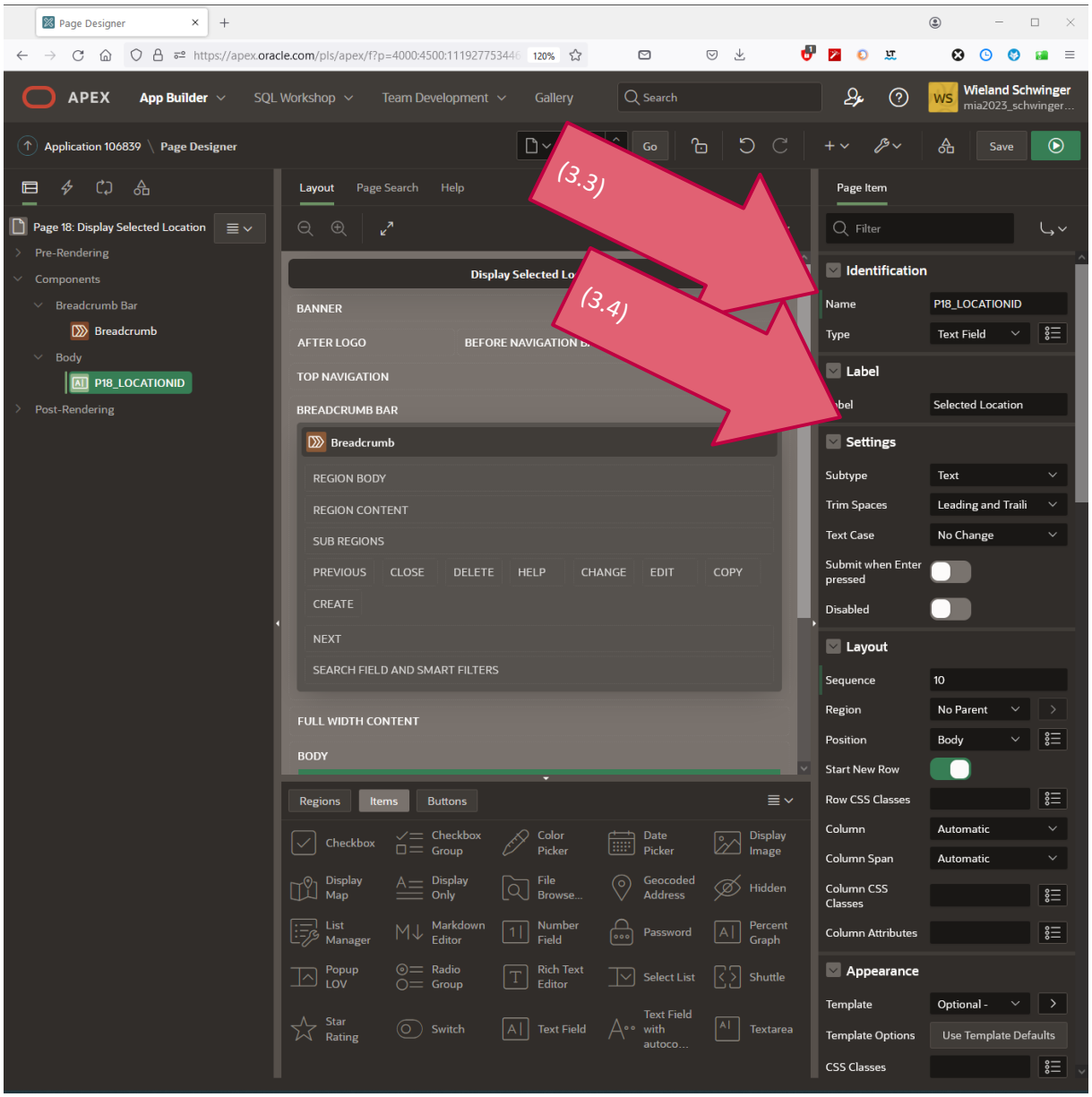
An **item** refers to a single field being text field, text area, number field, image, checkbox, etc.
A **region** is a course grained unit like a list of data, calendar, tree structure. For us in the beginning here the "list" is most important as it allows us to select multiple data rows from a table in the database.
A **button** allows to include buttons, of which different styles exist, with associated actions to be included in the page.



- Rename the newly created text field name **(3.3)**, which is for internal use, as well as the label being displayed to the user on the page. For this we will use the right side pane.

  Name: P18_LocationID
  Note that, it is common to use the page number, in our case it is page 18, thus "P_18", as prefix in front of "variable" names and other fields of a page. This makes it easier to identify their association to that particular page.
  Note that, page numbers are designed by default, although they can be manually changed, and may be different in your case depending on the order of page creation.

  Label: Selected Location
  (3.4) This is the text displayed to the user on the page in front / above of the text field.



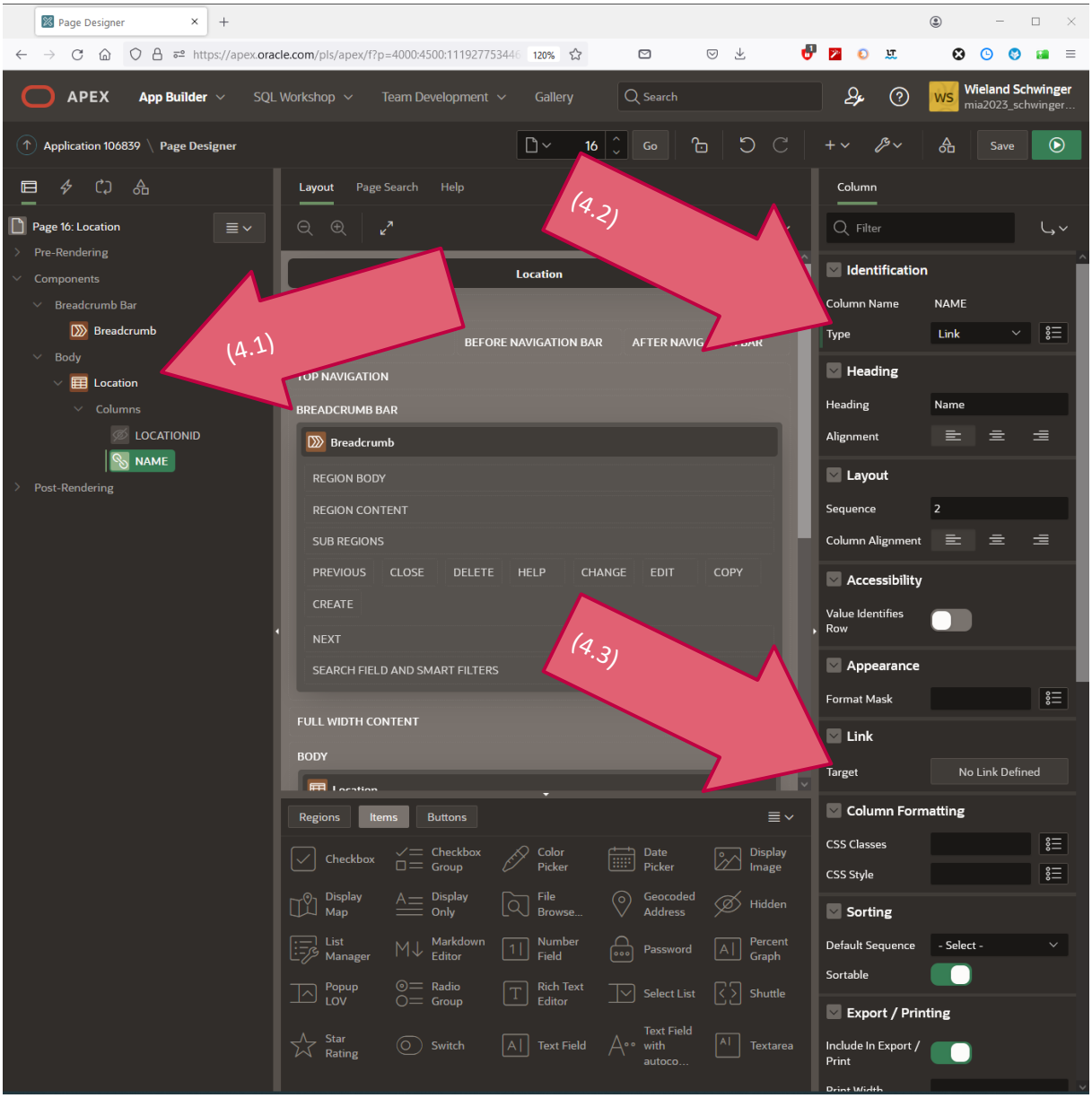
- Don't forget to **Save** the changes you made to the page.

4) Create the Link

- We now need to return back to our **"Locations"**-Page.
  Go back to the **overview on pages** of your application in the **App Builder**.
  Select the page **"Locations"** from the list of pages to invoke the **Page Designer** on that page.
  Note that you are able to also use the Developer Bar (on the bottom) of a displayed page to directly edit that page with the **Page Designer**.

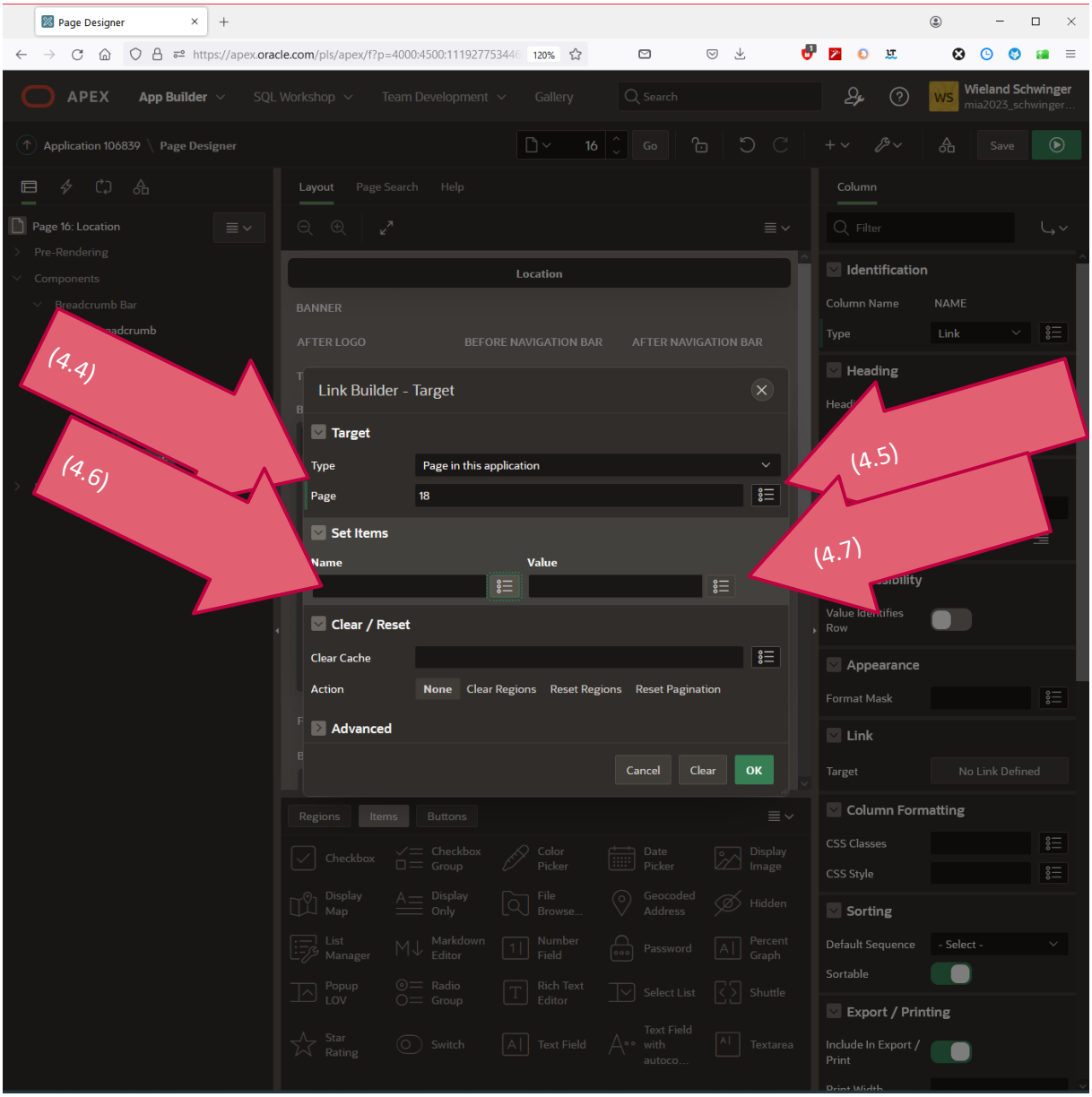
- Editing the page **"Locations"** in the **Page Designer**, we want to expand **"Location"** (=being the region of our list of locations) and then the **"Columns"** to see all the fields.
  You can see that we do have **"LocationID"** (that is a hidden field) and **"Name"** being a text field.

- We now want to change the type of the **"Name"** field to be a Link instead of a simple plain text field.
  For this, in the **right hand pane (4.2)**, we change the Type to **"Link"**. This basically converts the text displayed to an **anchor** of a link.

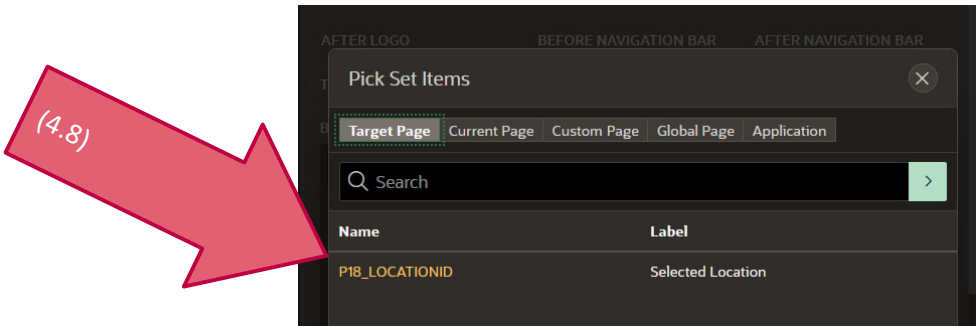


- Having the anchor defined, we need to specify the link, or better to say, the target of the link. Staying in the **right hand pane**, further down we have the **"Link" section**.
  Currently **(4.3)**, there is **No Link Defined**. Click on **No Link Defined** to specify a target.

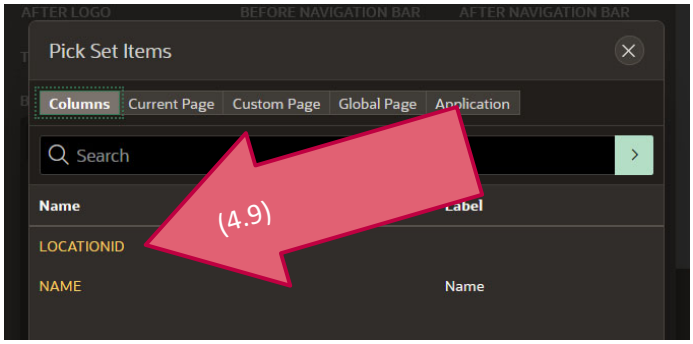
- Specify **(4.4)** our **"Display Selected Location"** under **Page**. You can use the **"chooser"** **(4.5)** **(4.7)** or directly type in the number of the page (in our case it was numbered as page 18). This is the target page to be called if the link is clicked.

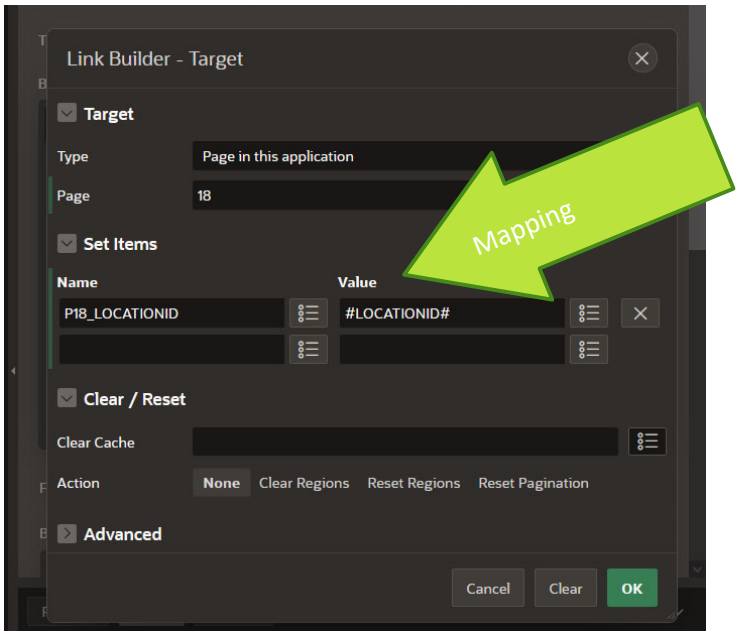


- We now want to "pass" on the **LocationID** to the target page.
  Under **Set Items** **(4.6)** we select for **Name** the receiving item in the target page, that is **P18_LocationID (4.8)**.



- Under **Value** **(4.7)** we specify which information/value is going to be passed to **P18_LocationID**, i.e. the receiving item **(4.9)**.

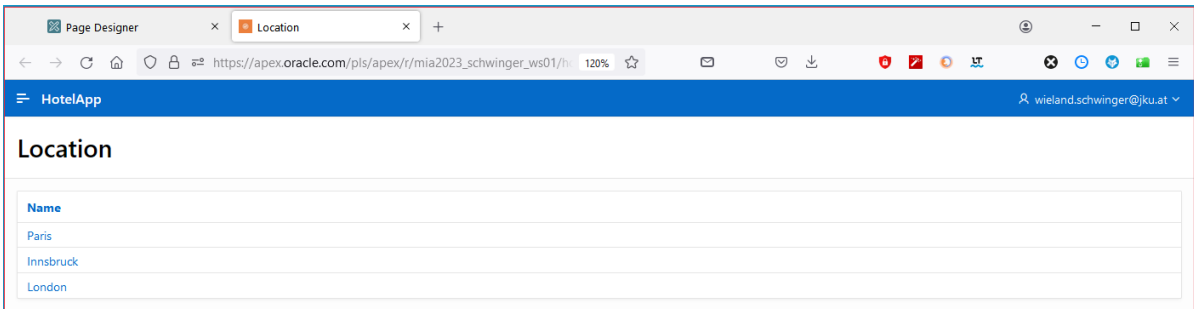




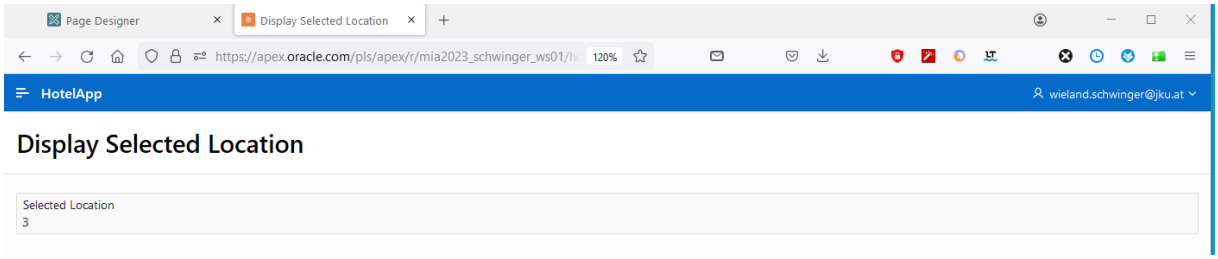
- Close with **OK**.
- **Save** the page.

We have now defined a **"contextual link"** between our page **"Locations"** to the **"Display Selected Location Page"** passing the information of the selected location as contextual information to the item "**P18_LocationID**" in the target page.

- Use the ▶-Symbol to run the page and see the result.



- **Click** on one of the displayed locations. Now those being rendered as a link, the Display Selected Location is displayed with the LocationID of that selected location.
- **Go back** using the back button and **try another location**. The LocationID displayed should now be a different one.

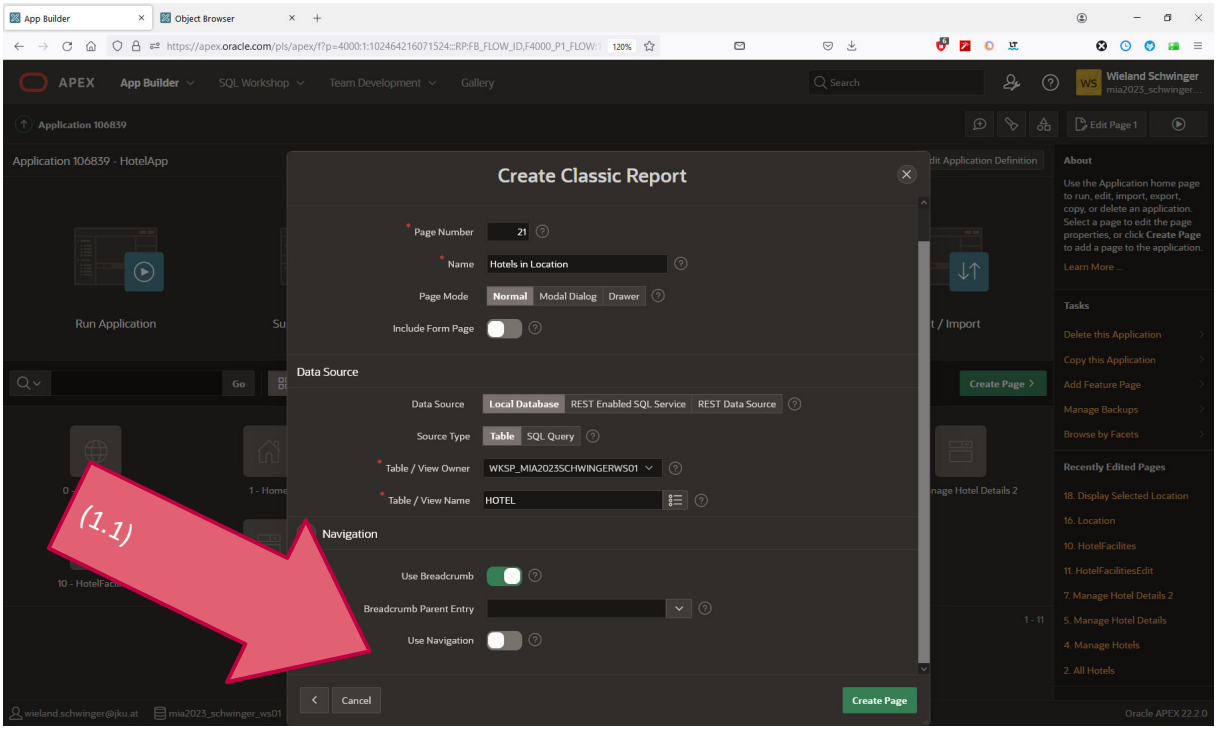


12) List of Hotels in a certain Location

Basing on what we have learned for the page **"Display Selected Location"** we will now learn how to use the passed information to select data on the target page.

1) Creating a new target page

- We create a new page. **App Builder → Create Page**
- The page shall again be a **"Classical Report"**
- Name the page **"Hotels in Location"**.
- Other than before, this time we would like to specify Navigation options. Since the page will not make sense as a stand alone page – requiring to be "navigated" to from our **"Locations"** page, we this time will **not Use Navigation (1.1)**.

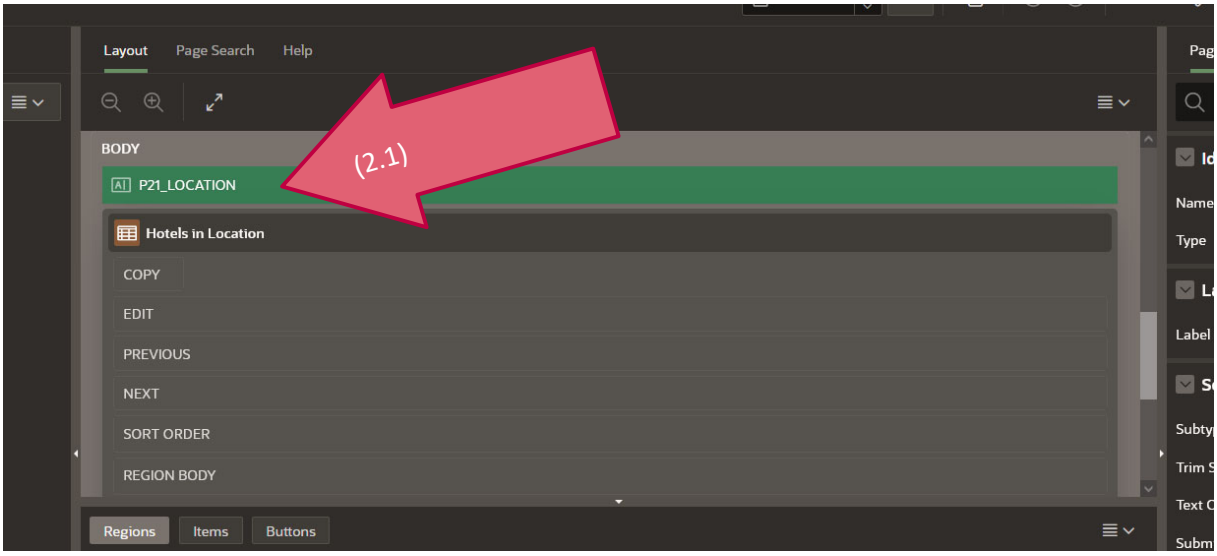


- Use the ▶-Symbol to run the page and see the result.
  Note that now, when you have a look at the navigation bar of our Web application, no entry for the page **"Hotels in Location"** has been generated.

2) Creating an Item for LocationID

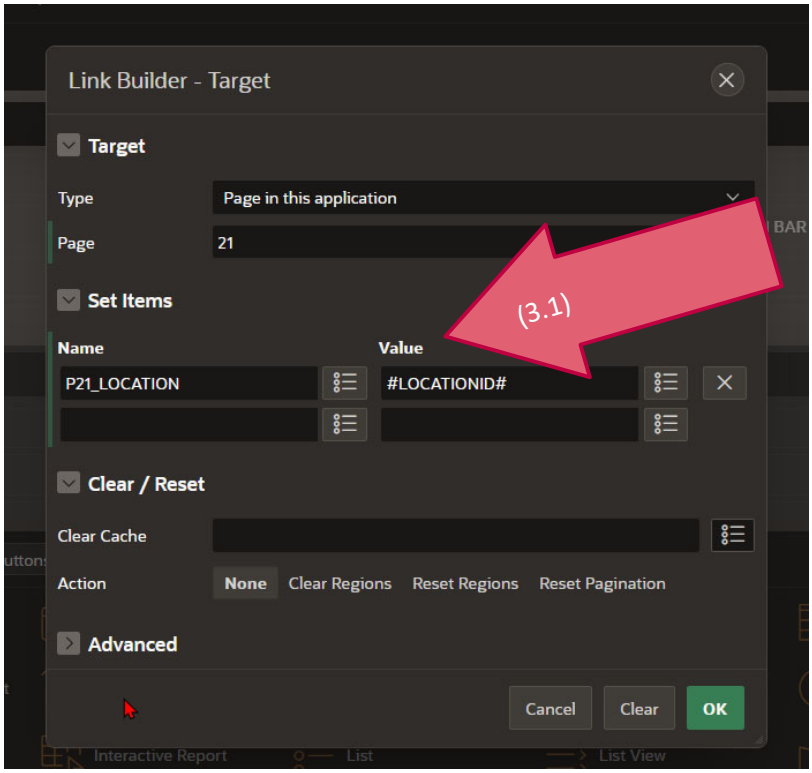
- Like before, add a **text item** for the **LocationID** on the **"Hotels in Location"**.
  Name: ***<PageNumber>*_Location**
  Label: "**Location**"

- Place the text item before the region of the hotels. Drag the text item in the middle of the Page Designer to the according position.



3) Change the Linkage for page "Locations"

- We now want to change the link we created for page **"Location"** to refer to **"Hotels in Location"** with passing the **locationID** to the text item of page **"Hotels in Location" (3.1)**.



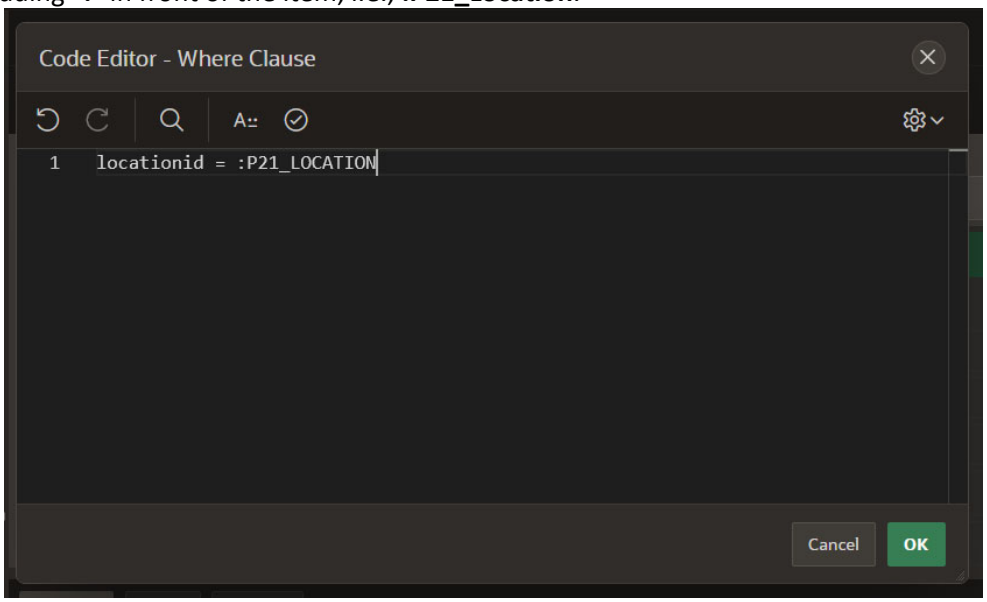
- Access the page **"Locations"** in your web browser, select a location and see the resulting page **"Hotels in Location"**.

3) Use the information passed to select data from the database

The page **"Hotels in Location"** is not yet exploiting the location information passed to it.

- For the **"region"** **"Hotels in Location"** we want to change the **Source** and there the **Where Clause.**

- Specify the where clause such that the **locationID** of Hotel needs to be equal to the value of the item that has been passed by the link.
  Access to the item – in our case **P21_Location** – can be achieved in the **WHERE**-clause by adding "**:**" in front of the item, i.e., **:P21_Location**.



- You can use the symbol to check whether your **WHERE**-clause is syntactically correct.

- Use **OK** to close the dialogue.
  Now the resulting list of hotels is constrained by the **WHERE**-clause only resulting in those hotels which have that particular **locationID**.

4) See the resulting page

Now we want to see the resulting **"Hotels in Location"** page.

- Go to the overview on the pages of your Web app in the **App Builder.**

- Start the application with **Run Application**

- Use the **navigation menu** (left hand side, you might to open it first if it is minimized) to go to **Locations**.

- Select a location which navigates to the **"Hotels in Location"**

- The resulting list of hotels should now only display the hotels in the selected location.

5) Replace the locationID with the name of the location

Currently, "Hotels in Location" displays the **locationID**. We want to change it now to display the name of the **location** rather then its **locationID**.

- Use the **Page Designer** to change the page **"Hotels in Location"**.
- Add a **text item** **"P21_LocationName"** to the page and place it near the **locationID**.
- Set the **Label** to empty.
- Change the **Source** such that its **Type** is **"SQL Query"**.
- Specify the **SQL-query** as follows.

```
select name from location where locationid = :P21_LOCATION
```

- Set the **Type** of **item** **"P21_Location"** (which contains the passed value) to **"hidden"**.
- View the now resulting page by using the navigation of your Web app starting from **"Locations"** and following one of the links.

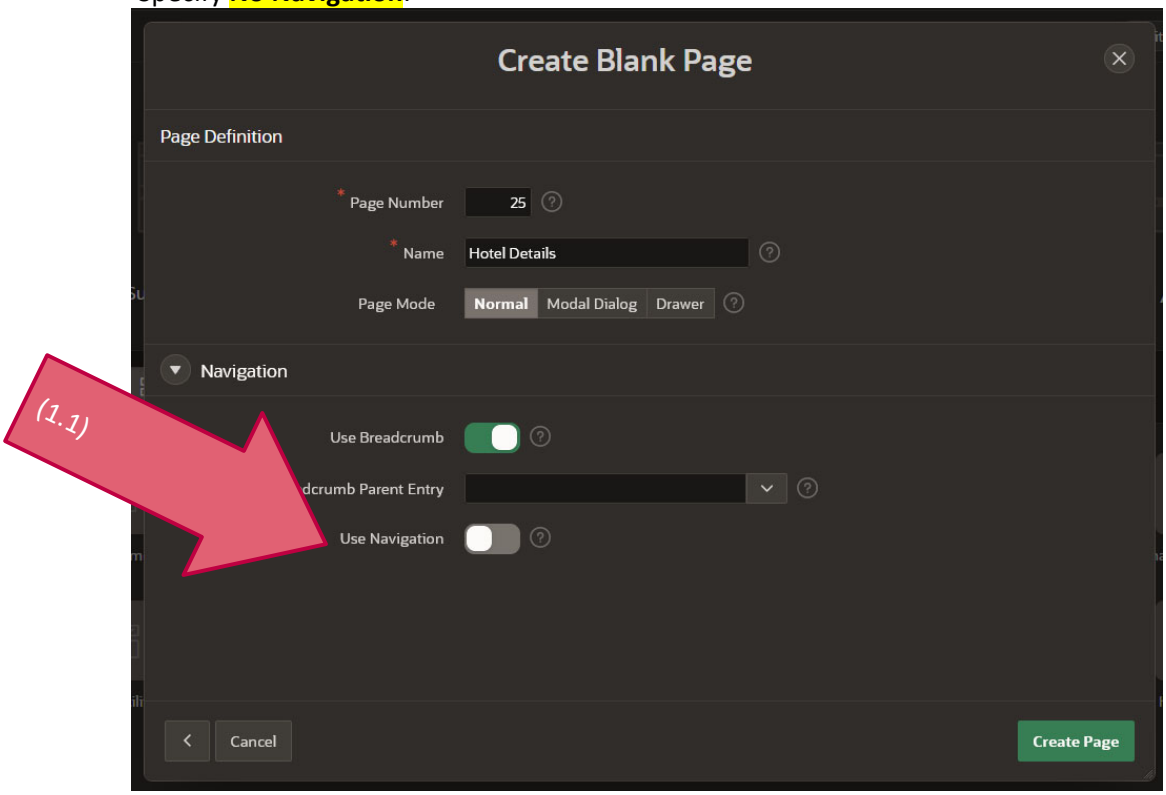
We have now learned how to constrain the selection of data in a region by a **WHERE-clause**. Additionally, we have seen how to retrieve the value for an item through a **SQL-query**.

13) Hotel Details Page

Basing on what we have learned for the page **"Display Selected Location"** we will now learn how to use the passed information to select data on the target page.

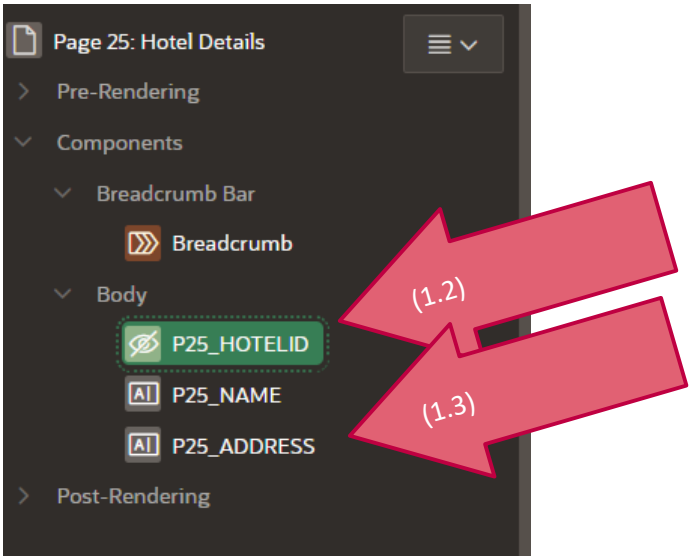
1) Creating a page "Hotel Details"

- Create a new page as a **"Blank Page"**.
  Name it **"Hotel Details"**.
  Specify **No Navigation**.



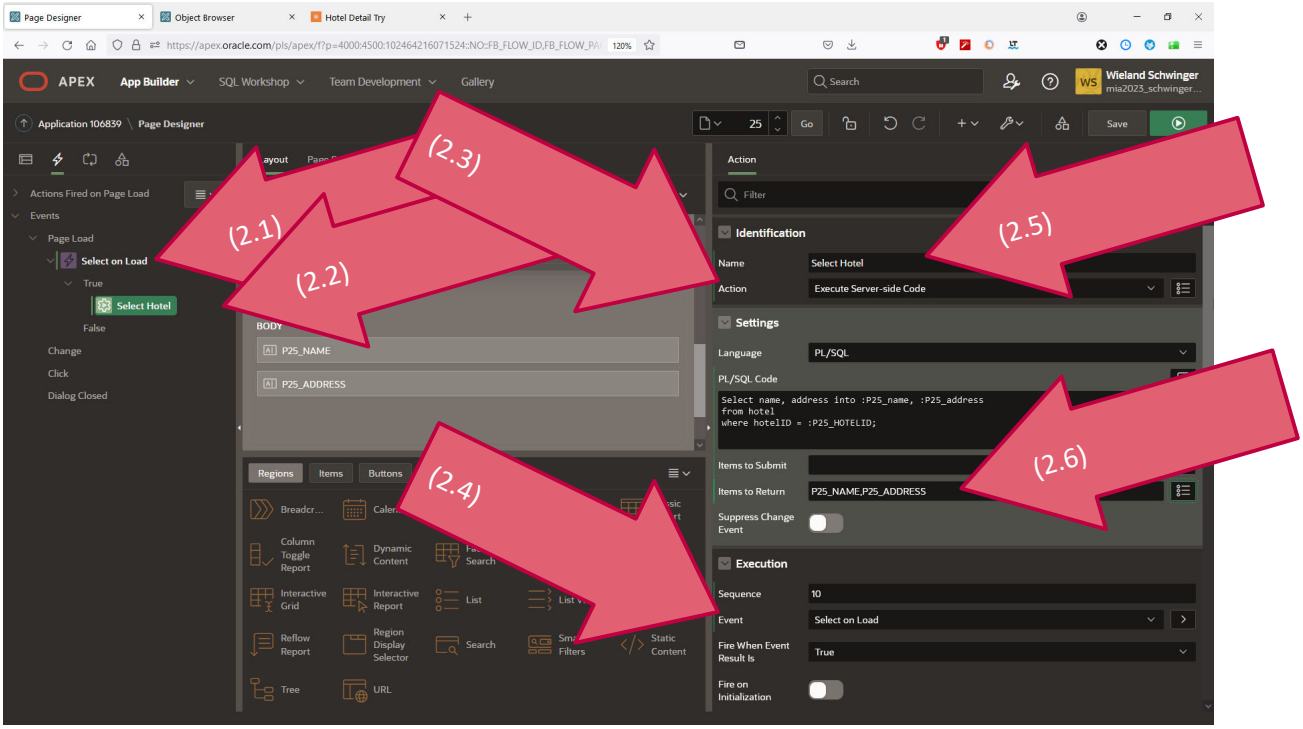
- Create **text items** for the **hotelID**, which again will serve to hold the parameter, set its **type** as **"hidden" (1.2)**.

- Create **text items** to hold **name** and **address (1.3)**.



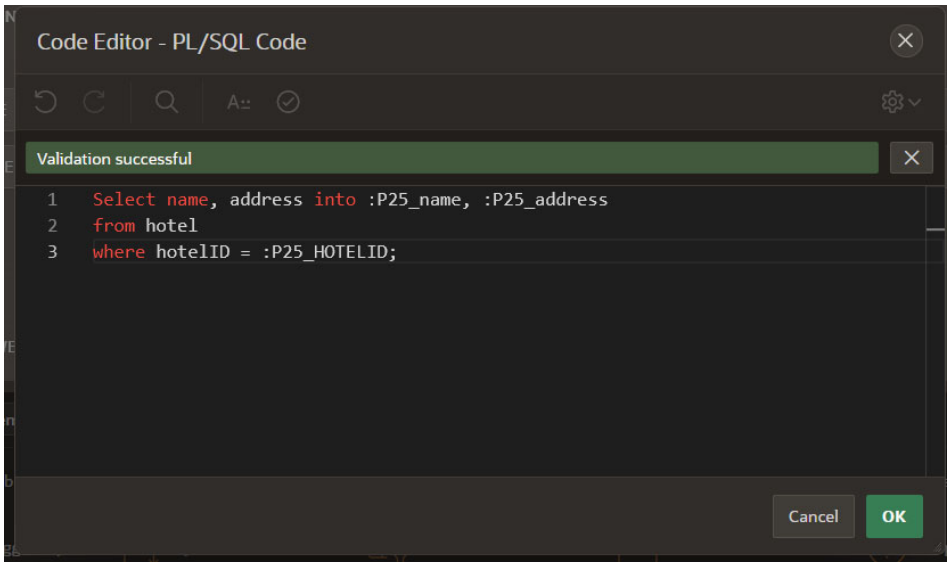
2) Creating server-side code to fill the items

- Create a **new action (2.1)** on **Page Load (2.2)** as **Execute Server-side Code (2.3).** Optionally, name the action **"Select on Load" (2.4)**.
- Optionally, name the true part of the action **"Select Hotel" (2.5)**.



- Set the PL/SQL code as follows:

```
Select name, address into :P25_name, :P25_address
from hotel
where hotelID = :P25_HOTELID;
```



Note that, in the select part contains an **"into"**-part which specifies the variables/items.

- Specify under **Items to Return (2.6)** to return **:P25_name**, **:P25_address**, to have the selected information to be used in the items.
  The names of the return items need to match those used in the **INTO**-clause of the SQL.
  The naming however is up to developer – the naming convention used is to prefix the variables with the "P" (for "page") and the number of the page (in our case "25", which may vary for your Web app).

3) Add navigation from page "Hotels in Location"

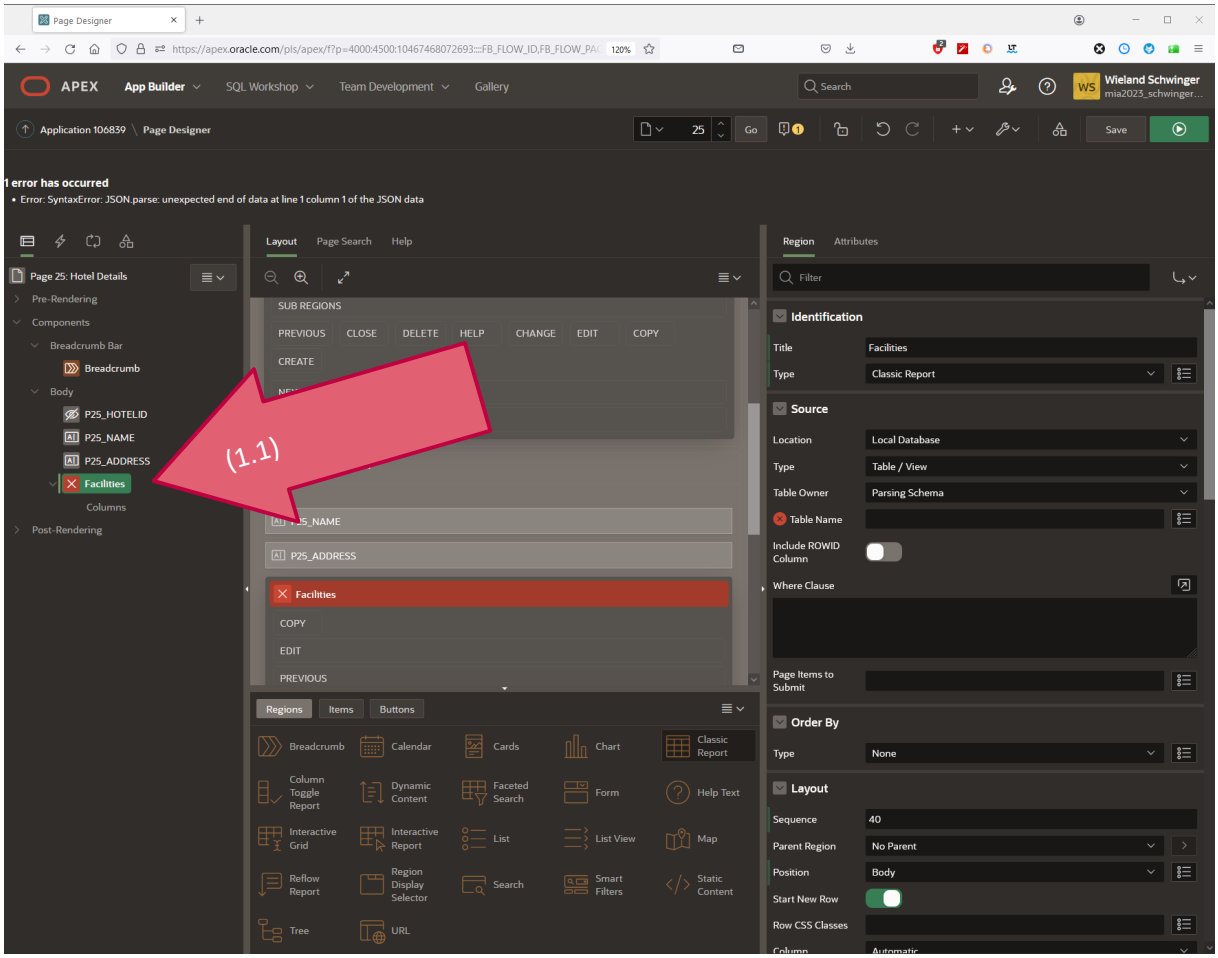
- Change the **type** of the item **"Name"** to **"link"**.
- Specify as **target** the page **"Hotel Details"**.
- Specify the parameter mapping such that the **"HotelID"** from the **region** **"Hotels"** of the **"Hotels in Location"** page to pass the **HotelID** to the receiving item **HotelID** of the page **"HotelDetail"**.
- Have a look at the result. For this use the **navigation** starting on **"Locations"** over selecting a particular location to the list of hotels in that location and by selecting a particular hotel to the details of hotels.

14) Hotel Facilities

In the next step we will learn to exploit the related information. We will demonstrate that by adding facility type information specific to a hotel.

1) Creating region "Facilites"

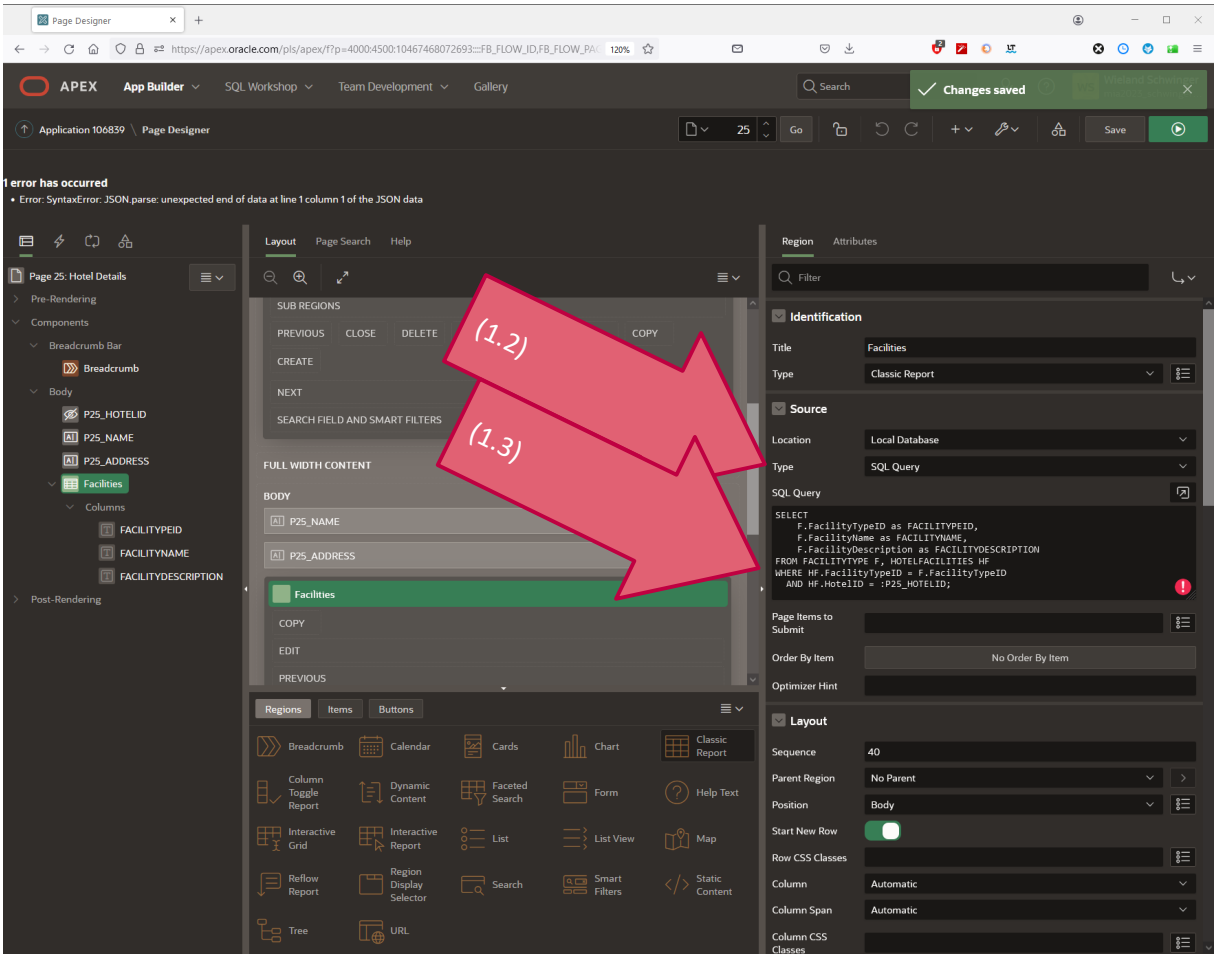
- Edit the page "**HotelDetails**" in the **PageDesigner**.
- Add a new **region** of type **"ClassicReport"** and position it below the hotel details **(1.1)**. You can achieve this be using **drag-n-drop** from the left bottom of the page designer.
- Name it **"Facilities"**.



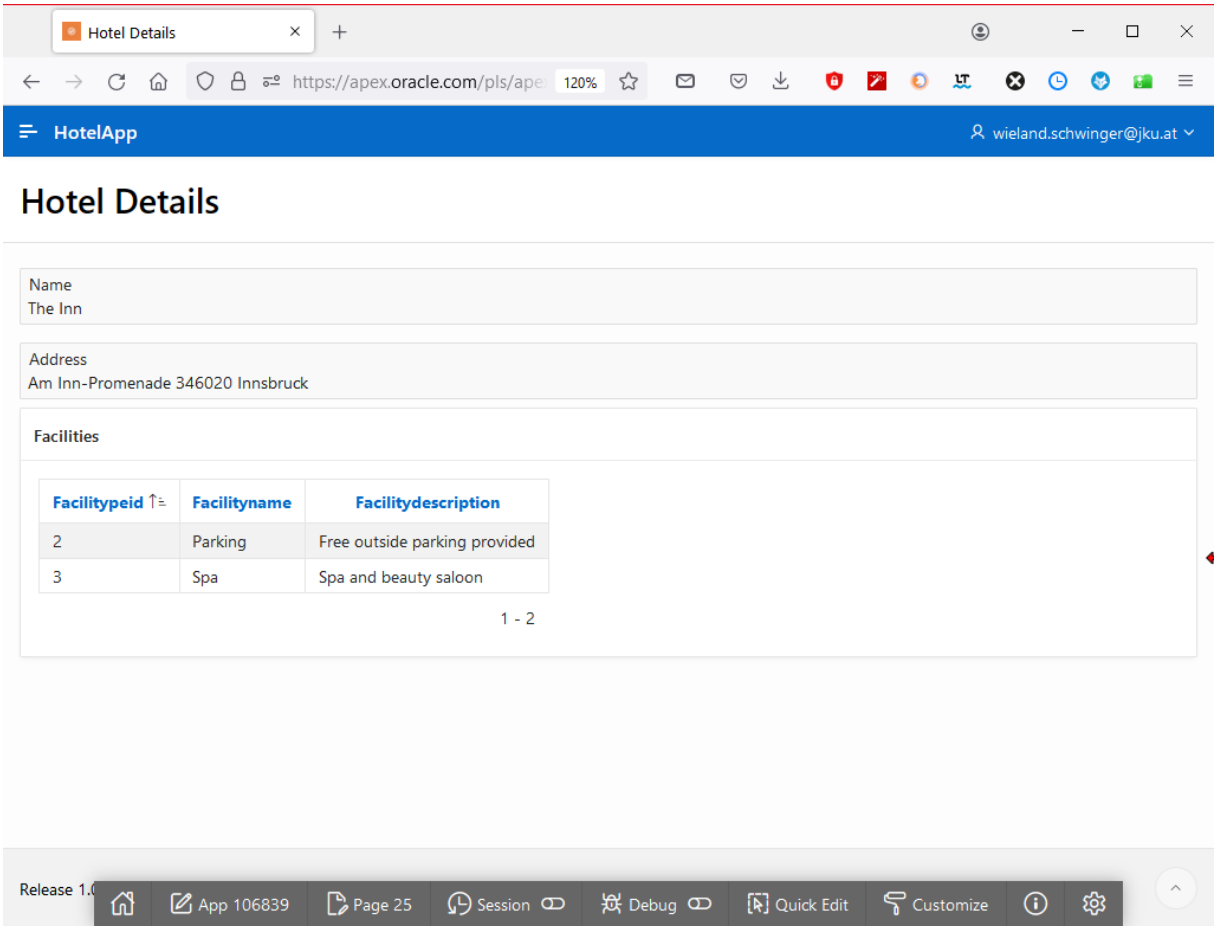
- Define the **Source** to be based on **SQL Query** **(1.2)**. Set the PL/SQL code **(1.3)** as follows:

```
SELECT
    F.FacilityTypeID as FACILITYPEID,
    F.FacilityName as FACILITYNAME,
    F.FacilityDescription as FACILITYDESCRIPTION
FROM FACILITYTYPE F, HOTELFACILITIES HF
WHERE HF.FacilityTypeID = F.FacilityTypeID
  AND HF.HotelID = :P25_HOTELID;
```

The selected columns from the joined tables **FacilityType & HotelFacilites** are alias (with **"AS"-clause** to the columns that are used in the region). The result set is also constrained to contain only those entries which belong to the specific hotel being selected for the page identified by **:P25_HotelID**.



- **See** the **resulting page**. Make sure to select a particular hotel, e.g., by navigating from the list of hotel in a location and selecting a particular hotel to navigate to the detail page.
  You may want to insert additional facility types in the **FacilityType**-table and associate some facility types to various hotels via the **HotelFacilities**-table. Use the **ObjectBrowser** for this.



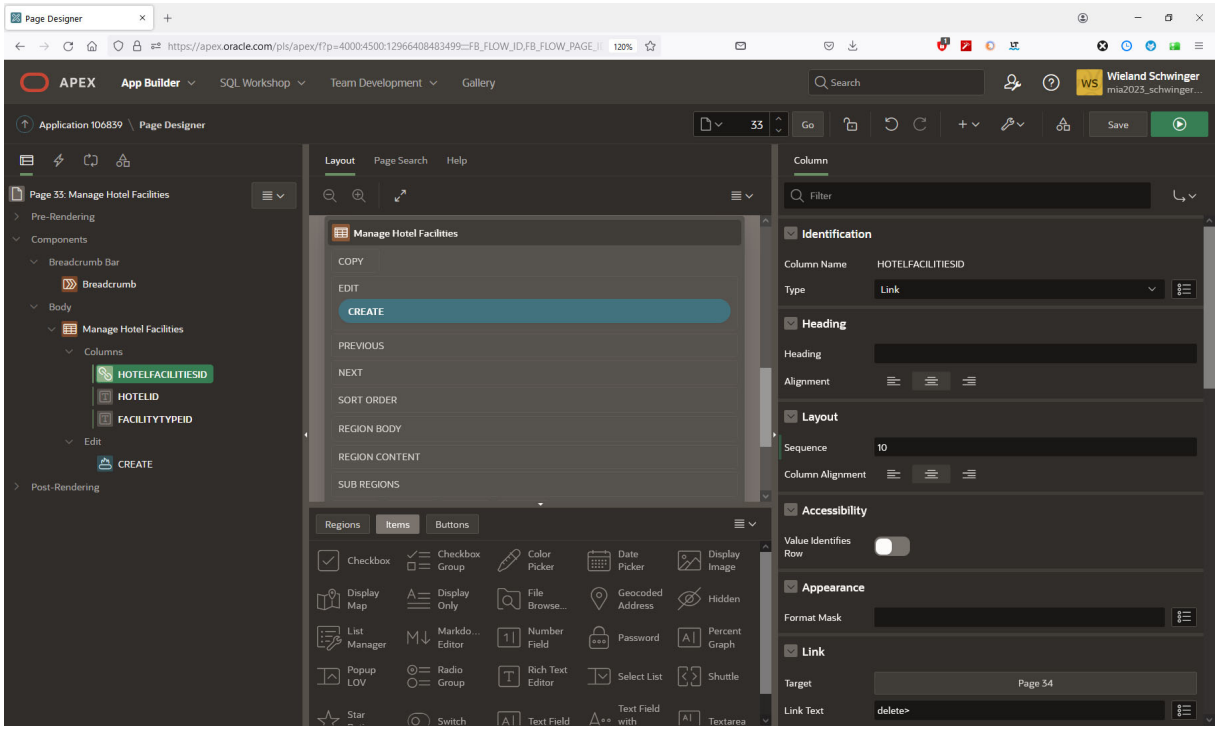
15) Manage Hotel Facilities

In the next step we will learn to delete related information. We will demonstrate that by deleting facility type information specific to a hotel.

1) Creating page "Manage Hotel Facilities"

- Create a page **"Manage Hotel Facilities"** by using the **AppDesigner**.
- Use as **page type Classic Report**.
- Name the page **"Manage Hotel Facilities".**
- **Include Form Page** of **type "Normal"**
- **Name** the **form page "Manage Hotel Facilites – Delete"**
- Use as **Data Source the table "HotelFacilities"**

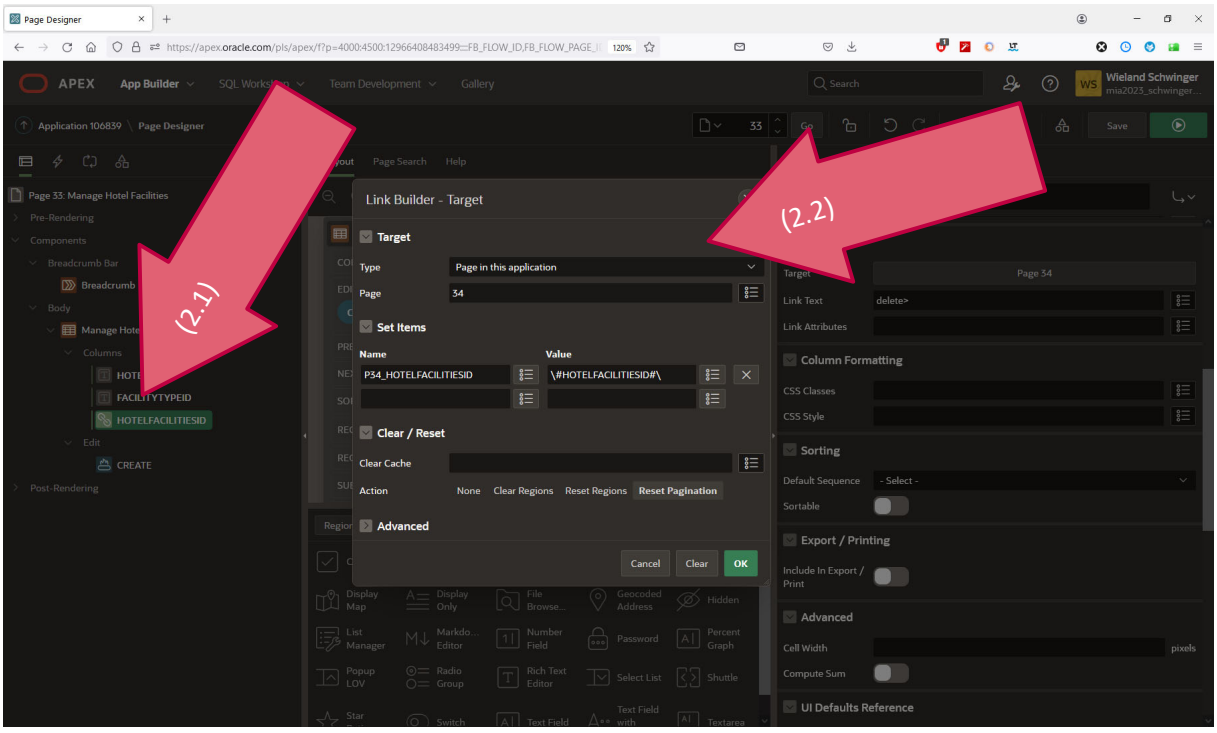
This will **create two pages**.
The **first page** will serve for selecting a particular facility of the hotel which then can be removed.
The **second page** will actually be used to delete the association of the facility type to the hotel.



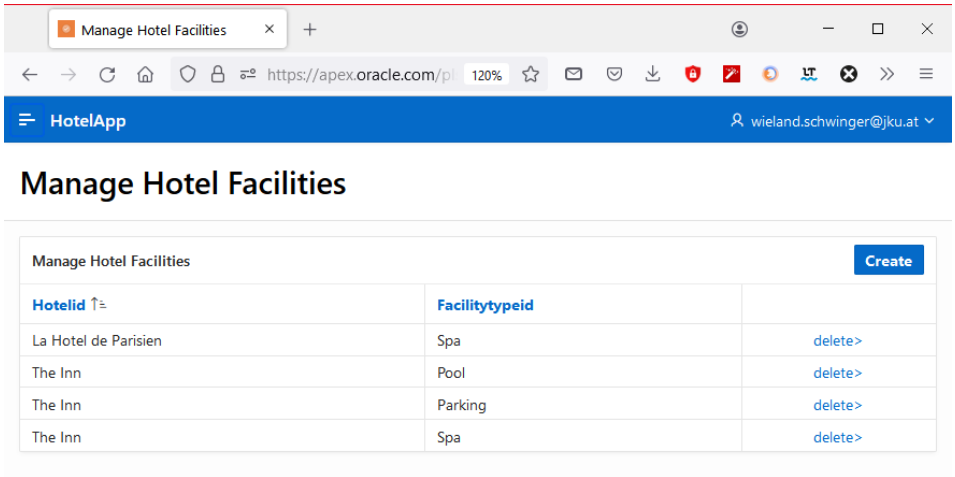
2) Change link to "delete"

Since we specified the page to have a **form page**, each row in our classical report already has a link for each row to initiate some standard **"edit"**.
We will change this now and thereby understand how to associate individual actions to row entries in a classical report.

- Drag the item **"HotelFacilityID"** to the end of the list of items **(2.1)** of the classic report region. This will render finally the **"anchor"** at the end of each row.
- To change the **"anchor"** previously generated referring to the also created **"Manage Hotel Facilities – Delete"** we first select the item **"HotelFacilityID"** and than inspect the already defined **link** by clicking on **Link** and there on the **target (2.2)**.
  You can see that the page **"Manage Hotel Facilites – Delete"** is the **target** of this **link** and the **HotelFacilityID** identifying the associated **facility type** of the hotel is passed on.
  You can now **close the dialogue window**.
- Under **target** you find **Link Text**. There the text / visualization of the anchor is specified.
  The standard "text" that is set here is an icon representing editing possibilities.
  We will change that and **replace** all with just the text **"delete>"**.



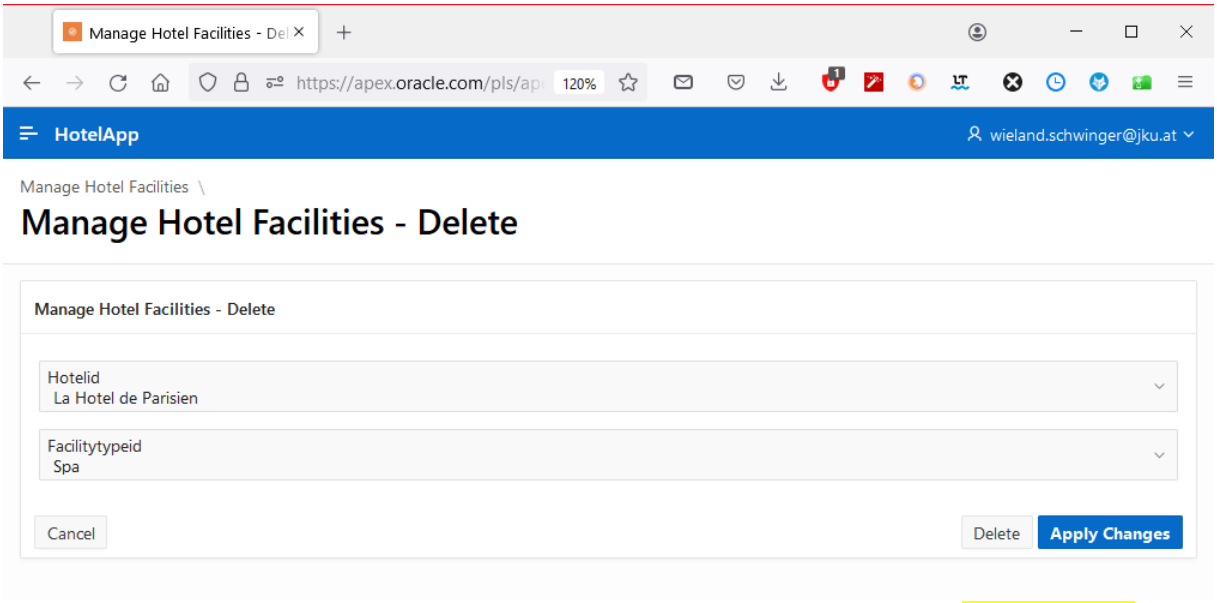
- You can now **see** in the **resulting page** that the anchor is rendered differently.



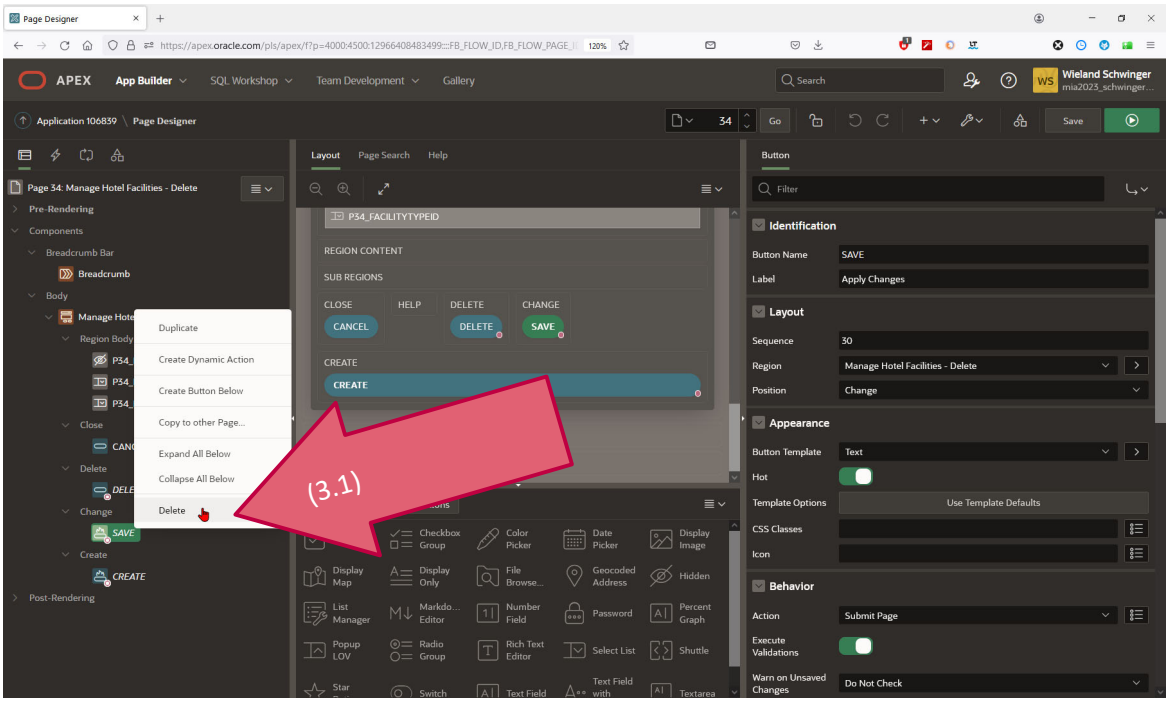
3) Adust the "Functionality" of the "Manage Hotel Facilites – Delete" page

The automatically generated form page **"Manage Hotel Facilites – Delete"** contains a set of regular update as well as delete "Functionality". We want to adjust that in the following.

- Follow the **"delete>"** link to get to the rendered **"Manage Hotel Facilites – Delete"** page. You see that the page also contains an **"Apply Changes"-button** we would prefer not to have as it is associated with **"update"**-functionality in the default generated form.

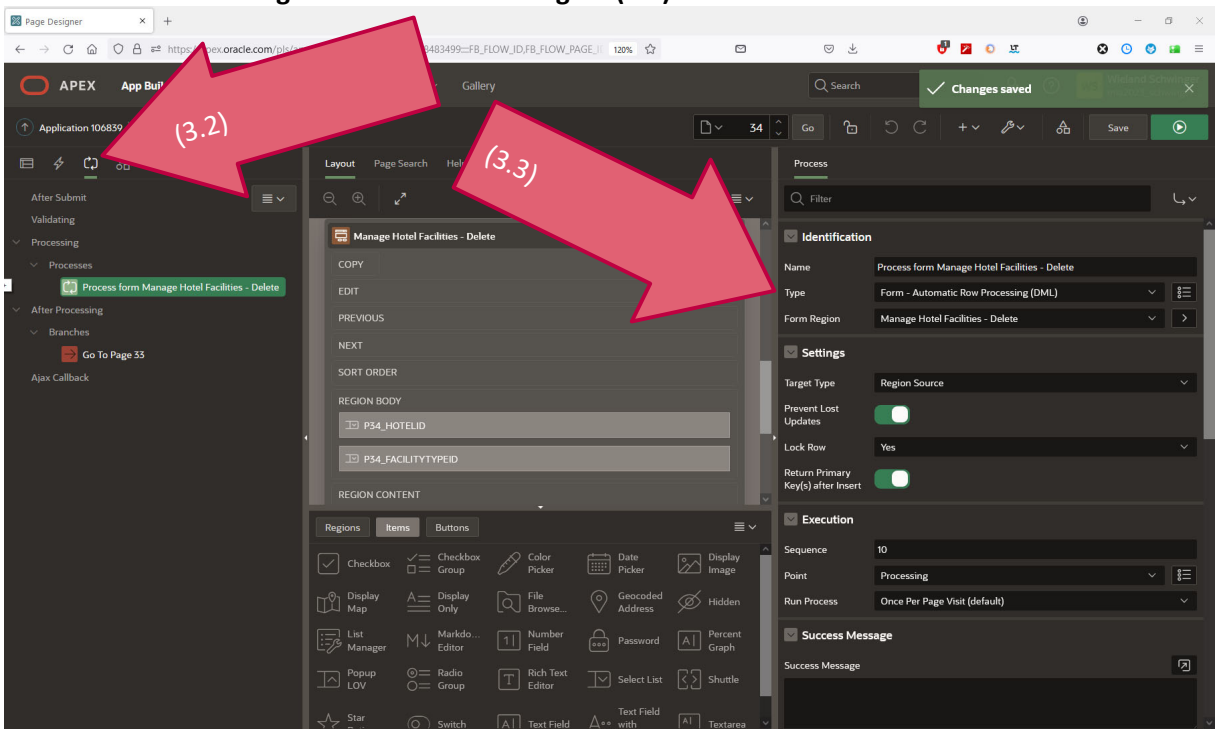


- To achieve this open the **"Manage Hotel Facilities – Delete"** page in the **Page Designer**. In our case it is the page with number 32.
- In the **right hand pane** you see a series of **buttons**.
  We **delete** the **"Create"** and the **"Save"**-button by selecting the button and the contextual menu.



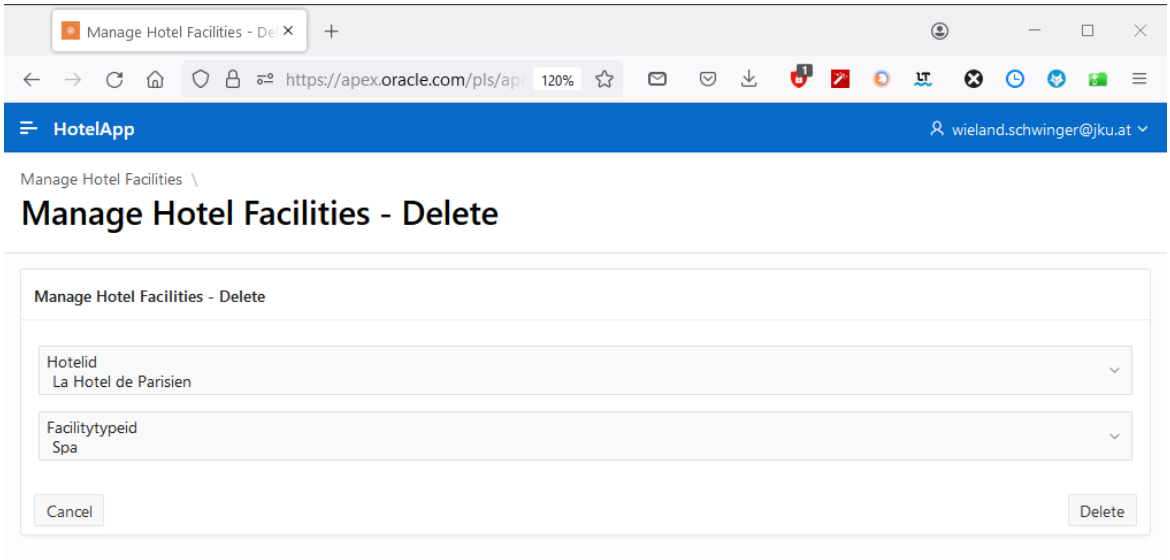
- **Save** the page.

- If you **investigate** the **processing (3.2)** of the page you can see that the **"Form – Automatic Row Processing"** is set for the **form region (3.3)**.

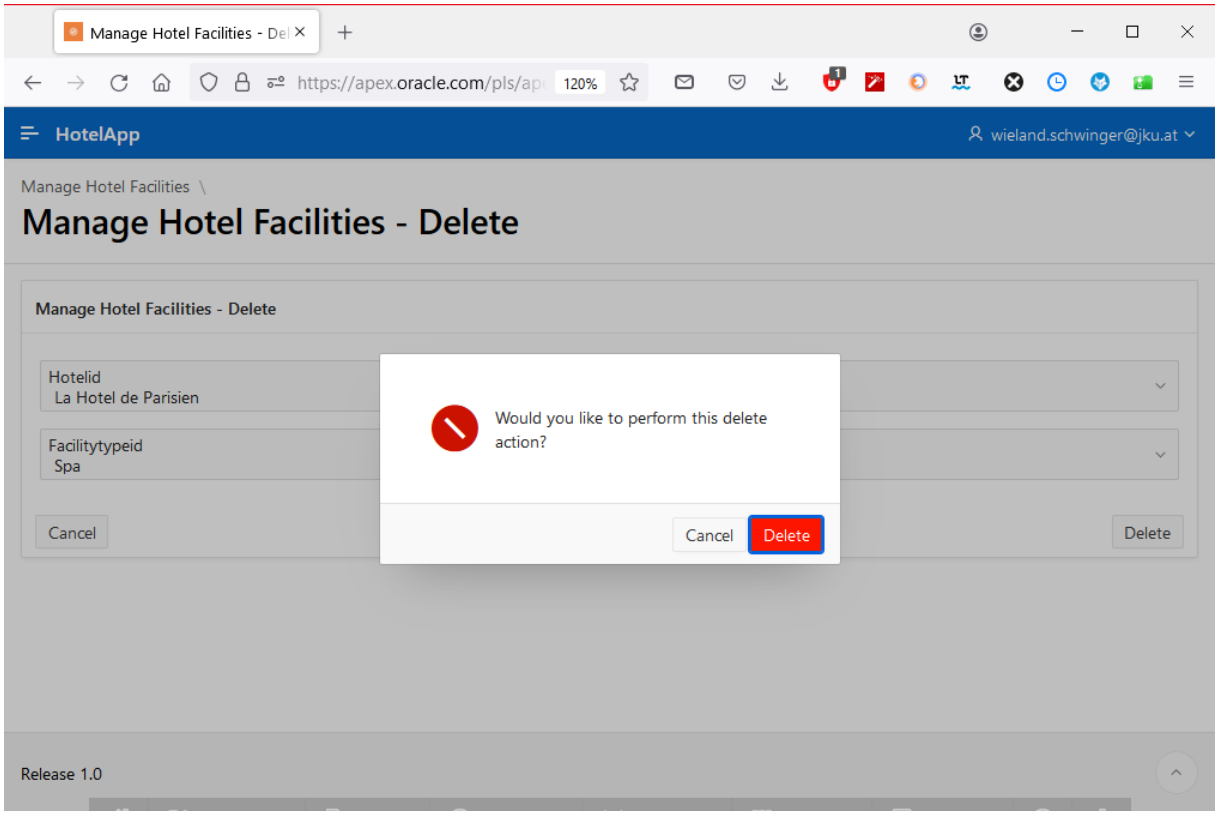


- If you turn back to the **button "DELETE"** and **investigate** the **behaviour** you can see that **behaviour** for **submit** for this button is set to **"SQL DELETE action"**.
  Additionally, you can see that a **confirmation** before the actual deletion is set **active**.

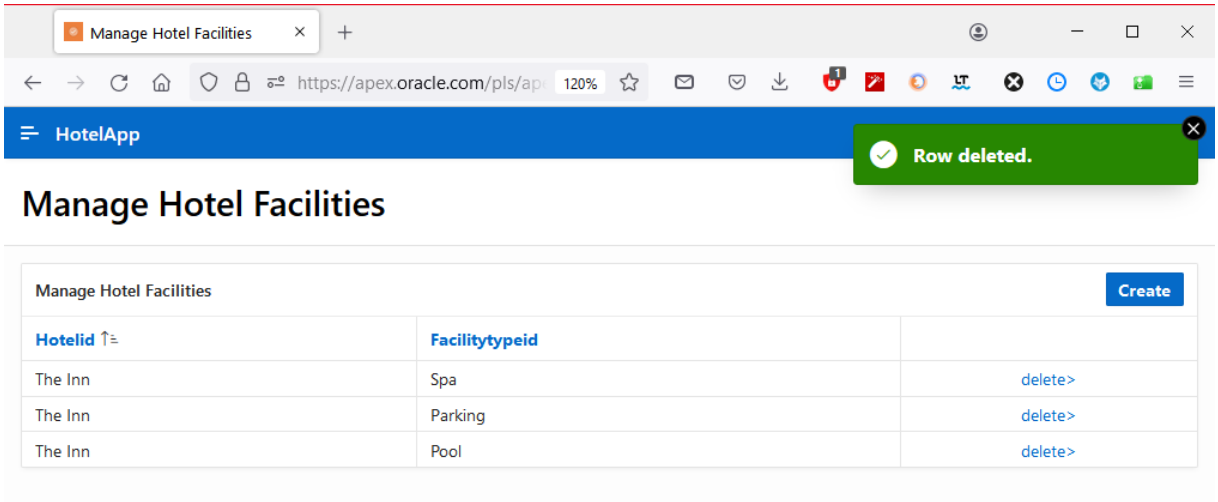
- If you now re-render the **"Manage Hotel Facilities – Delete"** page, only **"Cancel"** and **"Delete"** are available.



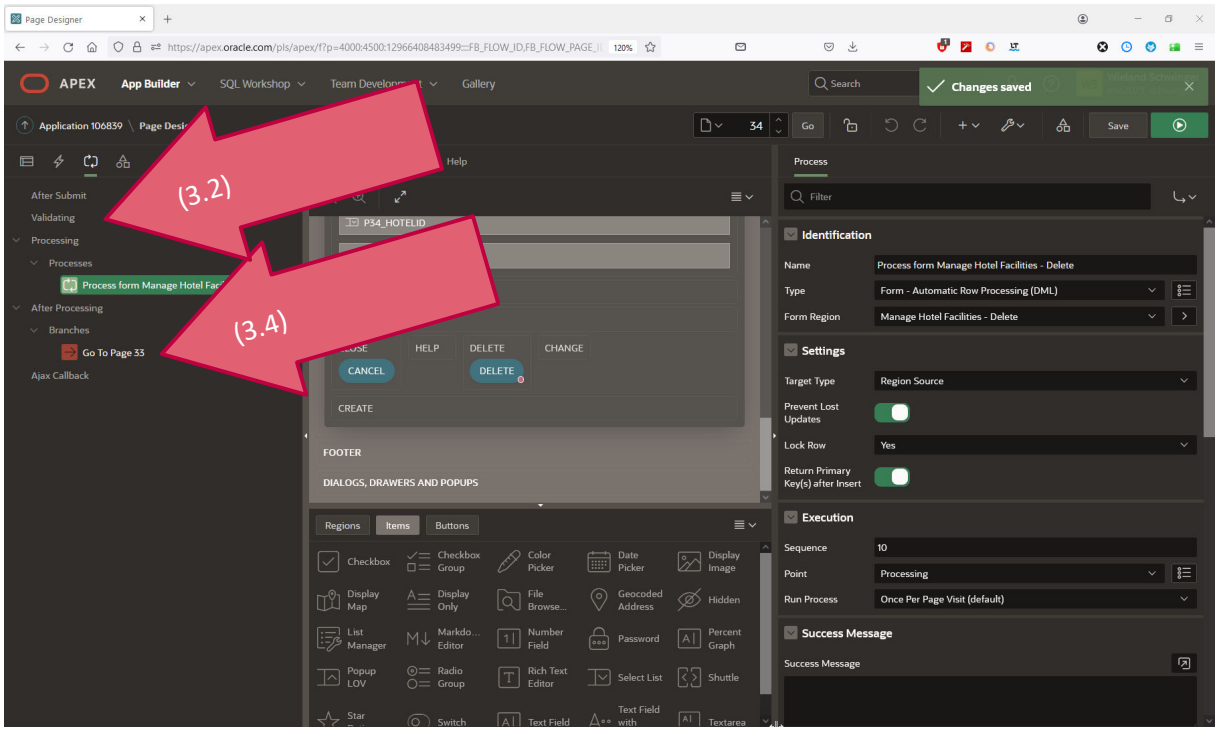
- If we give it a try to **"Delete"** the entry you see that you are **prompted** before the actual delete due to the activated confirmation.



- After **"Delete"** we are **redirected** to the **"Manage Hotel Facilities"** again.



- That we are redirected to the **"Manage Hotel Facilities"-page** is set in the **branch** specification and can be altered if a different page is required to be navigated to after the action.



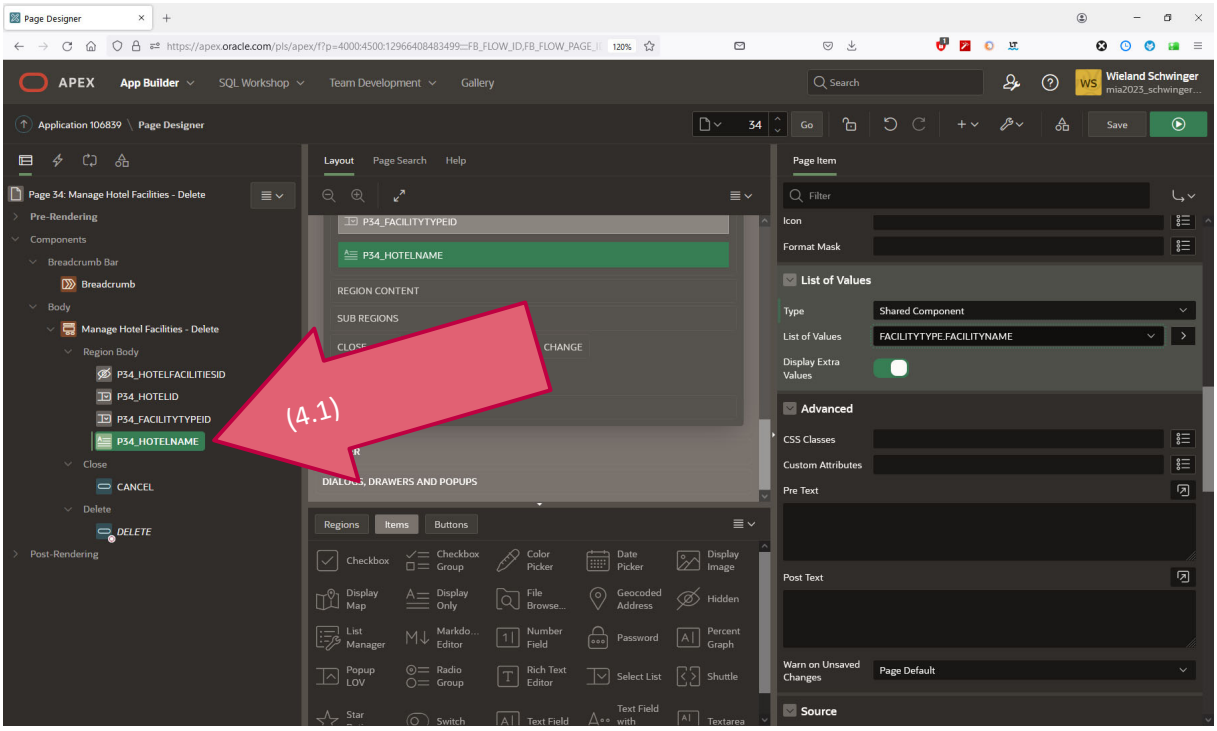
4) Changing from form-fields to display only

You may have seen on the **"Manage Hotel Facilities – Delete"** page **renders** the **hotel name** as well as the **facility type** which should be deleted for the hotel in terms of **selection list**.
This means that these actually could be changed – something we probably don't want here.
To change this we proceed as follows:

- Since actually **hotel name** as well as the **facility type** are filled from **selection lists** (thus they can be selected) we need **alternative items** for the hotel name and the facility type.

- **Add an additional item** of type **Display Only** to the **body region** of **"Manage Hotel Facilities – Delete"**.
- Name it **"P34_HotelName"**.
- Under **Source** select type **"SQL Query (return single value)"**.
  Specify the **SQL Query:**

```
SELECT Name FROM Hotel WHERE HotelID = :P34_HOTELID;;
```



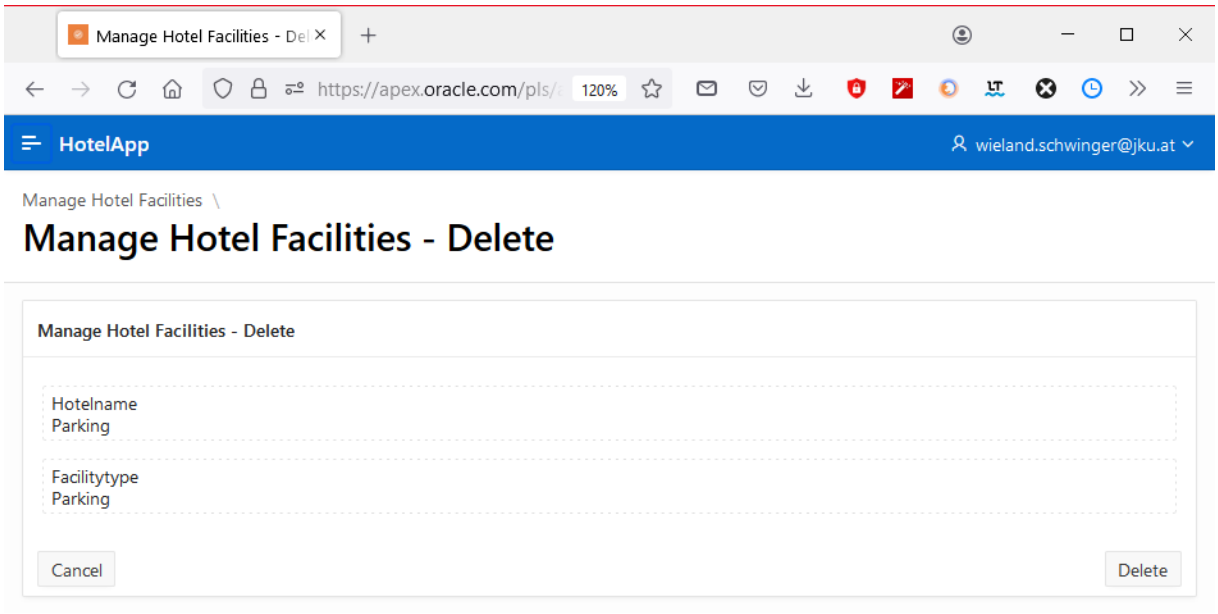
- **Add an additional item** of type **Display Only** to the **body region** of **"Manage Hotel Facilities – Delete"**.
- Name it "**P34_FacilityType**".
- Under **Source** select type **"SQL Query (return single value)"**.
  Specify the **SQL Query:**

```
SELECT FacilityName FROM FacilityType
WHERE FacilityTypeID = :P34_FACILITYTYPEID;
```

Both items are now "filled" through the result of the respective query.

- **Change** the **type** of the **other items** to **"Hidden"**.

- See the result.



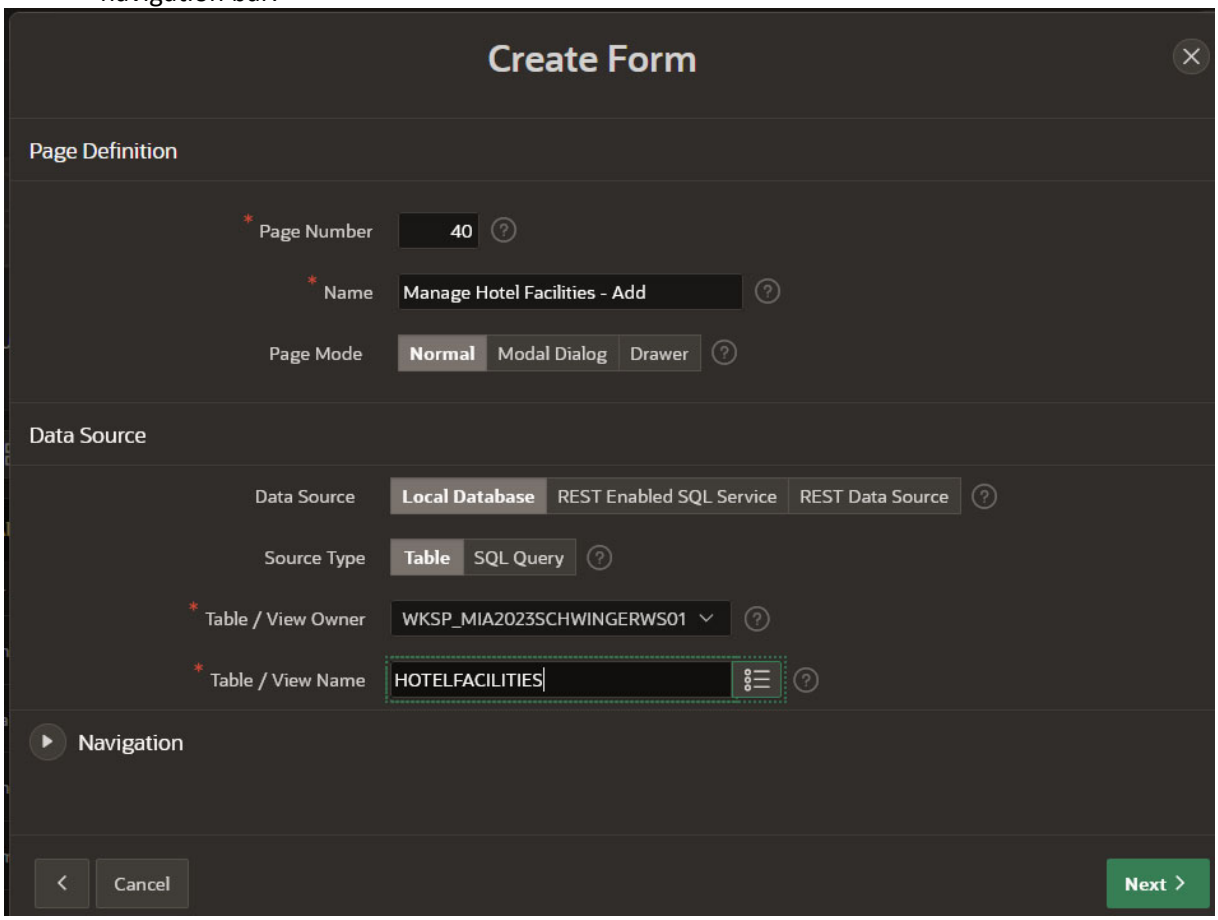
Deleting a facility type works now as intended.

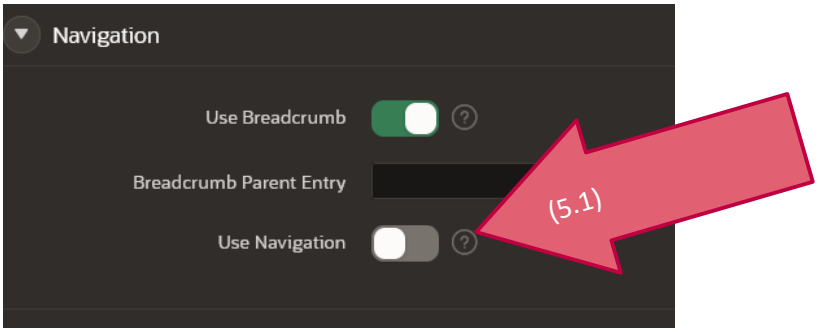
Next we will turn to adding a facility type to a hotel.

5) Adding a form to "associate" a facility type to a hotel

In the following we will learn how to use a form to insert information and in our case to exploit that to "associate" a facility type to a hotel.

- To achieve this, we will **Create a new page** of **type "Form"**.
- **Name** the page **"Manage Hotel Facilities – Add"** – in our case it has the page number 40.
- **Base it on the Local Database table "HotelFacilities"**
- **Under Navigation disable Use Navigation** – this will prevent the page to be listed within the navigation bar.





- Under **Branch Here after Submit** as well as under **Cancel and Go To Page** select your **"Hotel Facility Administration"-page**, in our case it is page 33.
  You can change the behaviour in the **Branch** through the **Page Designer** later too.

- Create the page with **Create Page**.

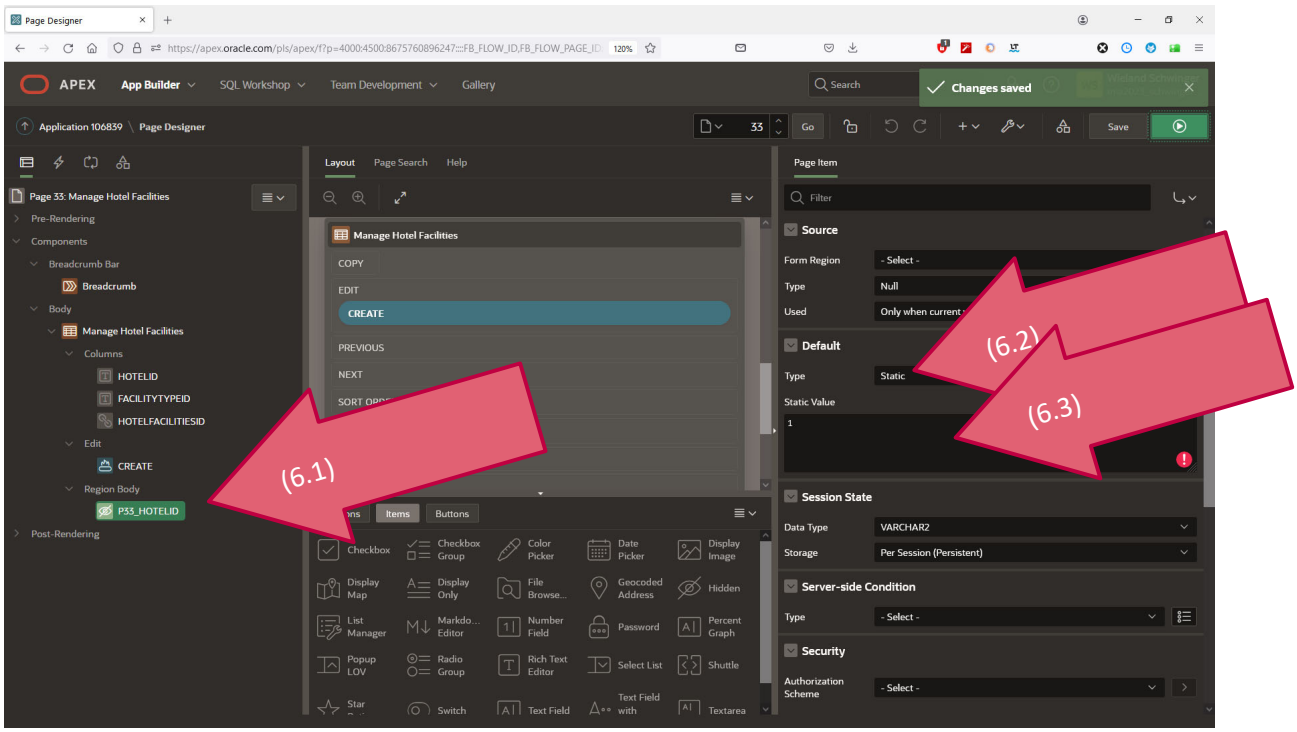
6) Constrain the List of Facility in "Manage Hotel Facilities" to a single Hotel.

Currently our page **"Manage Hotel Facilities" displays all hotels and their associated facility types.** For deleting those this is fine, however, for "adding" facilities to a specific hotel we do want to identify the specific hotel in beforehand.

We will – in preparation to link the page **"Manage Hotel Facilities"** to our existing page "**Manage Hotel"** constrain the list to **one specific hotel** in a first step.

- **Open** page **"Manage Hotel Facilities"** (being page in our case 33) in the **Page Designer.**
- **Add** a **Hidden item named "P33_HOTELID".** Drag it over/nearby **"Manage Hotel Facilities"** thus that a region body is created under which the item is placed.
- **Set the default** to a **static value (6.2)** to the **HotelID** value of a specific hotel **(6.3)**. We will later exchange this "fix" by passing a value from the calling page.
- **Add** a **WHERE-clause** to the **region** to select only those hotel facility entries that belong to a specific hotel:

```
HotelID = :P33_HOTELID
```

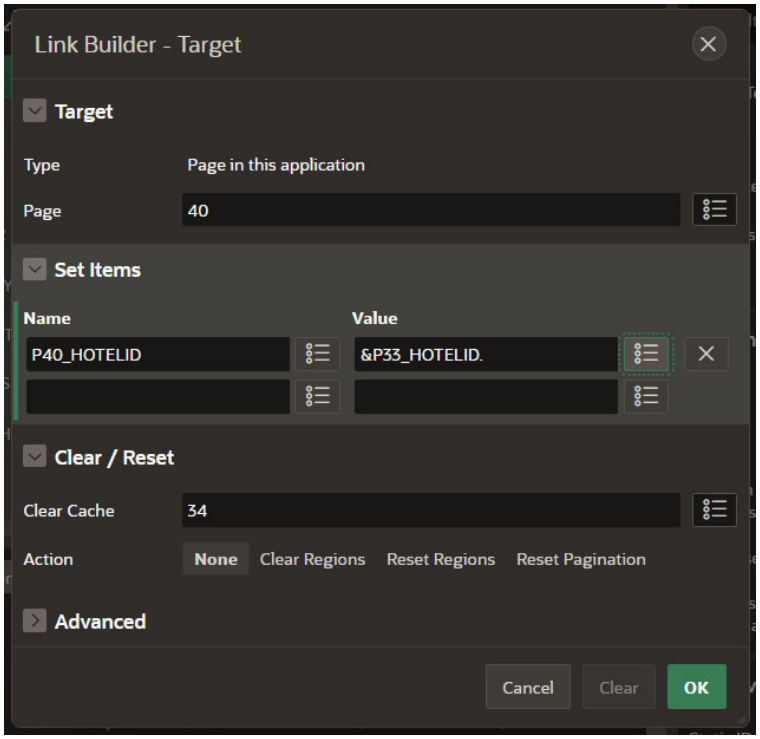


Now the list will show only the hotel and its associated facility type you specified for **"P33_HOTELID"**.

7) Associate "Create" of Page "Manage Hotel Facilities" with Page "Manage Hotel Facilities - Add"

It remains to associate the link of the "Create"-Button of the page **"Manage Hotel Facilities"** to redirect to our newly created page **"Manage Hotel Facilities – Add".**

- **Open** page **"Manage Hotel Facilities"** in the **Page Designer.**
- **Select** the **"Create"-button**
- Change the **Behaviour** by setting the **Target** to our page **"Manage Hotel Facilities – Add"** – in our case it has the page number 40.
- **"Pass"** to **"P40_HotelID"** the **"HotelID"**. We have this value available in **"P33_HOTELID"**.



In the following we will learn how to use a form to insert information and in our case to exploit that to "associate" a facility type to a hotel.

To achieve this, we will **Create a new page** of **type "Form"**.

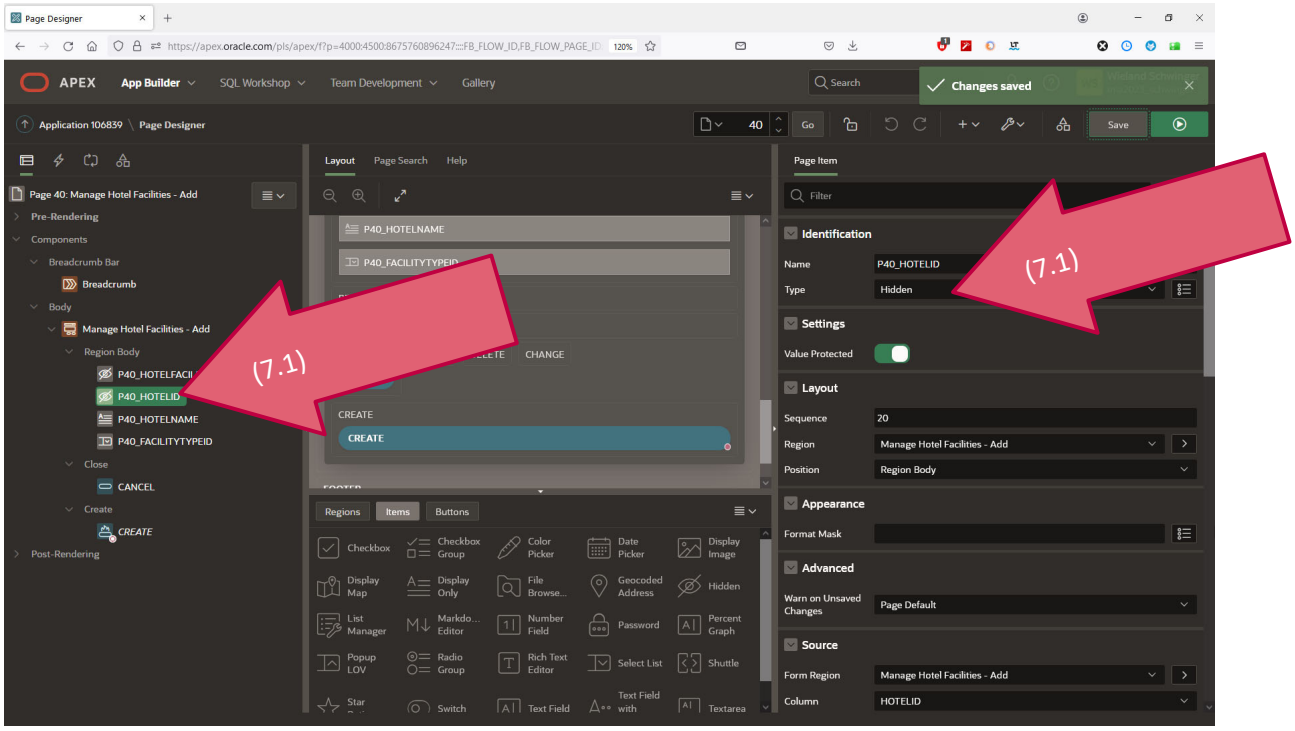
Turning to the **Page Designer** we now need to adjust the automatically created form page.

- **Add** an **item** named **"HotelName"** of **type "Display Only".**
- **Drag it** to the beginning.
- Set the **Source** of that item to **"SQL Query (return single value)"**:

```
SELECT Name FROM Hotel
WHERE HotelID = :P40_HOTELID;
```

- Set the **type** of **"HotelFacilityID" (7.1)** and **"HotelID"** to **"Hidden" (7.2).**

- **Remove** the buttons for **"DELETE"** and **"SAVE"**.



- **Save Page**.

See the resulting functionality.

- Navigate to page **"Manage Hotel Facilities"**
- Use the **"Create"-button.** You are **redirected to** page **"Manage Hotel Facilities – Add"**
- Here you can now **select** a **facility type** from the **drop box.**
- Use **"Create"** to save the associated facility type to the hotel. You are **redirected to** the **"Manage Hotel Facilities"** page**.**

You may find that you are able to associate a facility type multiple times to a hotel. While this should have been constrained for the table in the first place, we want to address this in the application. For this in the next step, we will only make those facility types selectable in the drop down list which have not been associated to the hotel yet.

8) Use a subset list for selection

Currently, in page **"Manage Hotel Facilities – Add"** all possible facility types are populated to the drop down list and thus can be chosen (again) to be associated to a hotel. We are changing this now.

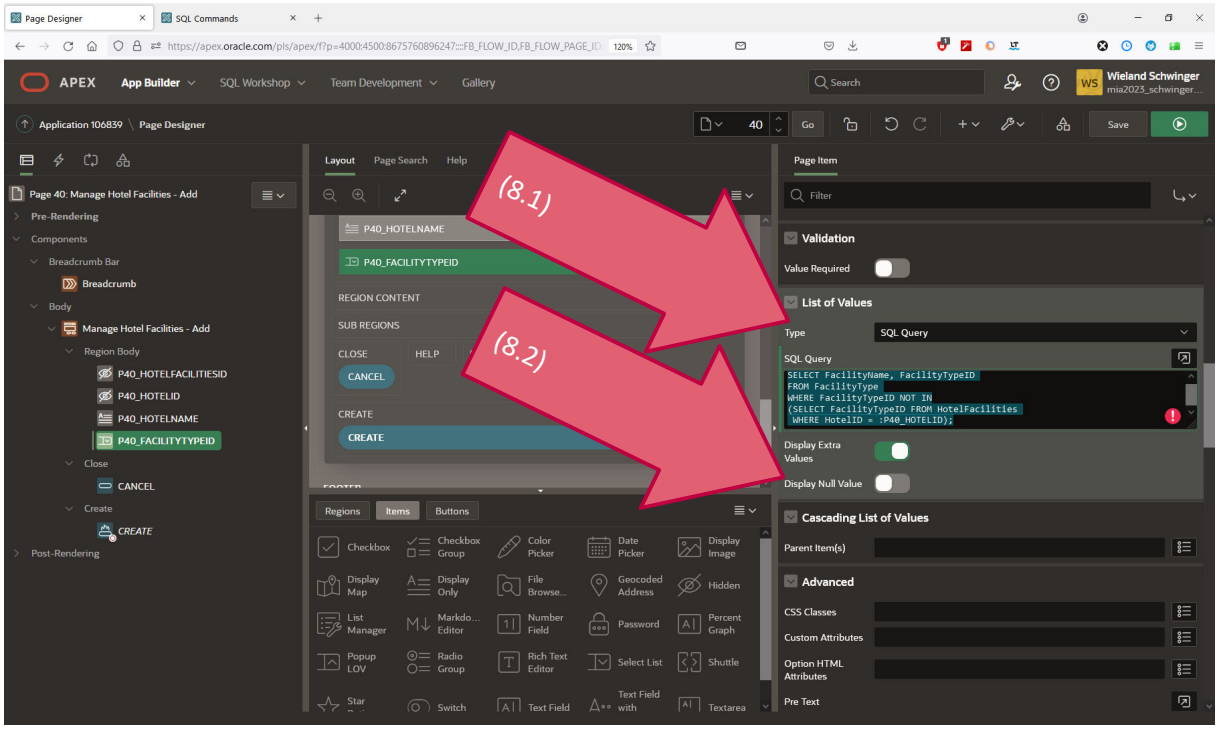
- **Open** page **"Manage Hotel Facilities – Add"** in the **Page Designer.**
- Under **List of Values (8.1)** set the **type** to **"SQL Query"**
- Specify the **SQL** as:

```
SELECT FacilityName, FacilityTypeID
FROM FacilityType
WHERE FacilityTypeID NOT IN
(SELECT FacilityTypeID FROM HotelFacilities
 WHERE HotelID = :P40_HOTELID);
```

  It selects only those facility types for the selection list which are **not already associated** to the hotel.

  For the **drop down list**, the **SELECT** specifies two columns to be selected. The **first** (i.e., FacilityName) acts for the drop down list as the **displayed value**, whereas the **second** column selected (i.e., **FacilityTypeID**) is **returned** when selected by the user.

- **Disable Display Null Value (8.2)** – this will prevent that also a null value can be selected. Again this is something which should have also addressed in the database, which we skipped due to simplicity of our example.



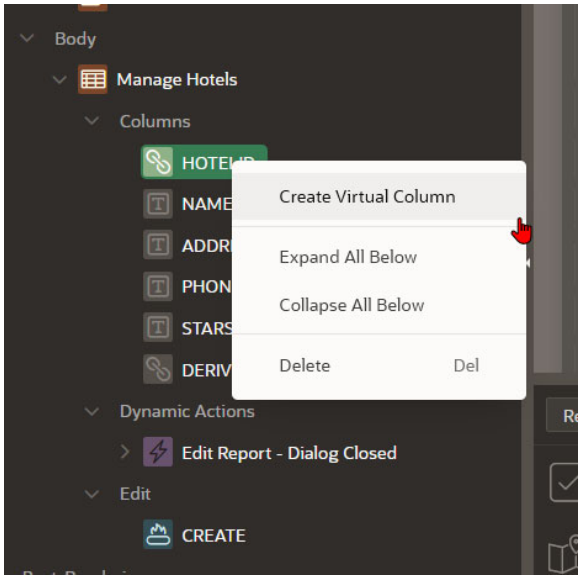
- **Save Page.**

- Observe the changed behaviour of your app when selecting to **"Create"** in **"Manage Hotel Facilities – Add"**

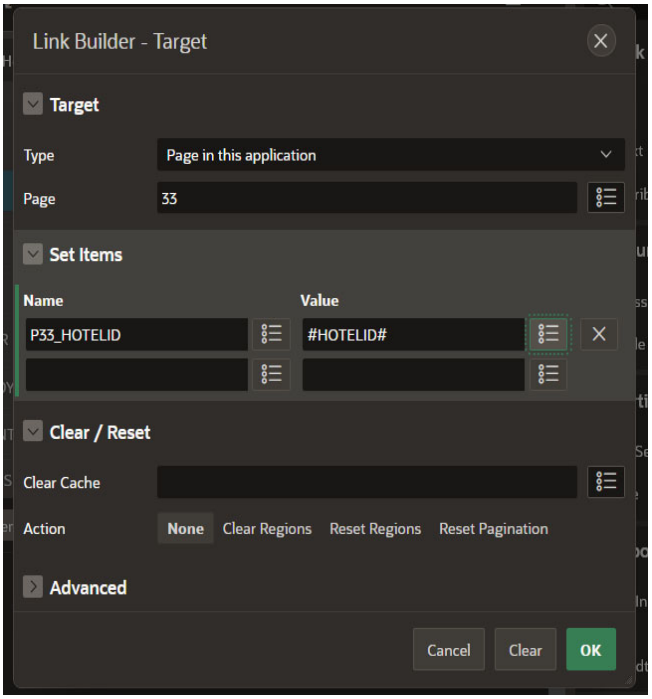
8) Use a subset list for selection

Currently, in the **HotelID** in page **"Manage Hotel Facilities"** is set to a fixed default value.
We will now chose the hotel for this from our **"Manage Hotel"** page.

- **Open** page **"Manage Hotel"** in the **Page Designer.**
- **Add** a **virtual column** to the **"Manage Hotel"** **region** through the **contextual menu**



- Set the **type** of this new virtual column to **link**.
- Under **Link** specify our page **"Manage Hotel Facilities"** as our **target.**



- **"Pass"** on the **HotelID.**
- As **link text** use **"manage facilities>"**

- **Save Page.**

- Now, observe what we have achieved!

    - We have our **"Manage Hotels"** to add, delete and change hotel information.

    - Additionally, we can now also manage a hotel's facilities since the **"Manage Hotel Facilities"** is linked from each hotel in the **"Manage Hotels"** page and the **HotelID** of the respective hotel is passed on to the **"Manage Hotel Facilities"** page.
    - The **"Manage Hotel Facilities"** links to two further pages:
      1) to be able to **create** a new entry in HotelFacilities for which the HotelID is passed to the **"Manage Hotel Facilities – Add"** page and there the not yet used facility type can be chosen.
      2) to be able to **delete** an entry for which the HotelFacilitiesID is used to delete that specific entry.

Up to now the tutorial should have learned you:

- How to **create tables** with **Object Browser**.
- How to **create 1:n-relationships** and **n:m-relationships** through **foreign keys** and **mapping tables** respectively.
- How to set **constraints**.
- How to **insert data** using Object Browser.

- How to create a Web app using **App Builder**.
- How to create a **page** with the wizard.
- How to **select data** for a region (classic report).
- How to **create individual items** and **select data** for these items.
- How to **link pages** together and pass on information.
- How to **add management operations** using the form wizard.
- How to **fine-adjust** the management **operations** on a page.
- How to work with **hidden items** to contain information on a page.
- How to **populate list of values**.

- How to **link pages together** to make up meaningful navigation.

16) Further Page Types

From here on you are called to investigate further into the functionalities offered by APEX.
Try out the following:

***) Use the page creation wizard to create pages of other types.**
Most useful may be:

- Blank page
- Interactive Grid
- Interactive Report
- Cards
- Faceted Search
- Smart Filters
- Search Page
- Master Detail

17) List of Values

We have used the List of Values feature in our Web app in two forms so far: i) pre-generated list of values and ii) by using an SQL-query in the source of an item.

However, you may define *reusable* List of Values on your own.

- Go to **App Builder**
- Go to **Shared Components**
- Go to **List of Values**
- There you see already defined List of Values for your app. You can also create a List of Value on your own using **Create**.
    - Static List of Values only contain those pre-specified fixed values
    - Dynamic List of Values load the available list entries during run-time either from a DB-table or through an SQL-query (amongst other options).
- In any case for a list of values always the display value and the return value need to be identified.

18) Navigation "Lists" & "Breadcrumbs"

APEX generates, if not disabled, entries for the **menu** of the application. This is called "list". To manipulate this "navigation list" explore this feature:

- Go to **App Builder**
- Go to **Shared Components**
- Go to **Lists**
- Go to **Navigation Menu**

Here you can see and manipulate the entries in the menu (per default available on the left hand side of your page).

Additionally, the **"Breadcrumbs"** can be managed under "Lists" too.

- Go to **App Builder**
- Go to **Shared Components**
- Go to **Breadcrumbs**

19) Access Control

ORACLE APEX offers a sophisticated access control schema. It comprises of an **Authentication Scheme** and an **Authorization Scheme**.

1) Authentication Scheme
This is how users are identified. The default is that users are authenticated by their APEX login. However other Authentication Schemas would be available. For example authentication could be based on the database users, on a LDAP directory or on social sign-in.

An Authentication Scheme is a saved configuration you can apply to your application.
To turn off authentication you must create and make current a **No Authentication** scheme.

2) Authorization Scheme
You manage authorization for applications, pages, and most other components using Authorization Schemes.
You associate an Authorization Scheme with an application, page, or other component on the component's attributes page.

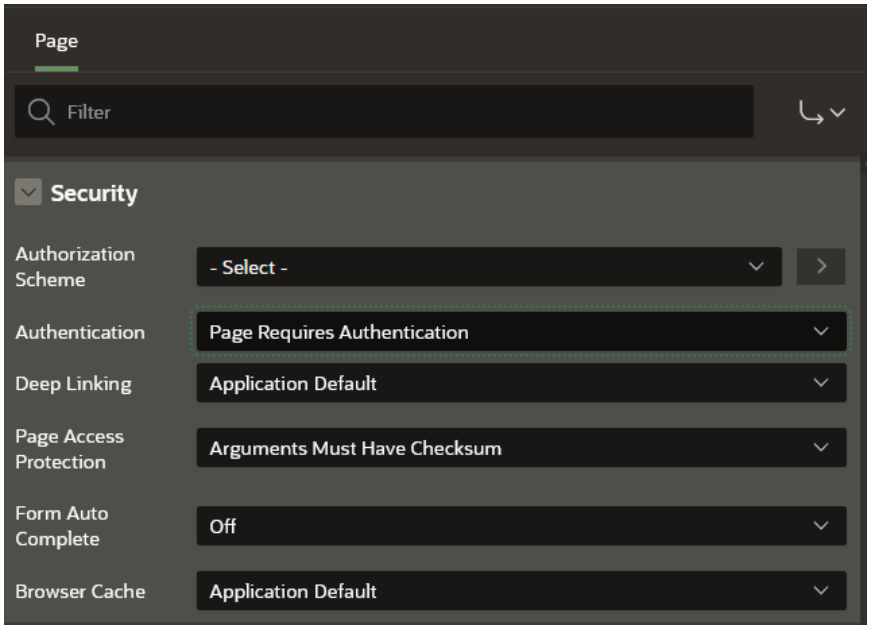
Authorization Schemes are pass/fail checks. Common Authorization Scheme types include Exists, Not Exists SQL Queries, and PL/SQL Functions Returning Boolean.

3) Controlling Access on Page Level

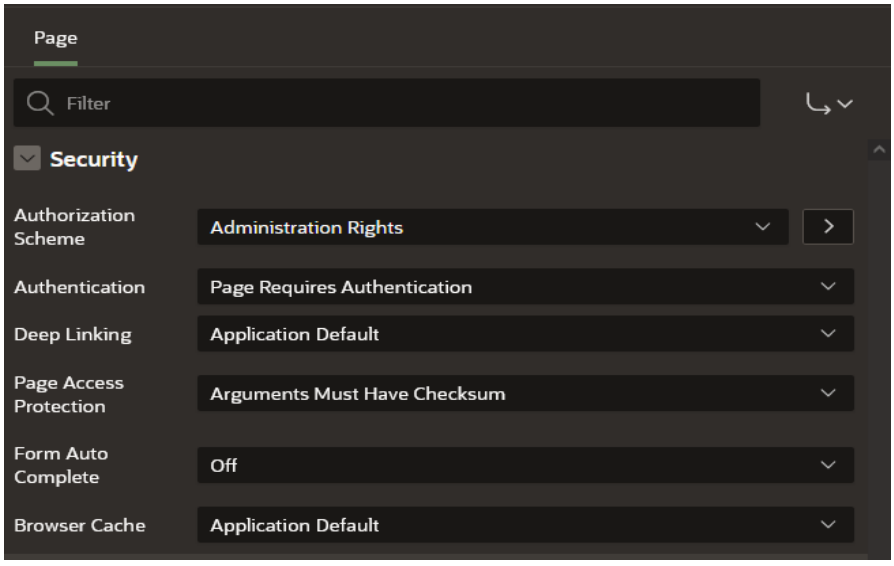
You can control how (or better to say how under which condition, cf. Authentication vs. Authorization) a user is able to access a page.

This is done for a page under **Security**:

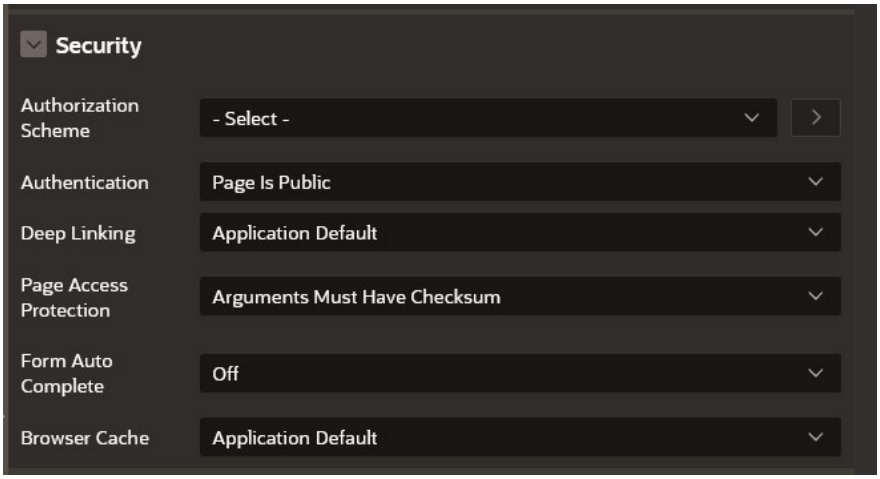
USER IDENTIFIED - Authentication without Authorization.
Here it is stated that the page requires authentication but no specific authorization schema need to apply.



ADMINISTRATOR ONLY - Authentication with Authorization as "Administrator".
In contrast here it is stated that the user is required to have Administration Rights. Administration Rights define as Authorization Scheme that the user needs to be in the Administrator Group.



PUBLICLY ACCESSIBLE - No Authentication and No Authorization.
Here is specified that a page is public without the requirement of authenticating through a login.



4) Personalizing to the Current User

Within the application the currently logged in user is accessible as follows.

APP_USER is the current user running the application.

Depending upon your authentication model, the value of the user is set differently. If the application is running using database authentication, then the value of the user is the same as the database pseudo column USER. If the application uses an authentication scheme that requires the user to authenticate, the value of APP_USER is set by the authentication scheme, usually to the user name used during authentication.

You can utilize the **current user** as follows:

- **Bind variable** **:APP_USER**

  ```
  SELECT * FROM some_table WHERE user_id = :APP_USER
  ```

- **PL/SQL** **V('APP_USER')**

  ```
  htp.p('Hello you are logged in as'||V('APP_USER'));

  SELECT v('APP_USER') from dual
  ```

- **Substitution string** **&APP_USER.**

  (HTML region) `Hello you are logged in as &APP_USER.`

- **Using the variable:** **SYS_CONTEXT**

  ```
  SELECT ... WHERE
  username = SYS_CONTEXT('APEX$SESSION', 'APP_USER')
  ```

APEX sets up the **APEX$SESSION** context when it starts to process an incoming request.
For example, you can use the value of '**APP_USER**' to access the current application user in queries.