

# Introduction to Database Reverse Engineering

FROM THE SPACE FLIGHTS EXERCISE  
STUDENTS VERSION V1.0 (22/10/2023)

DAVID BUENO VALLEJO



## Content

1	Introduction .....	3
2	Problem description .....	3
3	Preparing the database .....	4
3.1	Database cleanup.....	4
3.2	Load the model from SpaceFlight.sql.....	6
3.3	Watching the data in Data Modeler (option A).....	8
3.4	Watching the data in Data Modeler (option B).....	12
3.5	Workflow.....	15

## 1 Introduction

These are student notes to understand how to make Reverse Engineering using SQLDeveloper and DataModeler. We will see the different situations (modify attributes, change relations, correct data,...) that can be found in the reverse engineering exercises through the example of space flights.

## 2 Problem description

In the year 2321, the NASA has a secret project where they collect information about the different species that have been found in the Universe and the planets where they live. They asked to the company CheapSoftware to create this database, but they didn't make a good work. That's why they require us to modify the database to obtain the desired result of the Fig. 1. Where we can see that we must represent the different planets, and species. But also, there are many intelligent species inhabiting planets in the universe. In a planet can live more than one specie and an intelligent specie can control other species. The intelligent species can live in different planets that their original one. These species and their characteristics are included in the model. The following entity/relationship model has been chosen for its implementation, but the bad relational design of CheapSoftware must be updated until it exactly matches the following model:

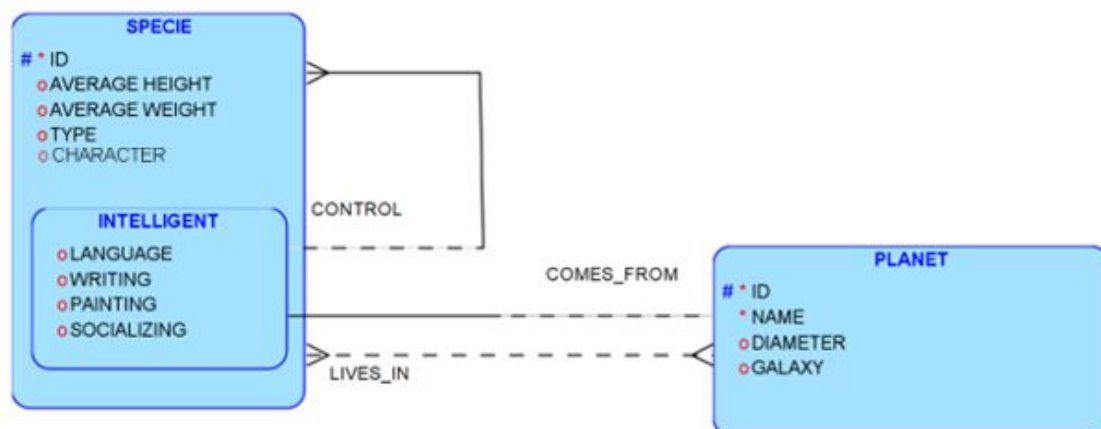




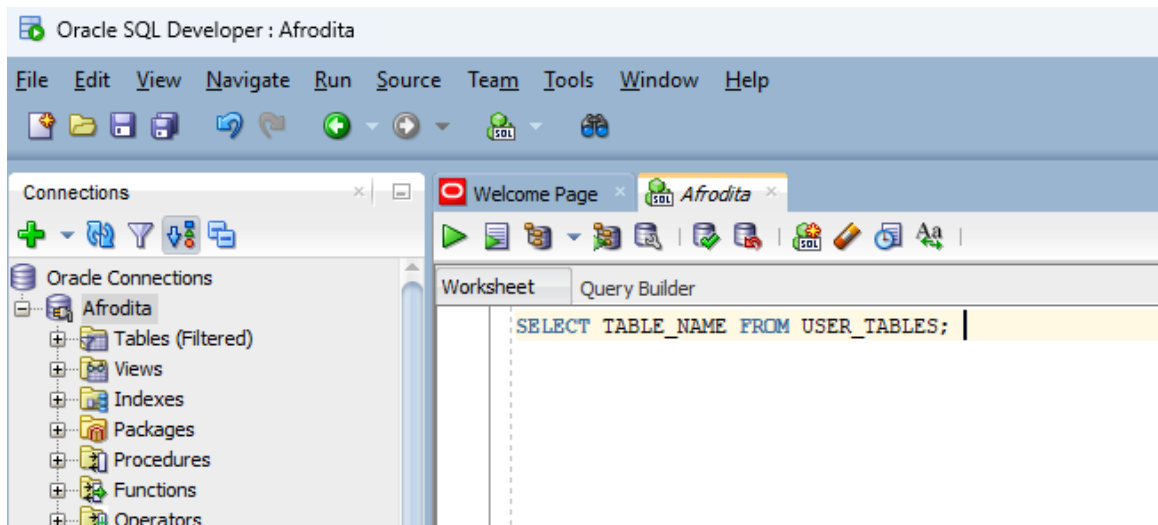
Fig. 1. Desired ER model

In addition, you must consider the following issues:

- The **TYPE** of species indicates whether it is **NOMAD** or **SEDENTARY**
- If the **TYPE** is **SEDENTARY**, the character appears as **NULL** but if it is **NOMAD**, the character can be **WARRIOR** or **PEACEFUL**.
- Initially the **TYPE** value is **SEDENTARY**.

### 3 Preparing the database


Before starting it is good to delete all user tables inside SqlDeveloper. We will go to the Worksheet where we can execute SQL Code. If we click F5 (  ) all the lines will be executed, and the result is shown as plain text. If we only want to execute one of them, we put the cursor in the line and click Ctrl+Enter (  ) and we will get a tabular result.



#### 3.1 Database cleanup

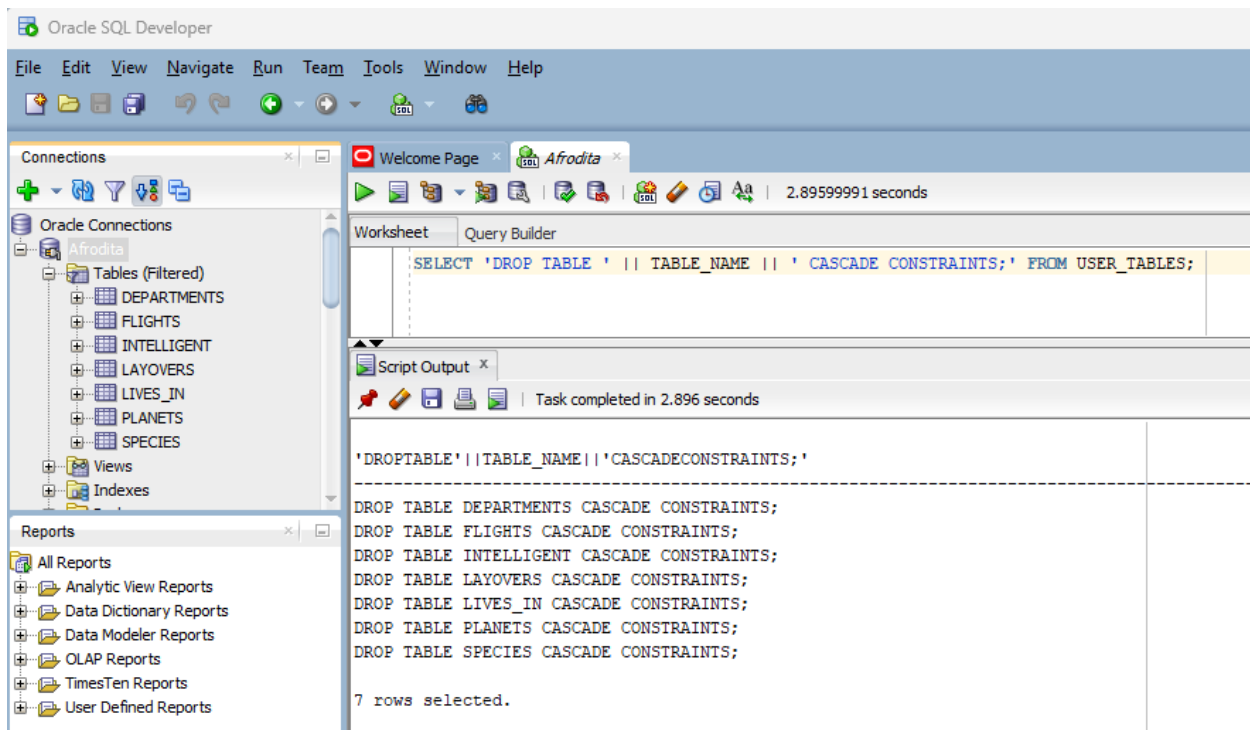
To see all the tables:

```
SELECT TABLE_NAME FROM USER_TABLES;
```

To delete all tables, we need two steps. The first step consists of obtaining a list of all the tables. We can obtain the list with this instruction. (Select F5(  ) because we want to obtain the plain text)

```
SELECT 'DROP TABLE ' || TABLE_NAME || ' CASCADE CONSTRAINTS;' FROM USER_TABLES;
```

The result is a list like this (in the example we had 7 tables so we will get 7 drop instructions. If your database is empty you will get no result):



Copy and paste into the worksheet the generated drop instructions and run the script. After it all the tables will be deleted.

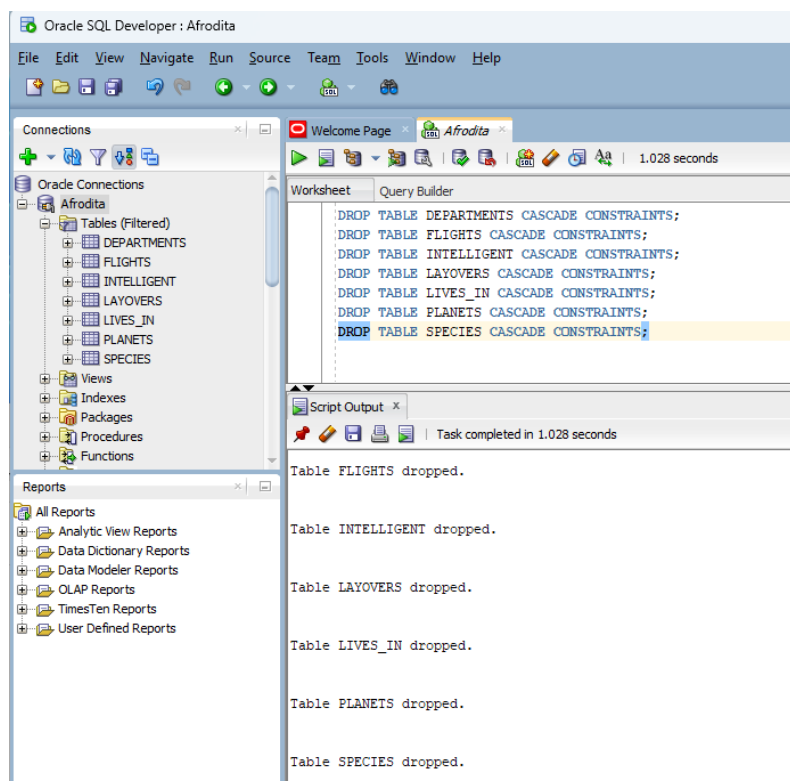



Fig. 2. Deleting the tables

But to see that there is no table in the database, place the cursor on the Tables Folder (top left frame), and click the refresh button .

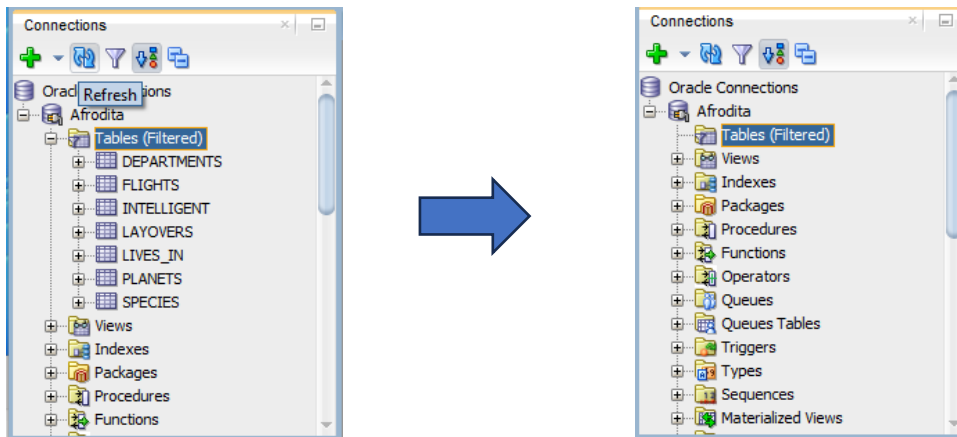
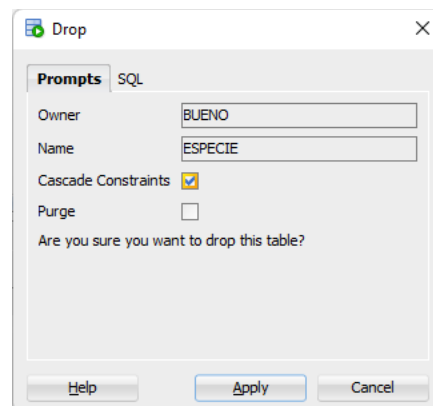


Fig. 3. Refreshing tables to see the changes

We can also delete tables visually one by one with the interface with the right button on the table by checking the cascade. Right button on table → Table → Drop




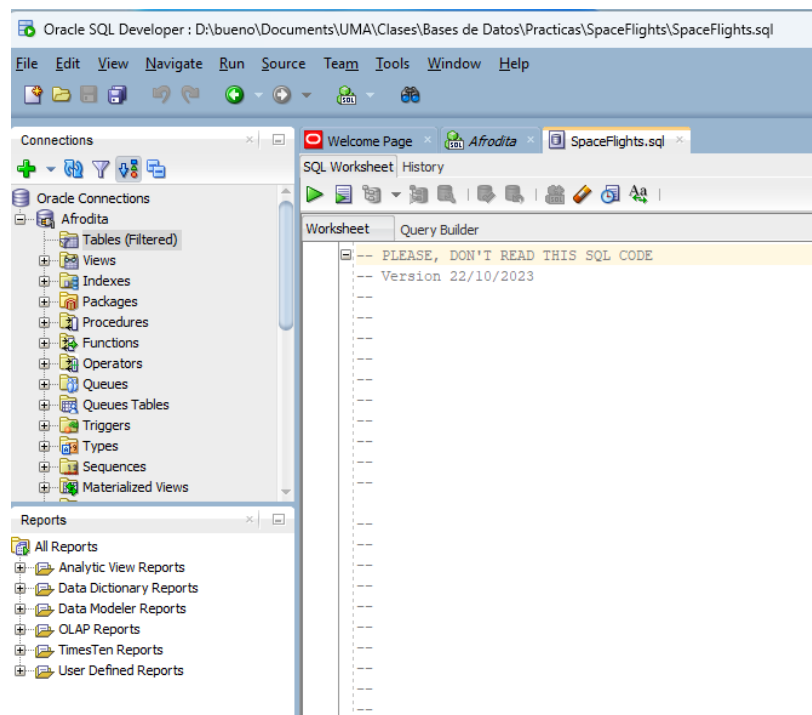
### 3.2 Load the model from SpaceFlight.sql

Our main goal is to modify the database done by CheapSoftware maintaining the important information included in the database about species.

We have the current database of CheapSoftware in a file named: SpaceFlight.sql. We will load in SQLDeveloper and then we will make reverse engineering to obtain the ER and to compare the current diagram with the desired one. Then we will make changes in SQLDeveloper (NOT in DataModeler) to refine the diagram and from time to time we will check if the ER looks like the desired one.

IMPORTANT NOTE: If we make changes, they cannot be sent from data modeler to Sql Developer since the database has data and they could be lost. Only make changes in SQLDeveloper.

From the sql. SpaceFlights.sql you must paste the script into SQLDeveloper Worksheet (or drag and drop the file on it) and run it as a script with F5(  ):



After running the script you must refresh the Tables as we did in Fig. 3. We will found the three tables with the following content.

	LANGUAGE	WRITING	PAINTING	SOCIALIZING	SPECIE	COMES_FROM	CONTROL
1	Gramatino	Depurate	y	n	10001	22222	20002
2	Levuorin	Begining	y	n	20002	33333	30003
3	Saporo	Unknown	n	n	30003	44444	40004
4	Arabino	Depurate	y	n	40004	55555	50005
5	Aghaga	Depurate	y	y	50005	66666	60006
6	Espatul	Unknown	y	n	60006	77777	70007
7	Forensic	(null)	y	y	70007	88888	10001

Fig. 4. Original table INTELLIGENT

	ID	NAME	DIAMETER	GALAXY	ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	LIVES_IN
1	22222	Sigfrido	30000	Lexus	1	10001	2,3	120,4 (null)	22222
2	33333	Nova Terra	4000	Orion	2	20002	1,3	120,4 (null)	22222
3	44444	Esperides	30000	Orion	3	30003	1,3	60,4 (null)	22222
4	55555	Sigfrido	22000	Aspen	4	40004	2,3	120,4 (null)	33333
5	66666	Andromeda	55000	Orion	5	50005	0,3	10,4 (null)	44444
6	77777	XP2002	8000	Orion	6	60006	4,3	160,4 (null)	77777
7	88888	Socrates	67000	Lexus	7	70007	2,9	90,48 (null)	88888
					8	80008	2,6	100,4 (null)	44444
					9	90009	2,5	70,4 (null)	77777

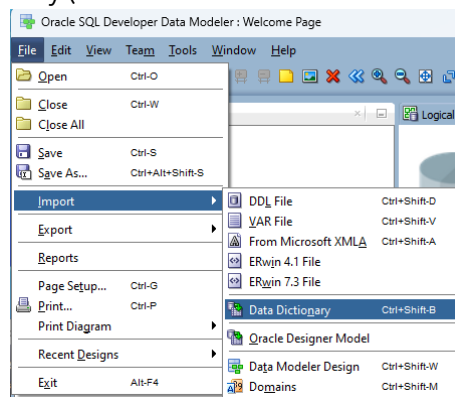
Fig. 5. Original table Planet and Specie



### 3.3 Watching the data in Data Modeler (option A)

Once the tables have been loaded and created, open the Data Modeler and import it with:

File→Import→Data Dictionary (This will take the information directly from the database)



We will have to create the connection to the database. The window that appears is Fig. 6, and we will select Add:

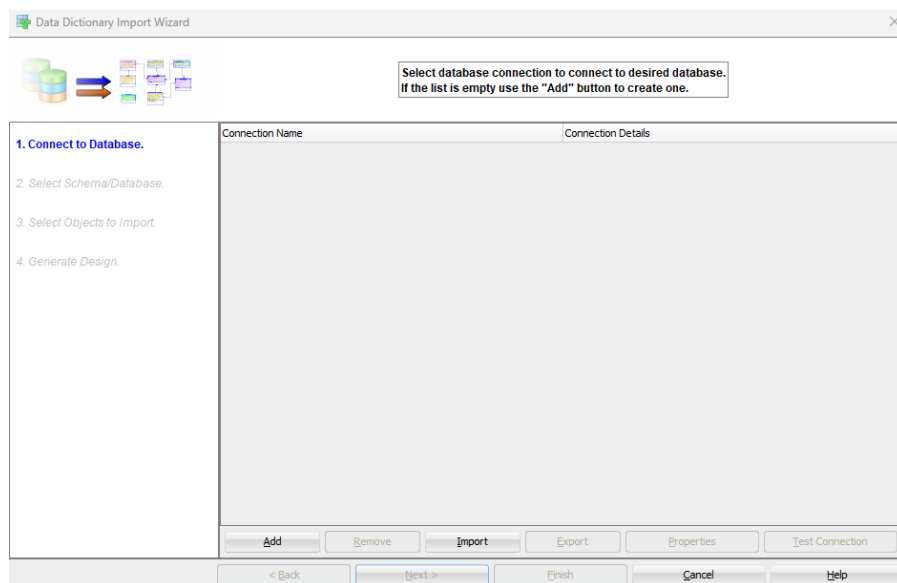


Fig. 6. Initial window to create a connection to the database in DataModeler

Then you get the same window that in SQLDeveloper to create a connection, where you must use your database user (UBDXXX). NOTE: In my case the user is BUENO. After inserting your data always Test the connection (with the button at the bottom) and the click Connect.

Fig. 7. Creating the connection to the database from DataModeler

After that select the new connection in the window and click Next

You will get the list of all the database Schemas, but you can only have access to your Schema (UBDXXXX) that you can filter you get yours typing in the filter textfield and then selecting in the checkbox as done in Fig. 8:

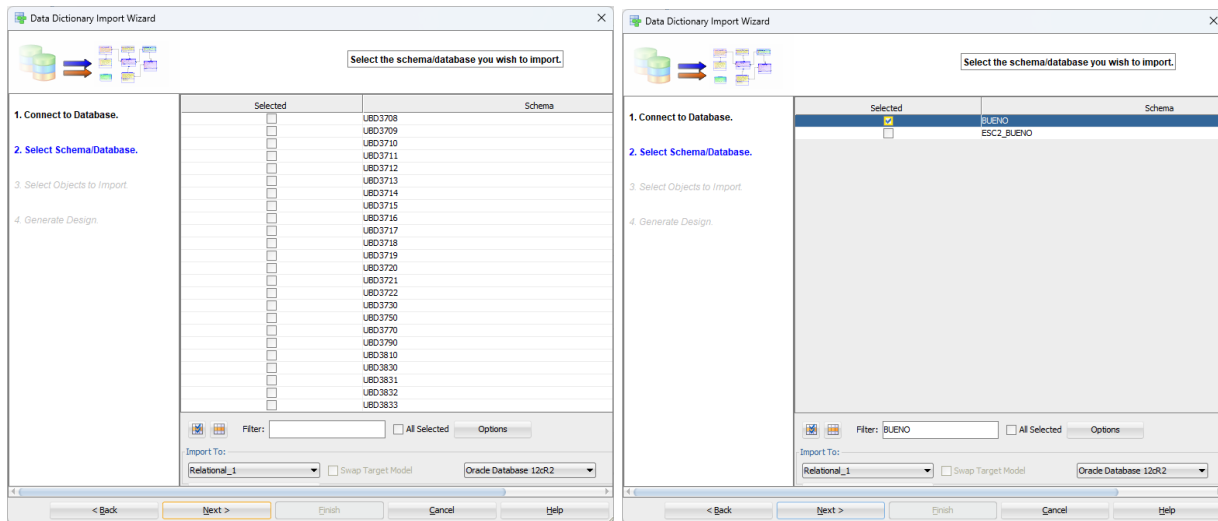
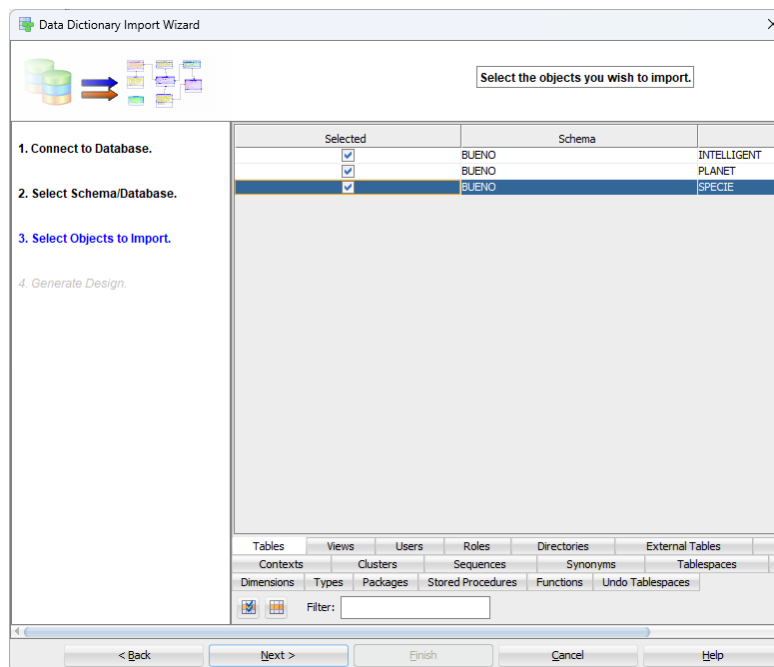
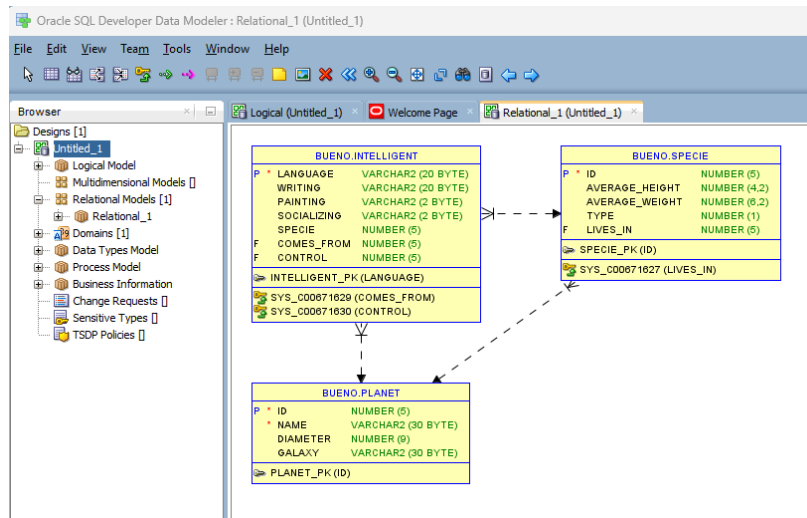


Fig. 8. Selecting the user Schema

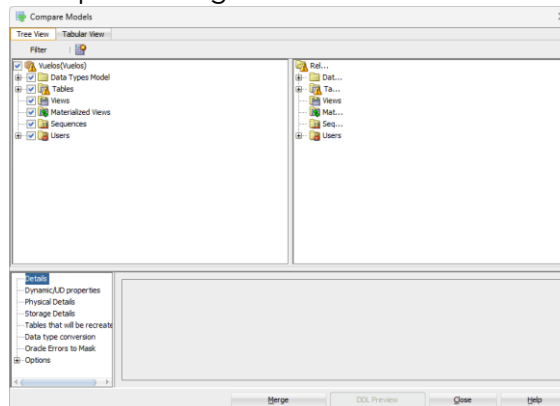
After clicking Next we will get the list of all the table in our database. We will select only the ones we want to get into Datamodeler (In our case the three) and the click next.




The model will be loaded and we will get the Relational Model created by CheapSoftware.



Sometimes, when importing it, it will ask us to merge it with the current model, which should be empty. Then we press Merge.



We Will pass the model to ER by pressing the: “Engineer to logical model”  in the menu bar. Then in the Fig. 9 we will select “Engineer”

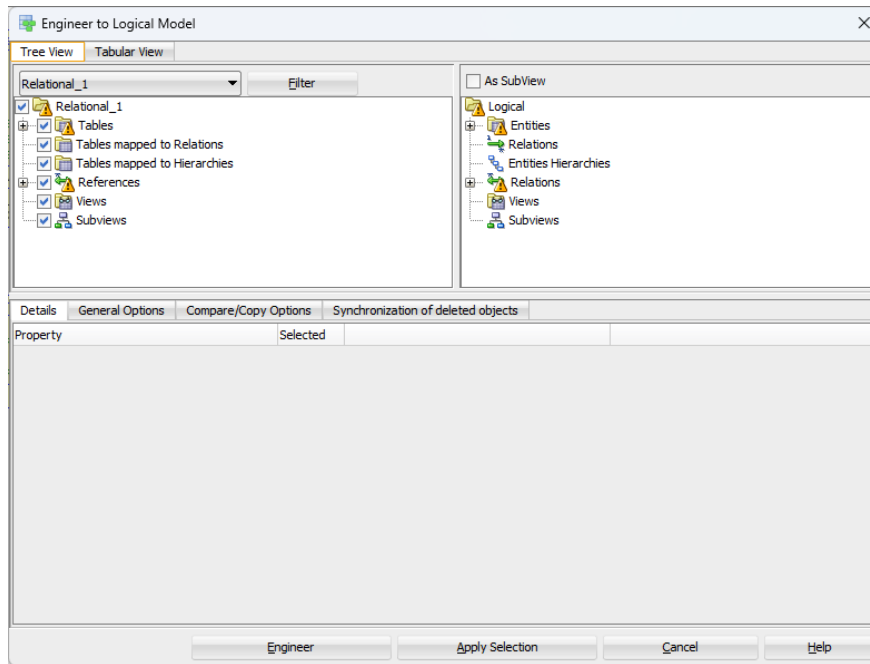


Fig. 9. Obtaining ER Model from Relational model in DataModeler

And we will obtain the following model:

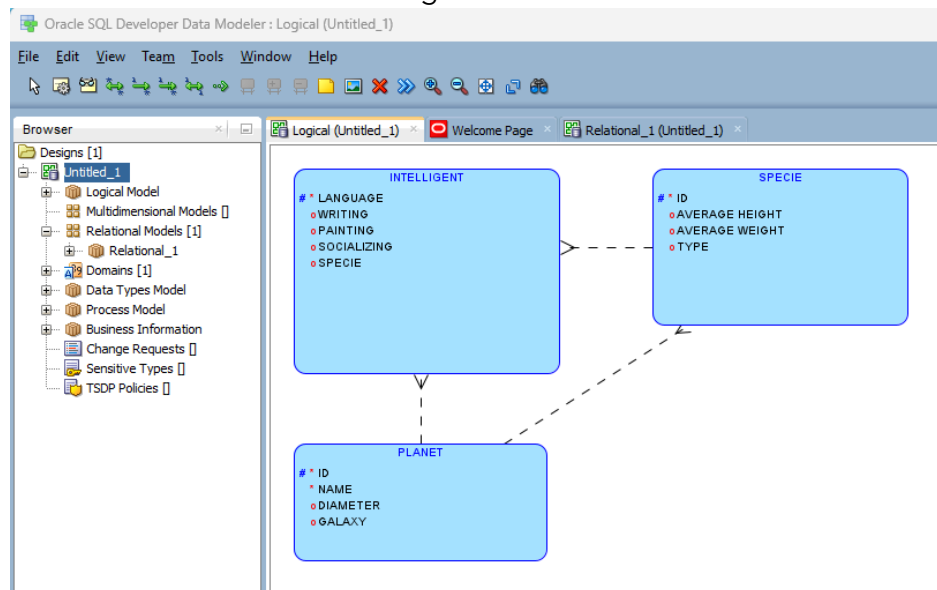
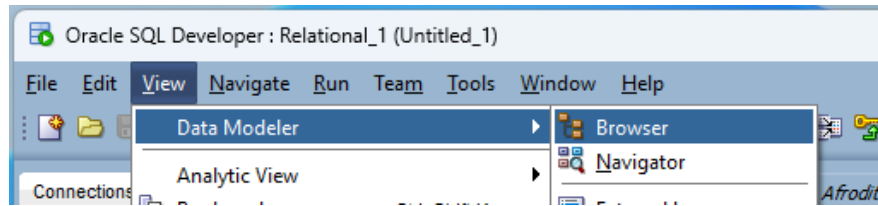


Fig. 10. ER Model created from the CheapSoftware database

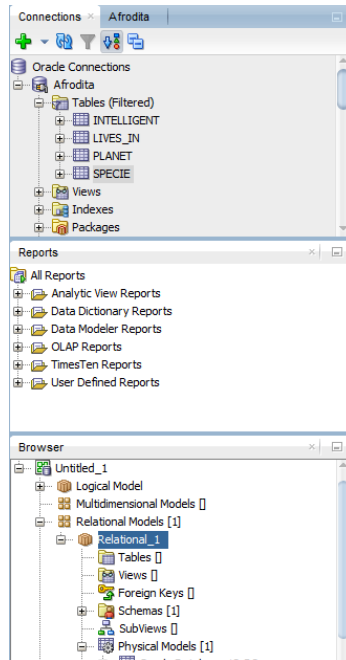
### 3.4 Watching the data in Data Modeler (option B)

There is another way to obtain the ER Model without exiting SQL Developer. This will work only if DataModeler is installed in the computer. This possibility consists of creating the diagrams inside SQLDeveloper, basically it is only a view of DataModeler inside SQLDeveloper. To view that we need to add a view of DataModeler.

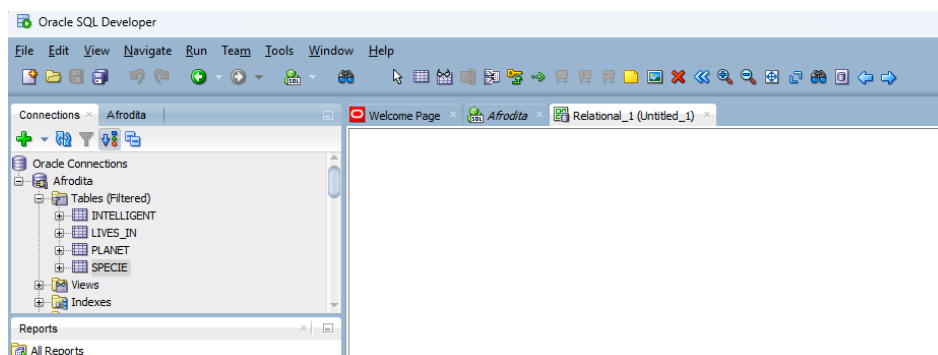
Inside SQLDeveloper in menu View→Data Modeler Browser.



This adds a new frame on the left, with the hierarchy of DataModeler.

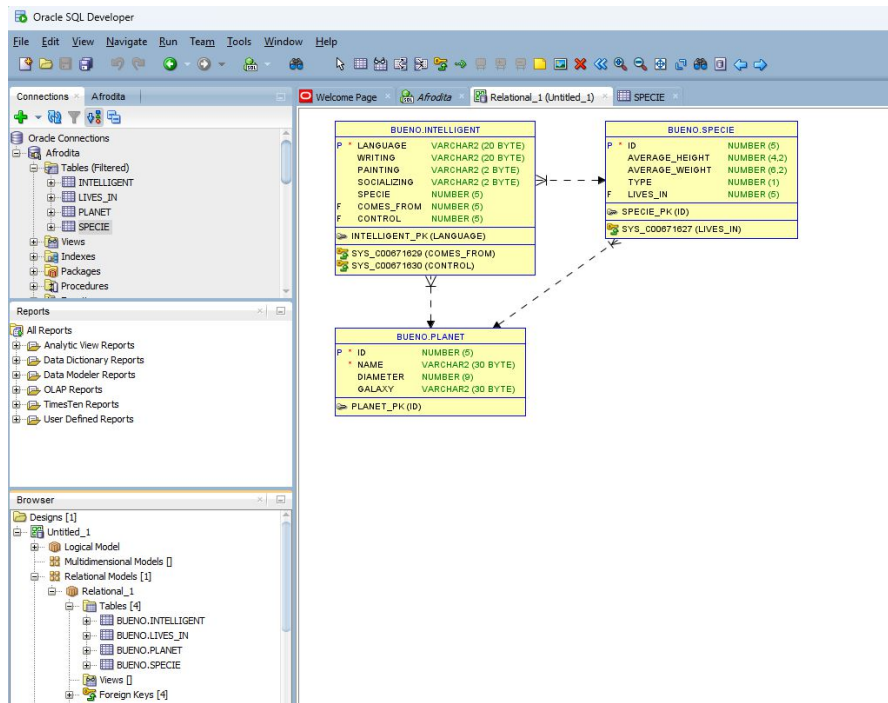



With the right button on Relational\_1 → Show we will get a new window Relational\_1



NOTE: The data of the example are not from the initial database (they are from the end of the section 4.5)

We can now drag and drop, one by one, the tables of our database (PLANET, SPECIE and INTELLIGENT) from the Oracle Connections to Relational\_1 and we will get:



Here we can make reverse engineering like in data modeler with the same button . Obtaining this diagram, that we can use to compare with the goal.

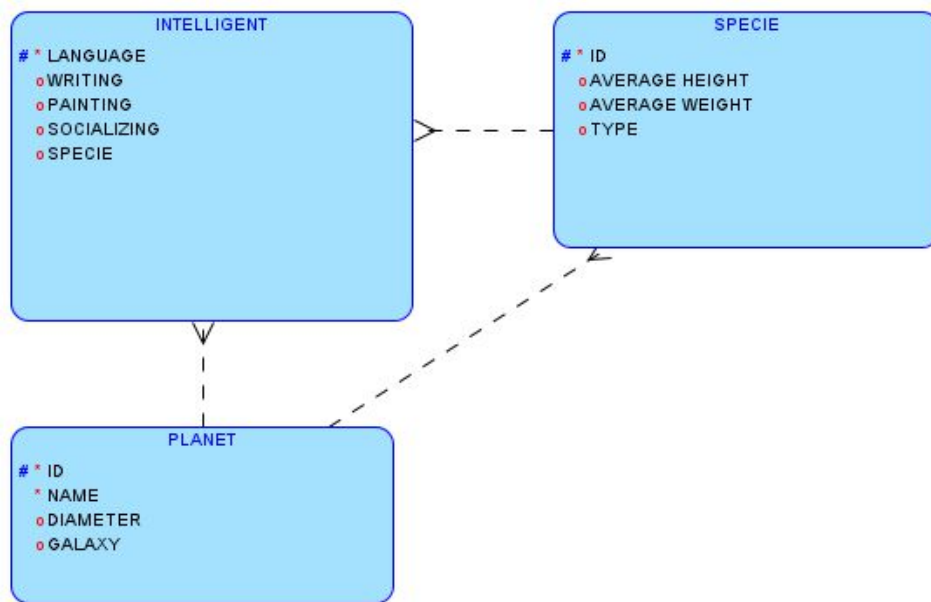


Fig. 11. Example of obtaining the diagram inside SQLDeveloper (for the end of the section 4.5)

### 3.5 Workflow

Our work from now will be to transform the existing database in SQLDeveloper to obtain the desired diagram. This means that we will be making modifications to pass from our current state to the desired diagram.

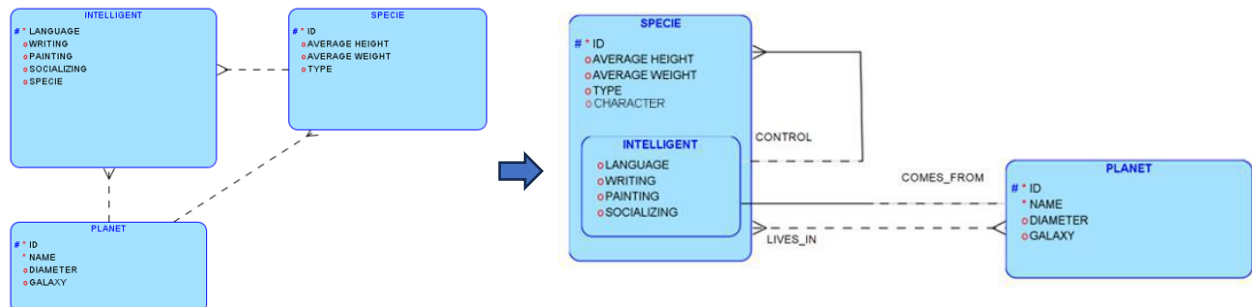

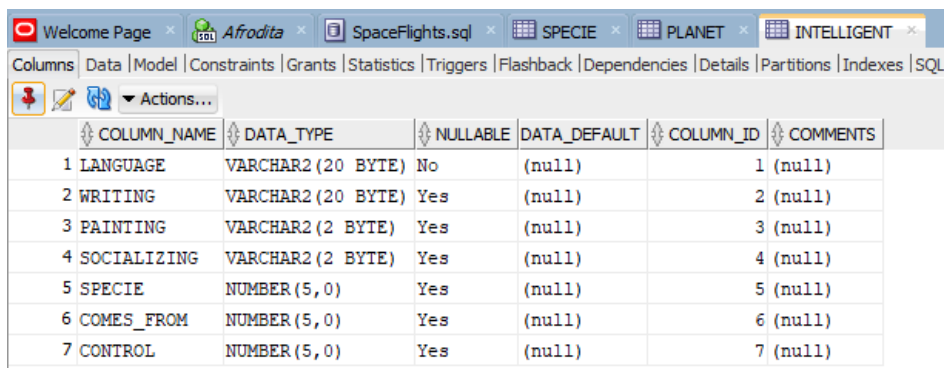


Fig. 12. Our work is to change the current state to the desired diagram

TIP: First change the strong tables, and then the relationships and weak tables.

TIP: In SQL Developer the button  allows to have multiple tables open this will facilitate your work. I suggest to pin at least the three main tables as in the example below. If you don't pin, you can have only the selected table on the top menu.



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	LANGUAGE	VARCHAR2 (20 BYTE)	No	(null)	1	(null)
2	WRITING	VARCHAR2 (20 BYTE)	Yes	(null)	2	(null)
3	PAINTING	VARCHAR2 (20 BYTE)	Yes	(null)	3	(null)
4	SOCIALIZING	VARCHAR2 (20 BYTE)	Yes	(null)	4	(null)
5	SPECIE	NUMBER (5,0)	Yes	(null)	5	(null)
6	COMES_FROM	NUMBER (5,0)	Yes	(null)	6	(null)
7	CONTROL	NUMBER (5,0)	Yes	(null)	7	(null)

## 4 Changing the database

In this section we are going to make all the changes necessary to modify our database.

### 4.1 Right names from PK and FK


When the data is generated, the constraints of the primary keys or foreign keys appear with a strange name like SYS\_C00671628 that is not self-explanatory of what it represents. That's why we will assume the following name convention:

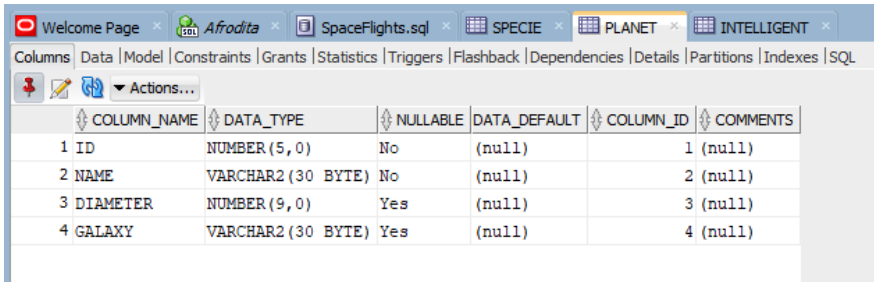
- The primary keys will be named as its table followed by \_PK. I.e., for the table INTELLIGENT the primary key will be INTELLIGENT\_PK.
- The foreign key is associated with a table and an attribute. We will name with the reference table "\_" reference column "\_FK". I.e., for the table SPECIE we have a foreign



key with the table PLANET related to its primary-key and the relations shows that the specie lives in (we will select a very brief description like LIVES). So we will can call it. PLANET\_LIVES\_FK.

- For the Check constraints we will use the name of the table "\_" some description of the constraint and "\_CHK" i.e., if the SPECIE table has a constraint related to the TYPE, we could call it: SPECIE\_TYPE\_CHK.
- For the Unique constraint we will use the same structure finished with UK "<table>\_<description>\_UK". i.e. If we have an attribute SPECIFIC in the table INTELLIGENT we can call the constraint INTELLIGENT\_SPECIFIC\_UK.

To modify any of these constraints we will go to the Table Window→Data→and click in the icon edit 



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(5,0)	No	(null)	1 (null)	
2	NAME	VARCHAR2(30 BYTE)	No	(null)	2 (null)	
3	DIAMETER	NUMBER(9,0)	Yes	(null)	3 (null)	
4	GALAXY	VARCHAR2(30 BYTE)	Yes	(null)	4 (null)	

Then in the edit windows select Constraints and the modify the name selecting it. In the example of the planet we will change SYS\_C00671625 by PLANET\_PK.

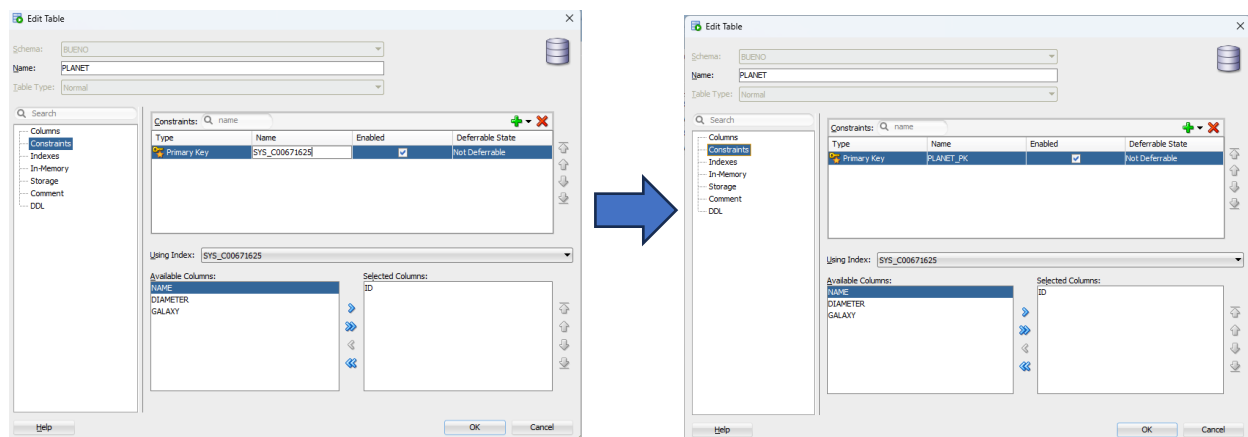
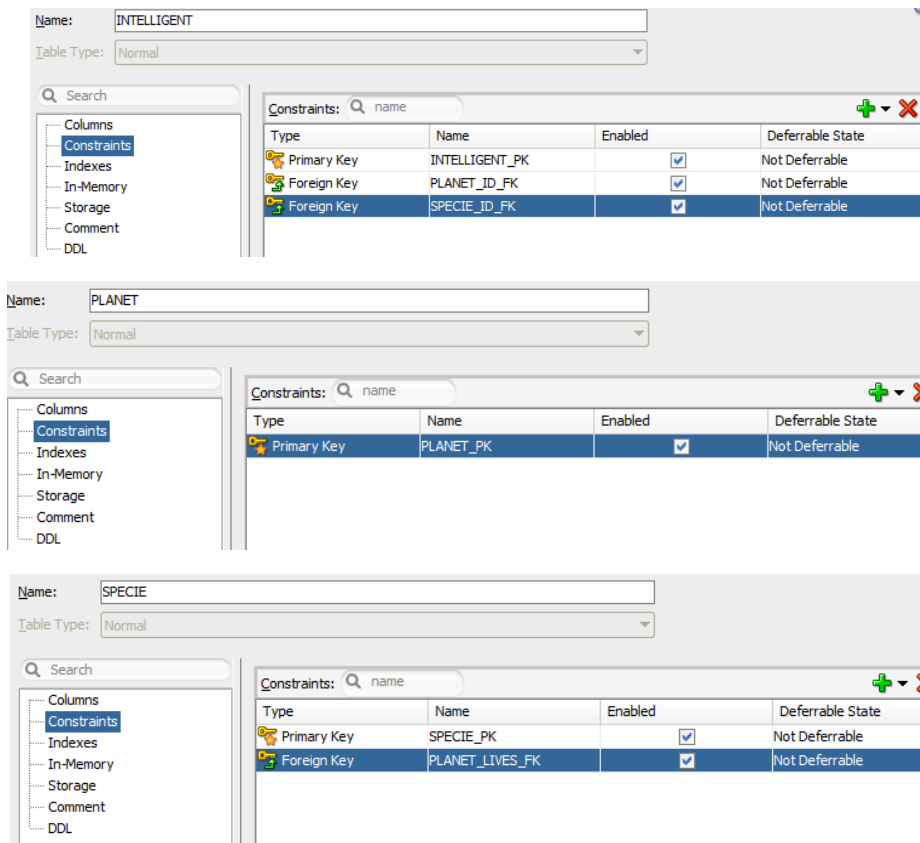


Fig. 13. Modifying the name of the constraints.

**Task:** Change all the primary and foreign existing keys to this standard in all the tables and use this convention in the new ones.

When finish you should have something like this in the three tables:




## 4.2 Change types

We have seen in the description that the TYPE of a SPECIE must be NOMAD or SEDENTARY but when we check the real data, we see that the TYPE has been defined as a number.

The screenshot shows a database tool window with tabs for 'Welcome Page', 'Afrodita', 'SpaceFlights.sql', 'SPECIE', 'PLANET', and 'INTELLIGENT'. The 'SPECIE' tab is active, showing a table structure with columns: COLUMN\_NAME, DATA\_TYPE, NULLABLE, DATA\_DEFAULT, COLUMN\_ID, and COMMENTS.

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 ID	NUMBER(5,0)	No	(null)	1	(null)
2 AVERAGE_HEIGHT	NUMBER(4,2)	Yes	(null)	2	(null)
3 AVERAGE_WEIGHT	NUMBER(6,2)	Yes	(null)	3	(null)
4 TYPE	NUMBER(1,0)	Yes	(null)	4	(null)
5 LIVES_IN	NUMBER(5,0)	Yes	(null)	5	(null)

We need to change it to be VARCHAR2 so that we can put the values NOMAD or SEDENTARY. The size of the type could be 10 (because is the length of the longest word +1 : SEDENTARY).

To change it, in the previous windows we select Edit  and then modify TYPE to appear like in Fig. 14:

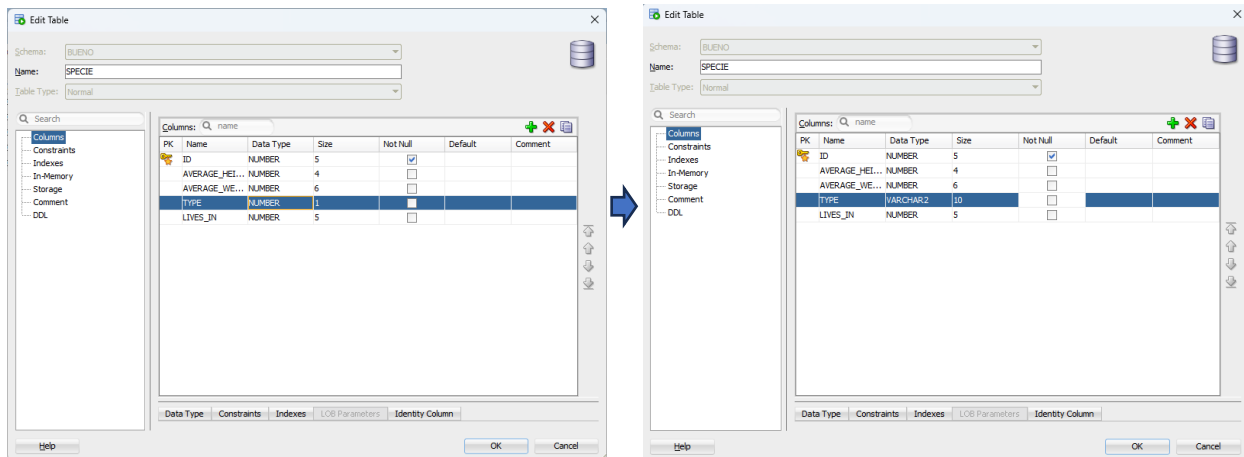


Fig. 14. Changing column values

### 4.3 Add Restrictions (Check\_constraints)

Now we want to add a restriction about the possible value of the TYPE that must be SEDENTARY or NOMAD, but the problem is that now this field is null for all the species.

	ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	LIVES_IN
1	10001	2, 3	120, 4 (null)	22222	22222
2	20002	1, 3	120, 4 (null)	22222	22222
3	30003	1, 3	60, 4 (null)	22222	22222
4	40004	2, 3	120, 4 (null)	33333	33333
5	50005	0, 3	10, 4 (null)	44444	44444
6	60006	4, 3	160, 4 (null)	77777	77777
7	70007	2, 9	90, 48 (null)	88888	88888
8	80008	2, 6	100, 4 (null)	44444	44444
9	90009	2, 5	70, 4 (null)	77777	77777

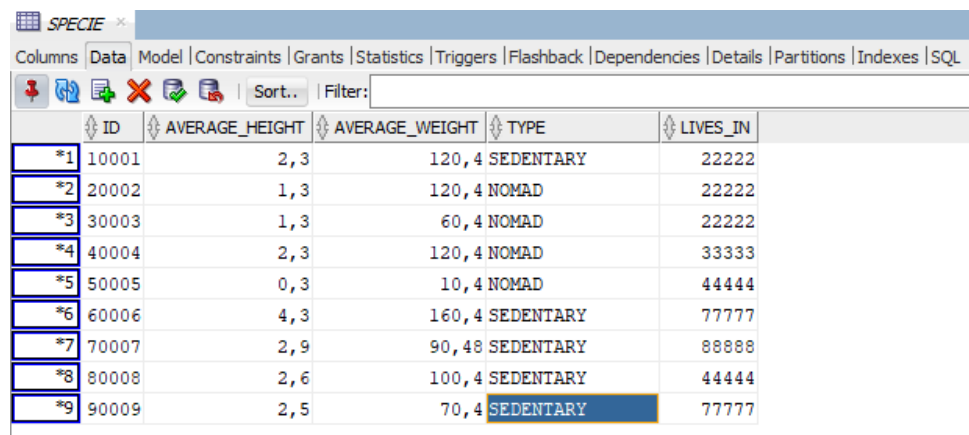
IMPORTANT: If we try to add a constraint and the data that already exists doesn't comply with the restrictions we will receive an error.

**Task:** So let's fix initially the data. Which is the type for the current species? Please try to do it by yourself analyzing the data of the tables SPECIE and INTELLIGENT. There you can see the attributes LIVES\_IN where you can find the ID of the planet. If the specie is not intelligent, they don't know how to travel in the space so they are SEDENTARY. If the specie is intelligent, you must check if the planet where it LIVES\_IN is the same that where the intelligent specie comes from. Can you fill the 9 species? Try it and only then see the solution in the next page:



It is easy to check. We have 9 species, 80008 and 90009 are not INTELLIGENT → SEDENTARY.

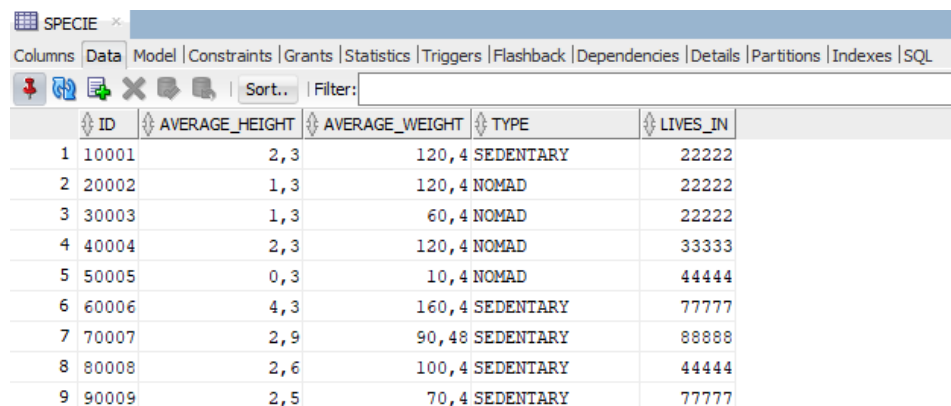
From the INTELLIGENT we can see that the planet where they live is different from the planet where they come from for: 20002,30003,40004,50005 → They are NOMAD. The rest are INTELLIGENT but they didn't leave their planet: 10001,60006,70007 → SEDENTARY.

The result table for SPECIE is:




	ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	LIVES_IN
*1	10001	2,3	120,4	SEDENTARY	22222
*2	20002	1,3	120,4	NOMAD	22222
*3	30003	1,3	60,4	NOMAD	22222
*4	40004	2,3	120,4	NOMAD	33333
*5	50005	0,3	10,4	NOMAD	44444
*6	60006	4,3	160,4	SEDENTARY	77777
*7	70007	2,9	90,48	SEDENTARY	88888
*8	80008	2,6	100,4	SEDENTARY	44444
*9	90009	2,5	70,4	SEDENTARY	77777

You can see that when you modify appears some \* and a blue box. This means that the data has only been modified in local but not in the database. You must COMMIT to fix the data with F11 (  ). NOTE: In case you want to go back to the previous state you can ROLLBACK F12(  ). After F11 the table will appear like:



	ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	LIVES_IN
1	10001	2,3	120,4	SEDENTARY	22222
2	20002	1,3	120,4	NOMAD	22222
3	30003	1,3	60,4	NOMAD	22222
4	40004	2,3	120,4	NOMAD	33333
5	50005	0,3	10,4	NOMAD	44444
6	60006	4,3	160,4	SEDENTARY	77777
7	70007	2,9	90,48	SEDENTARY	88888
8	80008	2,6	100,4	SEDENTARY	44444
9	90009	2,5	70,4	SEDENTARY	77777

Fig. 15. SPECIE table after adding TYPE values

Now we can add the constraint. We add a restriction CHECK\_CONSTRAINTS in the table SPECIES. To do that we Select our Table→Edit→Constraints→ Alt+1(  )→New Check Constraint

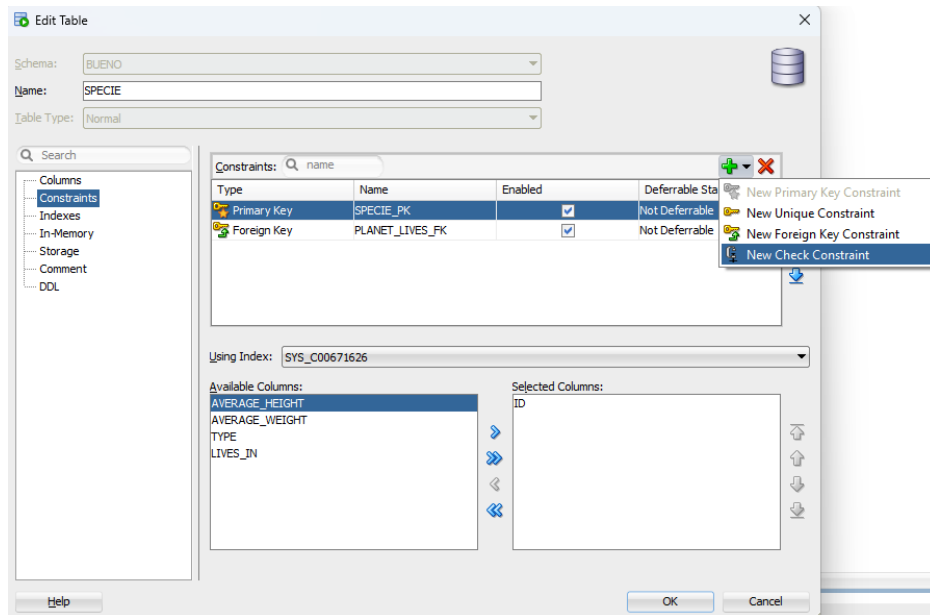
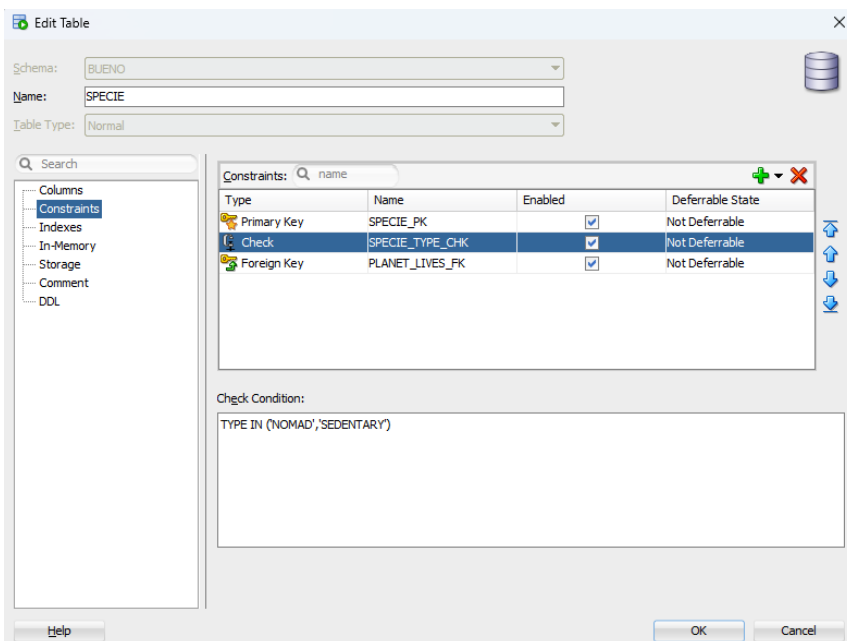


Fig. 16. Creating a new Check Constraint

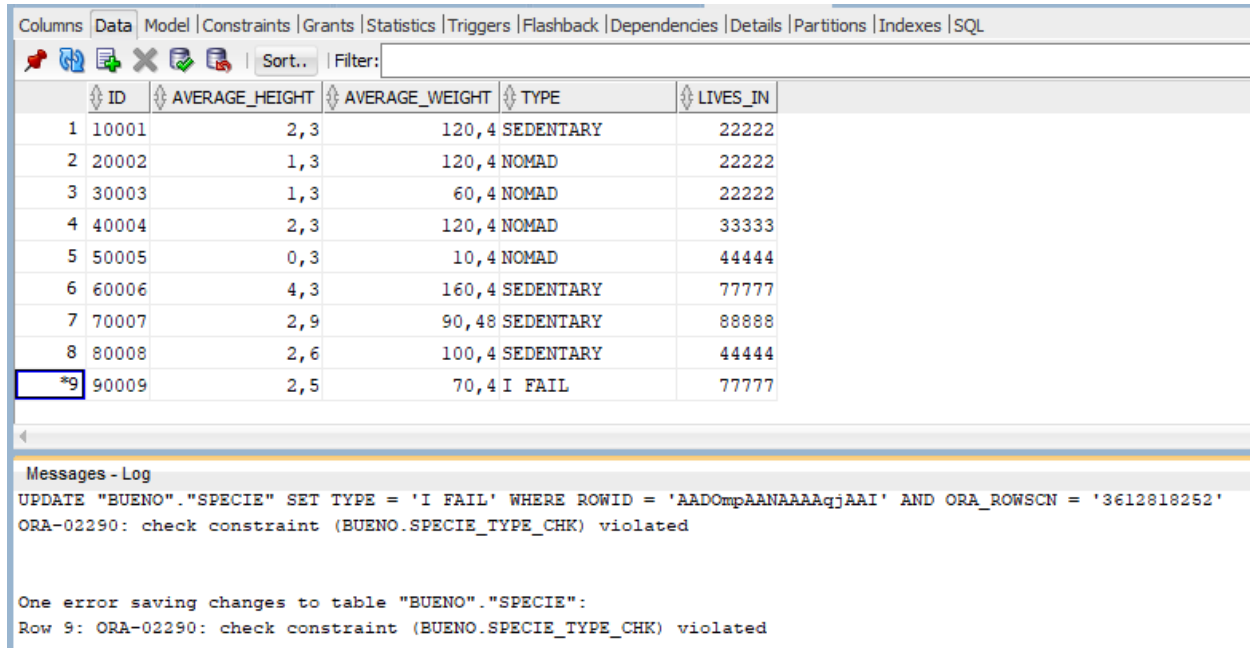
We put the name SPECIE\_TYPE\_CHK and in the Check Condition we must write the code in SQL.

```
// Choose one. These are two possibilities to do the same
// WARNING!!! DON'T COPY THE QUOTES ' FROM THE PDF THEY WILL GIVE TO PROBLEMS.
TYPE='NOMAD' or TYPE ='SEDENTARY'
TYPE IN ('NOMAD','SEDENTARY')
```



And finally, click ok.

Now we can check that if we try to add a new value or modify one of the exiting ones with something different to NOMAD or SEDENTARY, we will get an error message when we try to COMMIT. I.e., if I change the type for the last register to "I FAIL", when I try to commit with F11. I get the following message:



The screenshot shows the Oracle SQL Developer interface. The top pane displays a table with the following data:

ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	LIVES_IN
1 10001	2,3	120,4	SEDENTARY	22222
2 20002	1,3	120,4	NOMAD	22222
3 30003	1,3	60,4	NOMAD	22222
4 40004	2,3	120,4	NOMAD	33333
5 50005	0,3	10,4	NOMAD	44444
6 60006	4,3	160,4	SEDENTARY	77777
7 70007	2,9	90,48	SEDENTARY	88888
8 80008	2,6	100,4	SEDENTARY	44444
*9 90009	2,5	70,4	I FAIL	77777

The bottom pane shows the Messages - Log with the following text:



```
UPDATE "BUENO"."SPECIE" SET TYPE = 'I FAIL' WHERE ROWID = 'AAD0mpAANAAAAqjAAI' AND ORA_ROWSCN = '3612818252'
ORA-02290: check constraint (BUENO.SPECIE_TYPE_CHK) violated

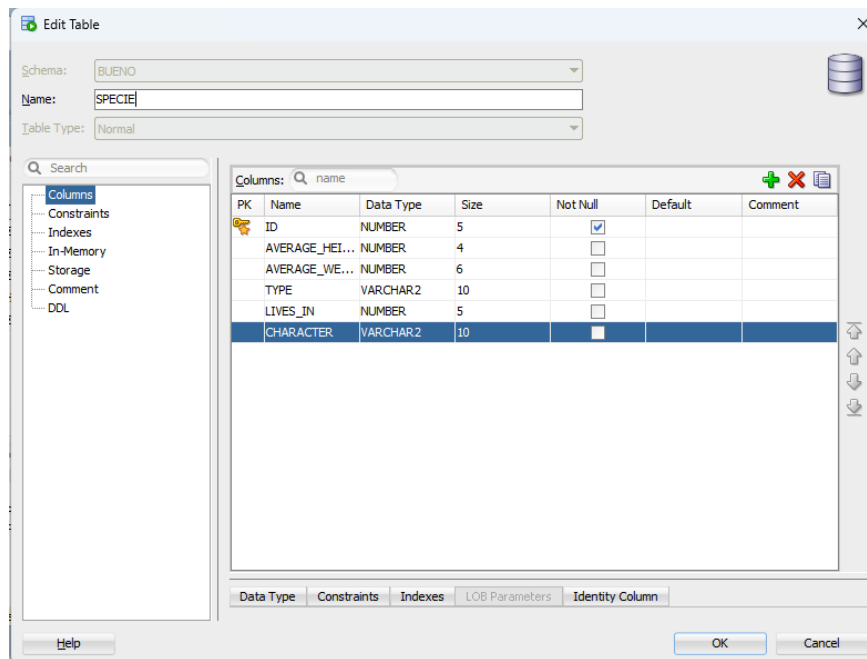
One error saving changes to table "BUENO"."SPECIE":
Row 9: ORA-02290: check constraint (BUENO.SPECIE_TYPE_CHK) violated
```

Click the ROLLBACK F12() to go to the last stable state.

As we used understandable names, we know that we have violated the restriction SPECIE\_TYPE\_CHK. With a generic value SYS\_0038303 we wouldn't know exactly what the error was.

#### 4.4 Add new attributes

1. **In SPECIE table does not appear the CHARACTER.** When we add it, we must also add the constraints of the problem description.
  - a. To add it with the SPECIE table selected (Columns→ Edit  → Columns→ Alt+1() to add a new row: Add a column CHARACTER with varchar2 10.



- b. Then we must add the constraints. If type is NOMAD the CHARACTER must be WARRIOR or PACIFIC, if it is sedentary there is no restriction.

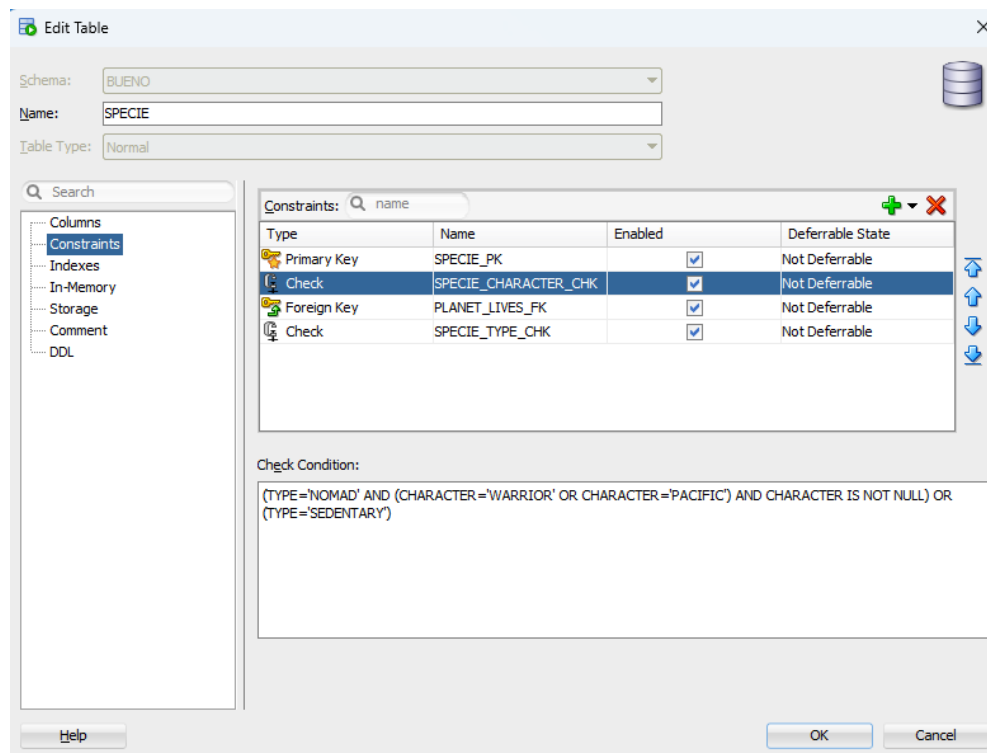
```
(TYPE='NOMAD' AND (CHARACTER='WARRIOR' OR CHARACTER='PACIFIC') AND CHARACTER IS NOT NULL) OR
(TYPE='SEDENTARY')
```

But again, before adding the constraint we should set the registers with VALID Values. As we may not have all the information, we could leave all the SEDENTARY AS NULL. And as we can see in Fig. 15, the INTELLIGENT species as 4004 and 5005 are NOMAD and are the only one intelligent specie in their planets we could say they are WARRIOR. The other NOMADS, 2002 AND 3003, lives in an planet controlled by other SPECIE (10001) so we can say they are PACIFIC. With this idea (you can choose different values) our SPECIE tables is like this:

ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	LIVES_IN	CHARACTER
1 10001	2,3	120,4	SEDENTARY	22222	(null)
2 20002	1,3	120,4	NOMAD	22222	PACIFIC
3 30003	1,3	60,4	NOMAD	22222	PACIFIC
4 40004	2,3	120,4	NOMAD	33333	WARRIOR
5 50005	0,3	10,4	NOMAD	44444	WARRIOR
6 60006	4,3	160,4	SEDENTARY	77777	(null)
7 70007	2,9	90,48	SEDENTARY	88888	(null)
8 80008	2,6	100,4	SEDENTARY	44444	(null)
9 90009	2,5	70,4	SEDENTARY	77777	(null)

Fig. 17. State of table SPECIE after adding the CHARACTER

Now we can add the Check Constraint



#### 4.5 Solve the Many to Many Relations (N-M)

To try to understand better the relationship between the entities I have reordered the position of the entities and labeled the original diagram obtained in Fig. 10.

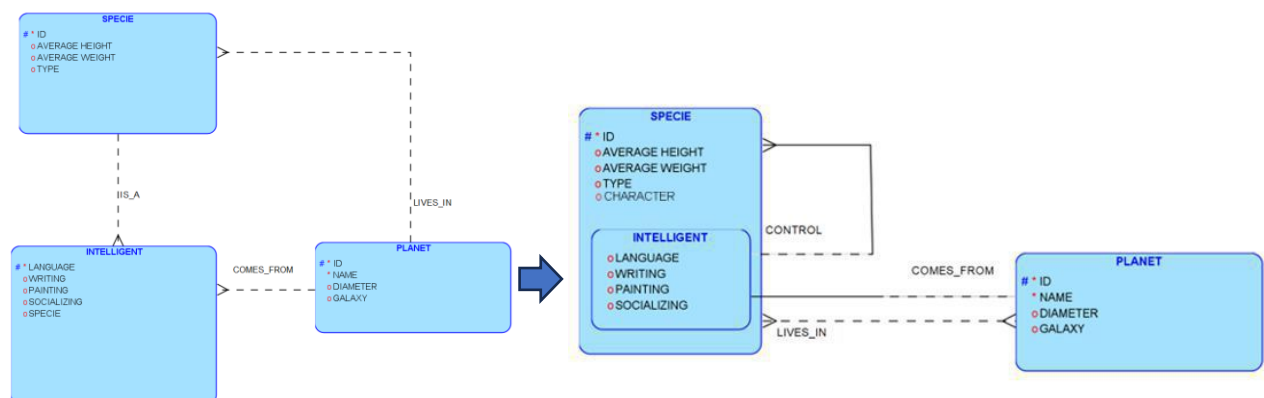


Fig. 18. Update of Fig. 12 with labels to compare the 17.a) original relations with the 17.b) desired ones.

Now we are going to focus on the LIVES\_IN relation. In the desired diagram it is a N-M Relation that means that a SPECIE can live in many planets and that in a PLANET can live many species. With our current implementation we can only document species in one planet because it lives\_in is an attribute of the SPECIE that can't be multivalued.



With our goal clear, the only way to convert 1-N to a N-M is creating a new table LIVES\_IN that will depend on PLANET and SPECIES, as we don't need special attributes it must have as its primary key a composition of the primary keys of PLANET and SPECIES. To reflect that in the relational model we must make some steps.



- 1) Create the new table LIVES\_IN
- 2) Add 2 attributes that represent each of the primary keys of the main tables, and
- 3) Make them also as foreign keys, to confirm that the values in LIVES\_IN will be only valid values.
- 4) Correct the values in the tables to keep the current values and delete columns not needed in SPECIE.

To create the table we can use two ways with SQL or with the visual interface. You can choose the one that you want, but you should know both.

#### 4.5.1 Create the table LIVES\_IN using SQL

To create the table using SQL, at the same time we could finish the previous steps 1), 2) and 3). The code to write in the worksheet is this:

```
CREATE TABLE LIVES_IN (  
    SPECIE_ID NUMBER(5),  
    PLANET_ID NUMBER(5),  
    CONSTRAINT LIVES_IN_PK PRIMARY KEY(SPECIE_ID,PLANET_ID),  
    CONSTRAINT SPECIE_ID_FK FOREIGN KEY(SPECIE_ID) REFERENCES SPECIE(ID),  
    CONSTRAINT PLANET_ID_FK FOREIGN KEY(PLANET_ID) REFERENCES PLANET(ID)  
);
```

We code it in SQL Developer and run the script with any of the two buttons  . If everything is correct we will see the message "Table LIVES\_IN created". As you can see in

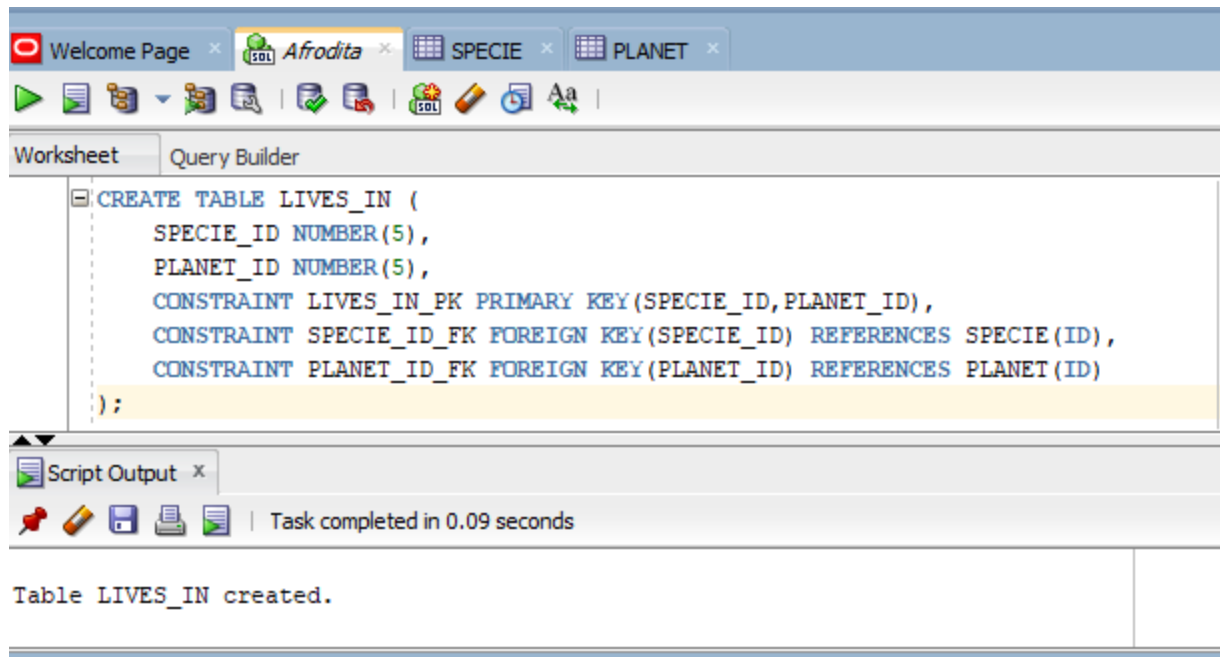
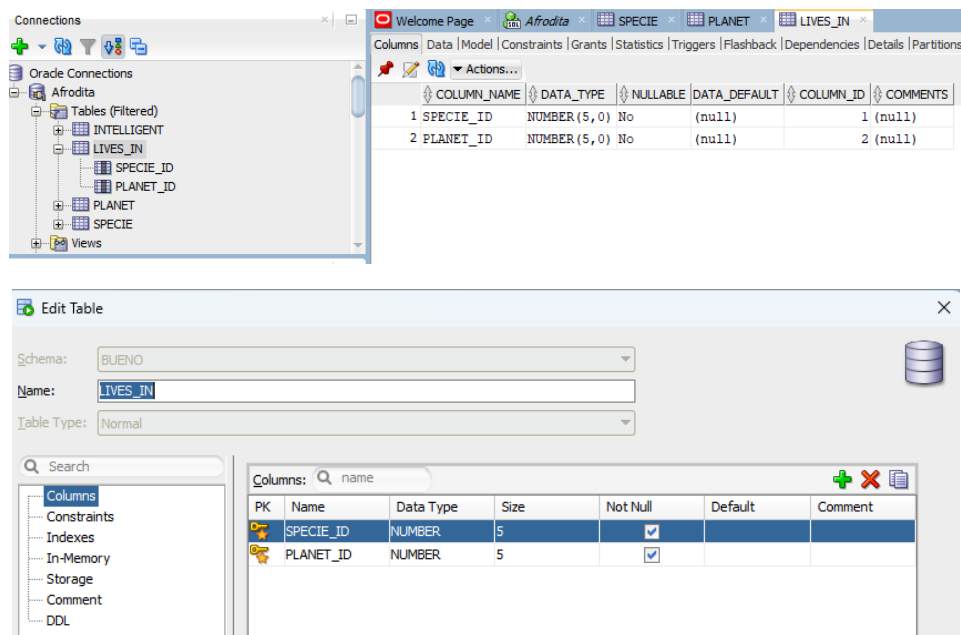


Fig. 19. Result of the creation of the table LIVES\_IN

After that we can check that everything is ok reviewing the information of the new table as we can see in the following images:



Schema: BUENO  
Name: LIVES\_IN  
Table Type: Normal

Search

- Columns
- Constraints**
- Indexes
- In-Memory
- Storage
- Comment
- DDL

Constraints: Search name

Type	Name	Enabled	Deferrable State
Primary Key	LIVES_IN_PK	<input checked="" type="checkbox"/>	Not Deferrable
Foreign Key	PLANET_ID_FK	<input checked="" type="checkbox"/>	Not Deferrable
Foreign Key	SPECIE_ID_FK	<input checked="" type="checkbox"/>	Not Deferrable

Using Index: LIVES\_IN\_PK

Available Columns:

Selected Columns:  
SPECIE\_ID  
PLANET\_ID

Schema: BUENO  
Name: LIVES\_IN  
Table Type: Normal

Search

- Columns
- Constraints**
- Indexes
- In-Memory
- Storage
- Comment
- DDL

Constraints: Search name

Type	Name	Enabled	Deferrable State
Primary Key	LIVES_IN_PK	<input checked="" type="checkbox"/>	Not Deferrable
Foreign Key	PLANET_ID_FK	<input checked="" type="checkbox"/>	Not Deferrable
Foreign Key	SPECIE_ID_FK	<input checked="" type="checkbox"/>	Not Deferrable

Referenced Constraint

Schema: BUENO  
Table: PLANET  
Constraint: PLANET\_PK  
On Delete: No Action

Associations:

Local Column	Referenced Column
PLANET_ID	ID

Schema: BUENO  
Name: LIVES\_IN  
Table Type: Normal

Search

- Columns
- Constraints**
- Indexes
- In-Memory
- Storage
- Comment
- DDL

Constraints: Search name

Type	Name	Enabled	Deferrable State
Primary Key	LIVES_IN_PK	<input checked="" type="checkbox"/>	Not Deferrable
Foreign Key	PLANET_ID_FK	<input checked="" type="checkbox"/>	Not Deferrable
Foreign Key	SPECIE_ID_FK	<input checked="" type="checkbox"/>	Not Deferrable

Referenced Constraint

Schema: BUENO  
Table: SPECIE  
Constraint: SPECIE\_PK  
On Delete: No Action

Associations:

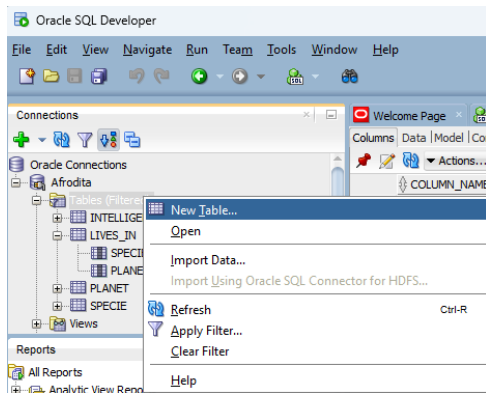
Local Column	Referenced Column
SPECIE_ID	ID

Fig. 20. Different images to check that the table LIVES\_IN has been created correctly

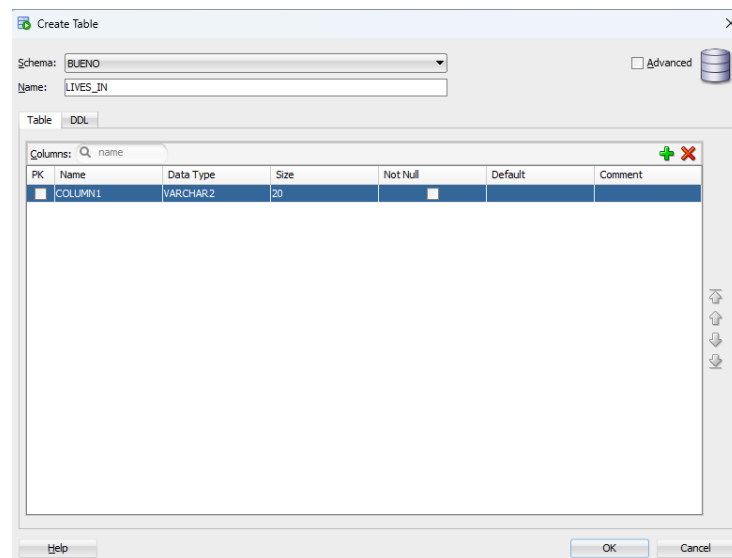
#### 4.5.2 Creating the table LIVES\_IN with the visual interface



NOTE: This is an alternative to 4.5.1 if you created the table there this steps are not necessary but if you want to learn, you can delete the LIVES\_IN created before.

To create a new table with the visual interface we click with the right button on Tables→New Table



In the new window we put the name to the table and click ok. We will change the attributes later.



Now we select the table LIVES\_IN→Columns → Edit . In this window we need to add the attributes and the constraints. As a HINT to add the attributes the simple way is to add them using the copy button , because we want to create the attributes PLANET\_ID and SPECIE\_ID that must have the same format than the ID of PLANET table and ID of SPECIE table respectively. I.e., to create the PLANET\_ID we use the copy button, and in the new window that appear "Copy columns to Table LIVES\_IN", we select the table PLANET and the column "ID NUMBERS(5,0)" as you can see in Fig. 21.

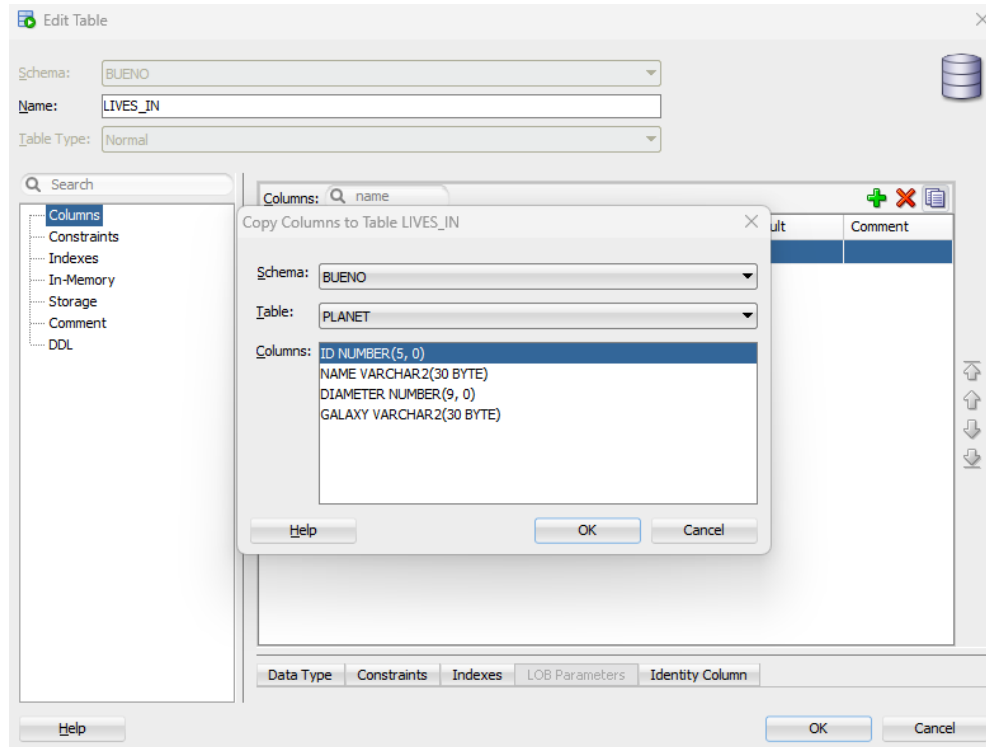
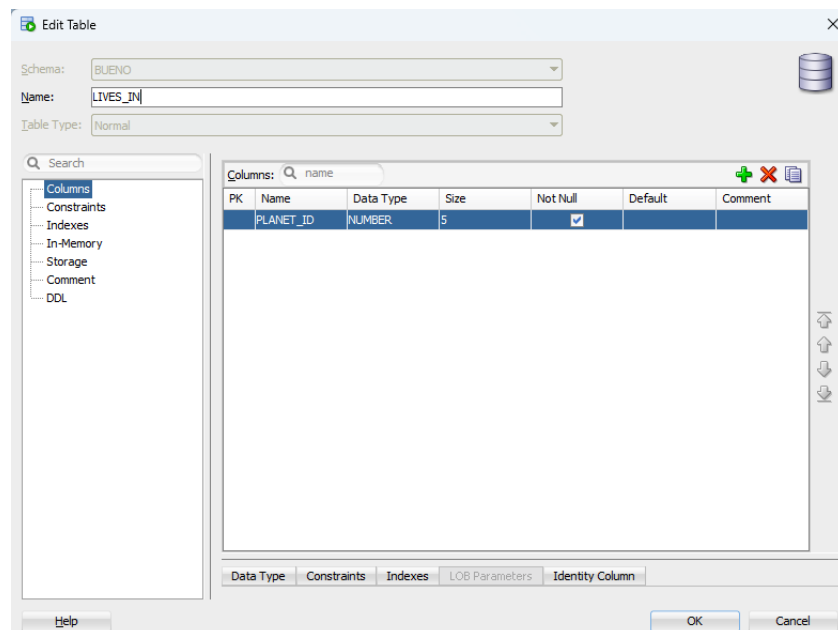


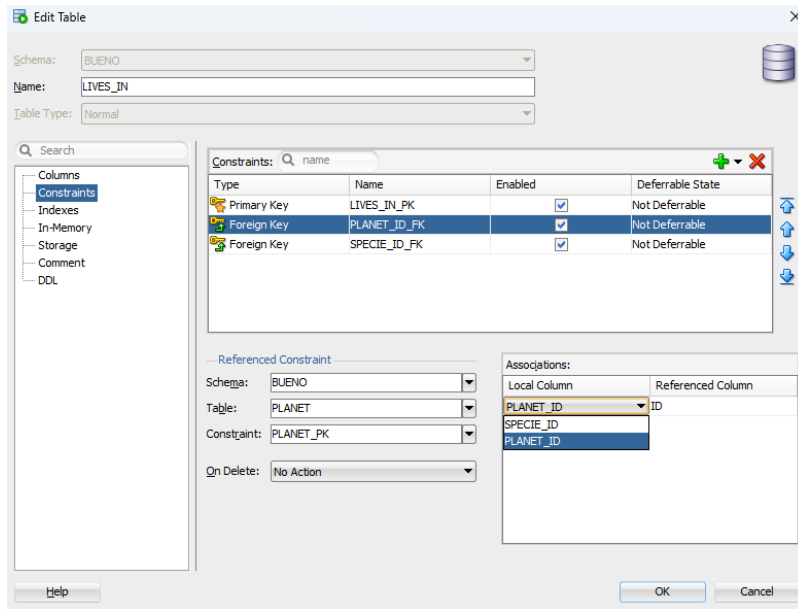
Fig. 21. Copying format values from other tables.

After clicking ok, we change the name of the column to PLANET\_ID and then we can delete the default attribute. Now it should look like this:



**Task:** Continue adding the attributes and Constraints to fit with the Fig. 20.

NOTE: Pay attention when creating the constraints to put the right values in the local column and reference column. I.e., for planet as Local Column you must select PLANET\_ID.



#### 4.5.3 Copying the data from SPECIE to LIVES\_IN

To finish the steps of 4.5 we need to fill our new table LIVES\_IN with the data that was in SPECIE. We need to pass from this specie table:

ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	LIVES_IN	CHARACTER
1 10001	2,3	120,4	SEDENTARY	22222 (null)	
2 20002	1,3	120,4	NOMAD	22222 PACIFIC	
3 30003	1,3	60,4	NOMAD	22222 PACIFIC	
4 40004	2,3	120,4	NOMAD	33333 WARRIOR	
5 50005	0,3	10,4	NOMAD	44444 WARRIOR	
6 60006	4,3	160,4	SEDENTARY	77777 (null)	
7 70007	2,9	90,48	SEDENTARY	88888 (null)	
8 80008	2,6	100,4	SEDENTARY	44444 (null)	
9 90009	2,5	70,4	SEDENTARY	77777 (null)	



Fig. 22. Table SPECIE a) Before moving data to LIVES\_IN b) After copying data to LIVES\_IN

ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	LIVES_IN	CHARACTER
----	----------------	----------------	------	----------	-----------



Fig. 23. Table LIVES\_IN a) Empty before copying the data b) After copying the data from SPECIE

To do that we see that the information that we need to fill LIVES\_IN is the ID of the specie and the ID of the planet where lives in. We have this information as the column ID and LIVES\_IN. This means that with this sentence we can get all the data needed for LIVES\_IN:

```
SELECT ID,LIVES_IN FROM SPECIE;
```

We can see the result of this query in:

The screenshot shows the Afrodita interface with a SQL query in the Query Builder: `SELECT ID, LIVES_IN FROM SPECIE;`. The Query Result pane displays the following data:

ID	LIVES_IN
1 10001	22222
2 20002	22222
3 30003	22222
4 40004	33333
5 50005	44444
6 60006	77777
7 70007	88888
8 80008	44444
9 90009	77777

Fig. 24. Querying the SPECIE\_ID, PLANET\_ID to fill the LIVES\_IN table

Now. How do I copy the data to the table LIVES\_IN? You can try different things with the visual interface like create 9 new entries in the table LIVES\_IN and then copy and paste the result of Fig. 24 but the most elegant way will be to do it with a SQL sentence.


```
INSERT INTO LIVES_IN (SPECIE_ID,PLANET_ID) (SELECT ID,LIVES_IN FROM SPECIE);
```

This will be the result:

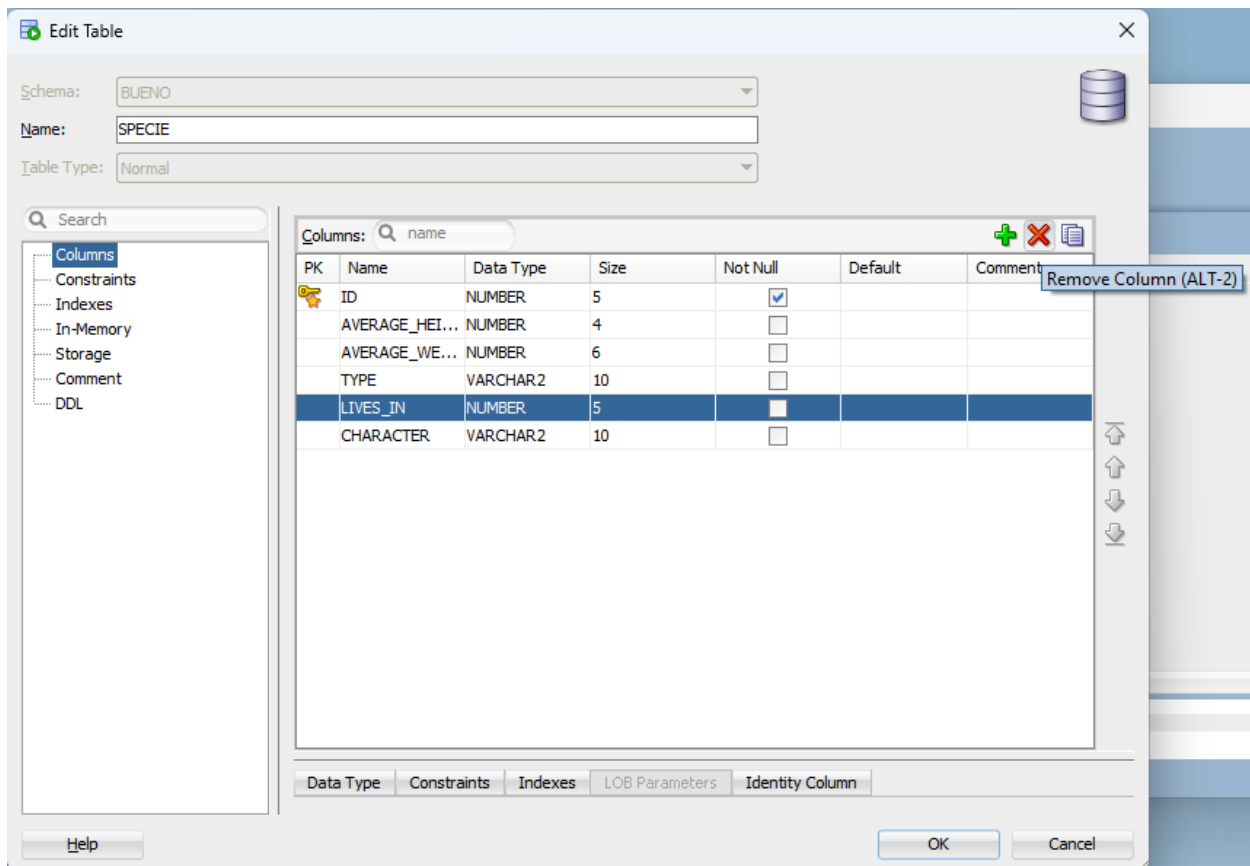
The screenshot shows the Afrodita interface with the SQL query `INSERT INTO LIVES_IN (SPECIE_ID,PLANET_ID) (SELECT ID,LIVES_IN FROM SPECIE);` in the Query Builder. The Script Output pane shows "9 rows inserted." and the Messages pane shows "Task completed in 0.129 seconds". A blue arrow points to the resulting table data:

	SPECIE_ID	PLANET_ID
1	10001	22222
2	20002	22222
3	30003	22222
4	40004	33333
5	50005	44444
6	60006	77777
7	70007	88888
8	80008	44444
9	90009	77777

#### 4.5.4 Deleting a Column

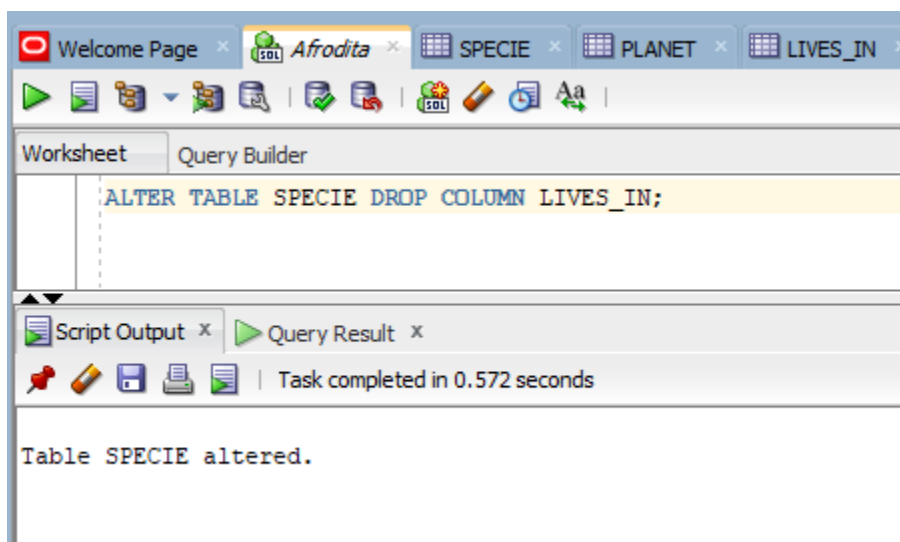
Now that we are sure that the data is in the right place, we will delete the column LIVES\_IN in the table SPECIE. You can do with the visual interface selecting the column LIVES\_IN in the table SPECIE and then Removing the column with ALT-2(

Databases. ETSI Informática



Or doing it using SQL with:

```
ALTER TABLE SPECIE DROP COLUMN LIVES_IN;
```



In both cases the result will be that the column LIVES\_IN disappeared from the table:



ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	CHARAC...
1 10001	2, 3	120, 4	SEDENTARY	(null)
2 20002	1, 3	120, 4	NOMAD	PACIFIC
3 30003	1, 3	60, 4	NOMAD	PACIFIC
4 40004	2, 3	120, 4	NOMAD	WARRIOR
5 50005	0, 3	10, 4	NOMAD	WARRIOR
6 60006	4, 3	160, 4	SEDENTARY	(null)
7 70007	2, 9	90, 48	SEDENTARY	(null)
8 80008	2, 6	100, 4	SEDENTARY	(null)
9 90009	2, 5	70, 4	SEDENTARY	(null)

This is a good moment to review how is evolving our database. We can do it directly from Data Modeler as explained in 3.3 or doing it inside SQLDeveloper as shown in 3.4 (This is the exact point of this example). We can see that now the relation LIVES\_IN are equivalent in both diagrams as we say in theory.

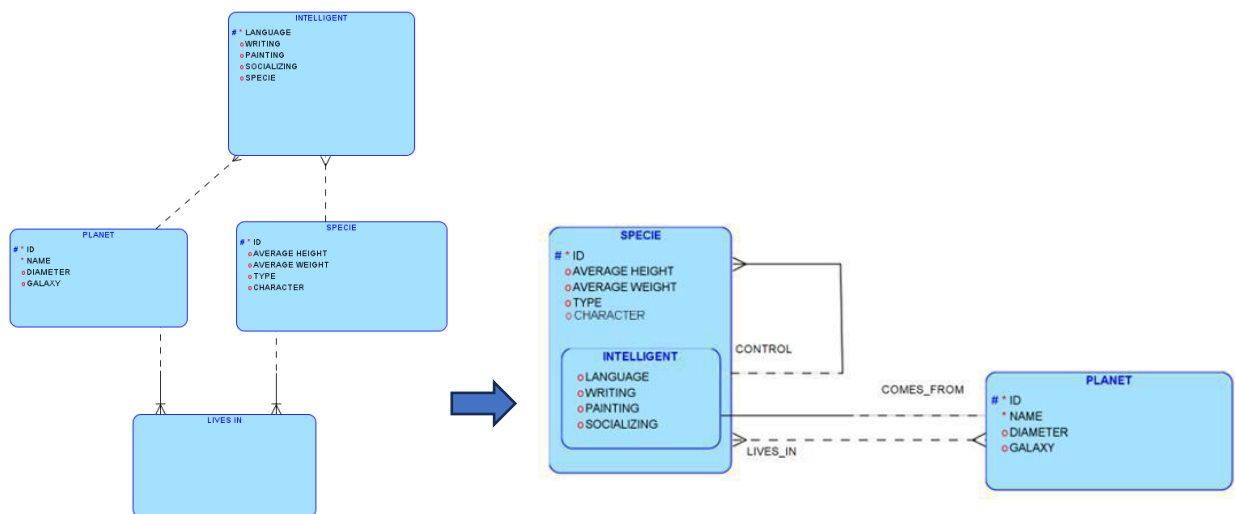


Fig. 25. Situation at the end of this section compared to the final result

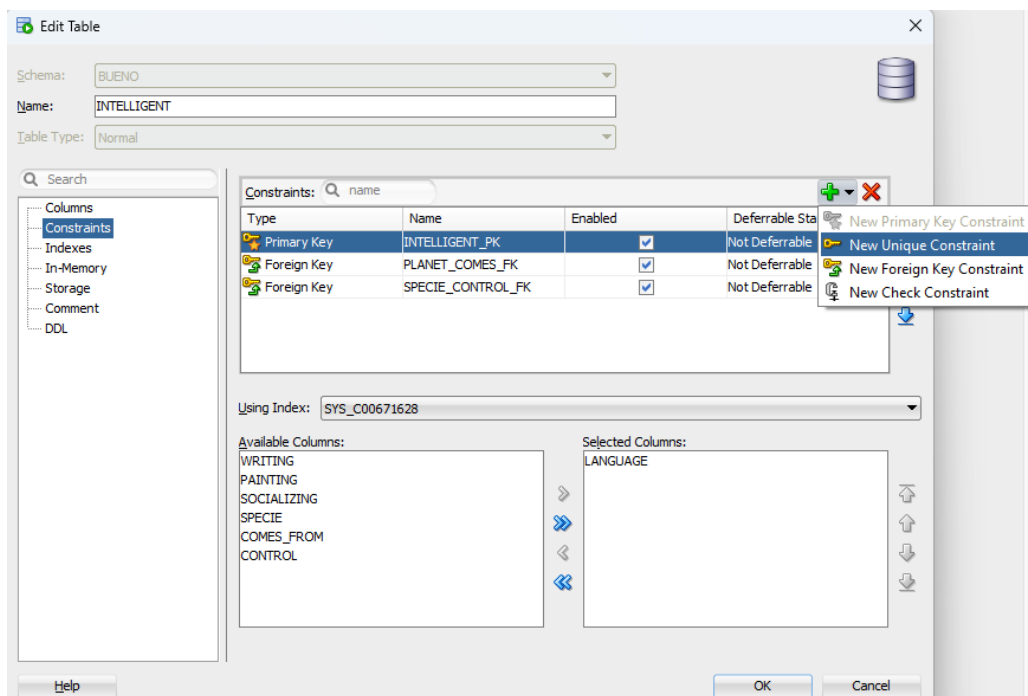
#### 4.6 Convert relation 1-N to 1-1 (UNIQUE)

In the Diagrams we find that the relation between the PLANET and the INTELLIGENT Specie corresponds to COMES\_FROM. In the current diagram the relationship between them is 1-N but in the final diagram is 1-1 because it is supposed that only one Intelligent Specie from one planet can travel to other planets.

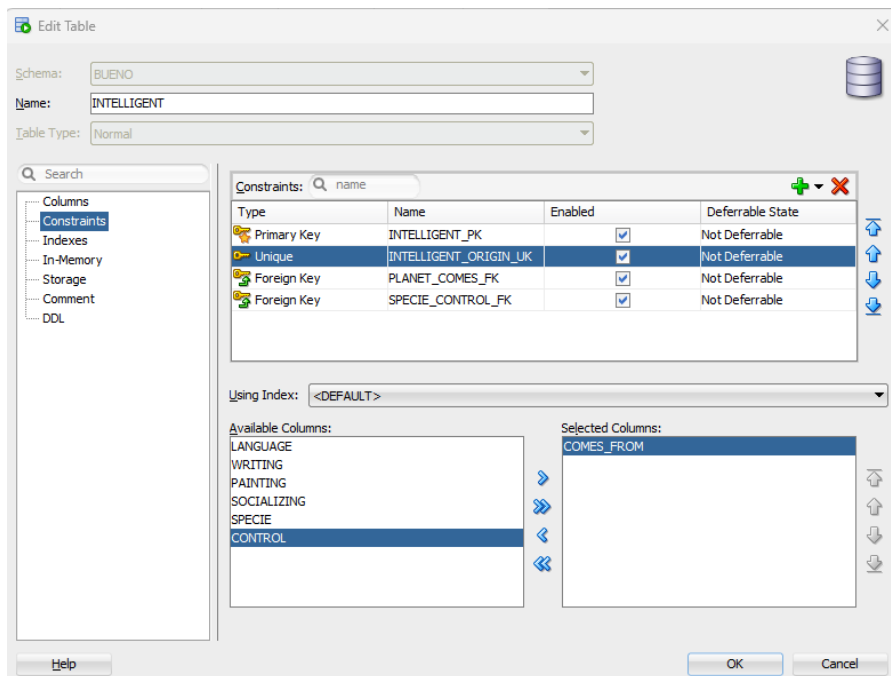
To solve this problem, we are going to add a Unique Constraints that must says that the COMES\_FROM attribute can't be repeated.

	LANGUAGE	WRITING	PAINTING	SOCIALIZING	SPECIE	COMES_FROM	CONTROL
1	Gramatino	Depurate	y	n	10001	22222	20002
2	Levuorin	Begining	y	n	20002	33333	30003
3	Saporo	Unknown	n	n	30003	44444	40004
4	Arabino	Depurate	y	n	40004	55555	50005
5	Aghaga	Depurate	y	y	50005	66666	60006
6	Espatul	Unknown	y	n	60006	77777	70007
7	Forensic	(null)	y	y	70007	88888	10001

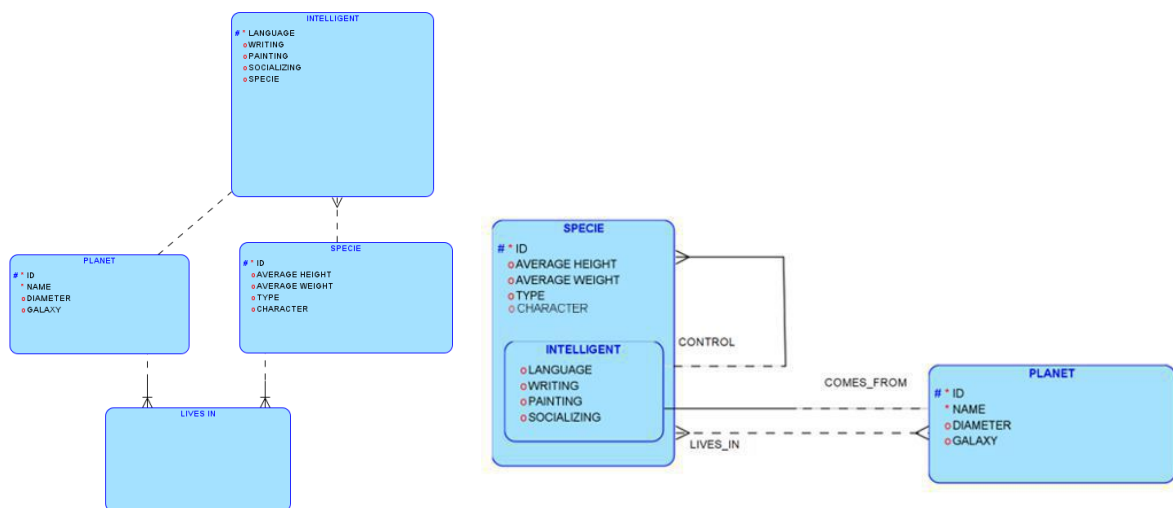
To add a Unique Constraint, goes to Table Intelligent→Columns→Edit→Constraints→Add Unique Constraint



The relationship COMES\_FROM from appears as 1 to many. The way to fix it is by putting a UNIQUE constraint. We will name it INTELLIGENT\_ORIGIN\_UK and will select the attribute COMES\_FROM to be Unique.



We are now at this point. We have solve the 1-N to 1-1 but in the final diagram the side of INTELLIGENT in the relation COMES\_FROM is solid. This means that the attribute COMES\_FROM in INTELLIGENT is mandatory. We will solve in the next section.



#### 4.7 Indicate that an attribute cannot be NULL

As we have just seen in the previous section, we must indicate that COMES\_FROM can't be NULL (to make this part of the relation mandatory). This is very easy, and the only change will be to fix is the column NOT NULL for the attribute COMES\_FROM

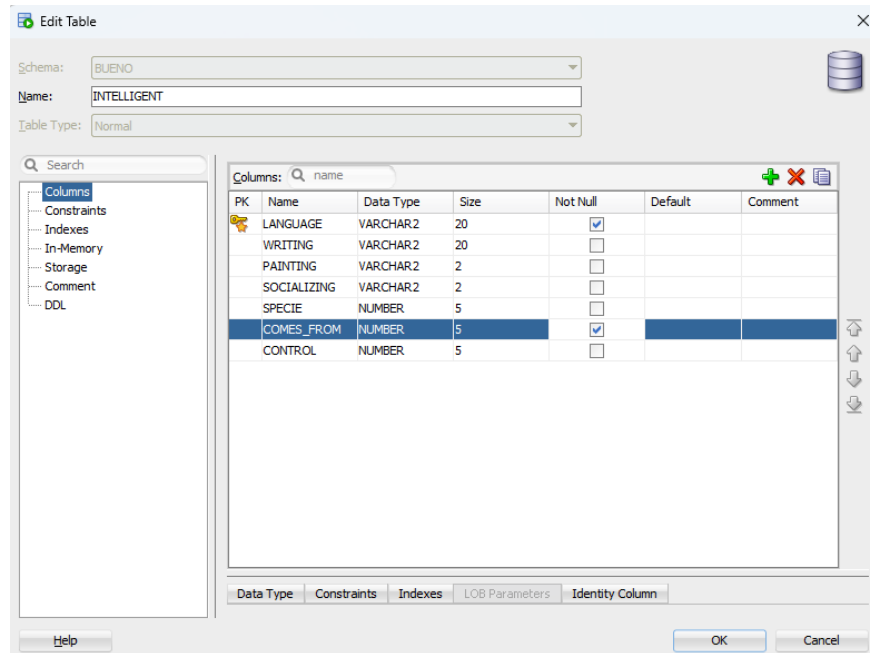


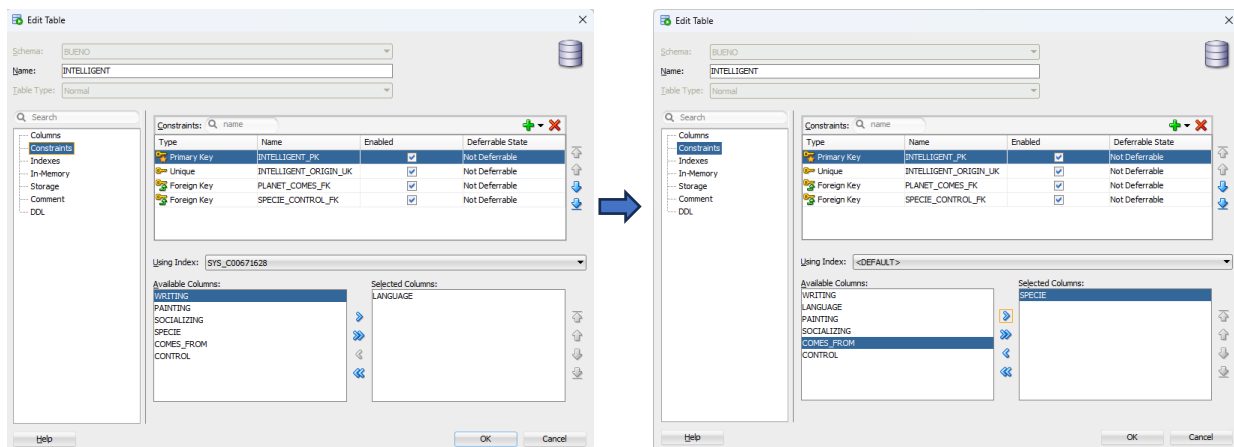
Fig. 26. Current status of INTELLIGENT table after settings COMES\_FROM as NOT NULL

#### 4.8 Solving the Sub entity relation (in INTELLIGENT)

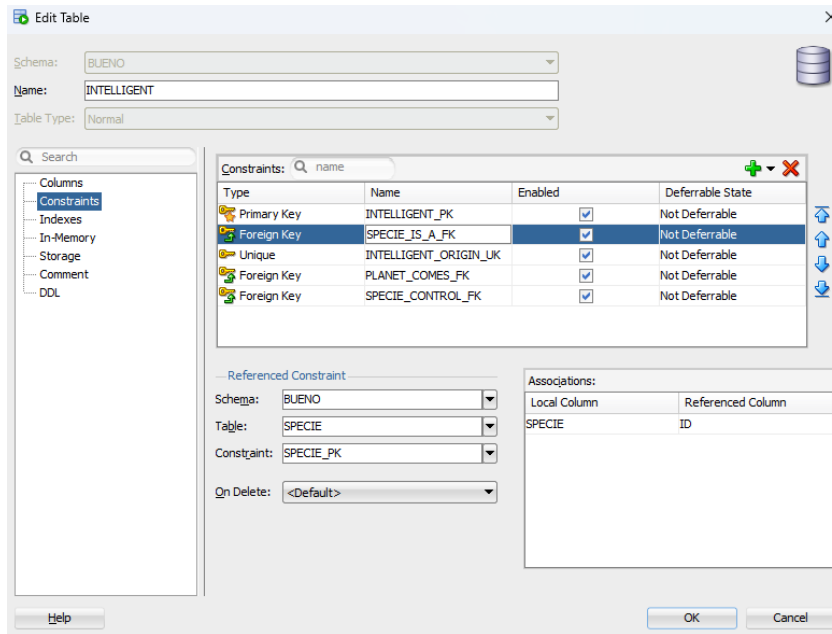
Checking desired the diagram, we can see that the INTELLIGENT table is a sub entity of SPECIE. We find in INTELLIGENT an optional attribute SPECIE that should be the PRIMARY KEY and related as FOREIGN KEY with a relation 1-1 with the main table SPECIE.

Step by Step:

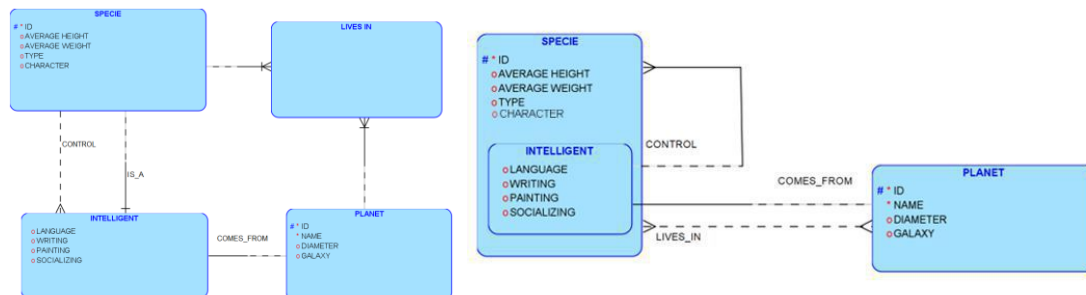
1. Change the Primary Key from LANGUAGE to SPECIE



2. Add a foreign key IS\_A with the table SPECIE that relates the INTELLIGENT.SPECIE attribute to the SPECIE.ID. It should be like this:



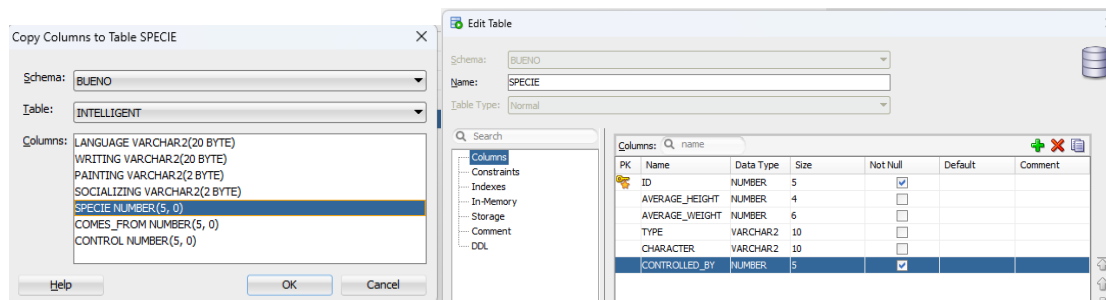
At this point we have everything nearly correct (the LIVES\_IN relation are equivalent and the same for the Sub Entity). The only problem is the CONTROL relationship that is reversed.



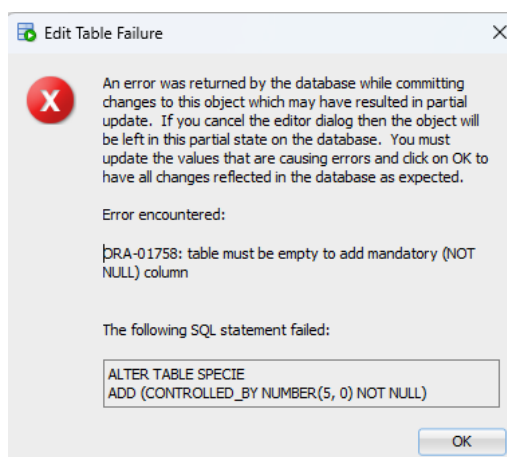
#### 4.9 Change direction in a 1-N relationship

As seen before the relation CONTROL is reversed as it is, a SPECIE can control MANY INTELLIGENT species, but it should be the contrary. It should be 1 to many from INTELLIGENT to SPECIES and is backwards with a CONTROL attribute in the INTELLIGENT table.

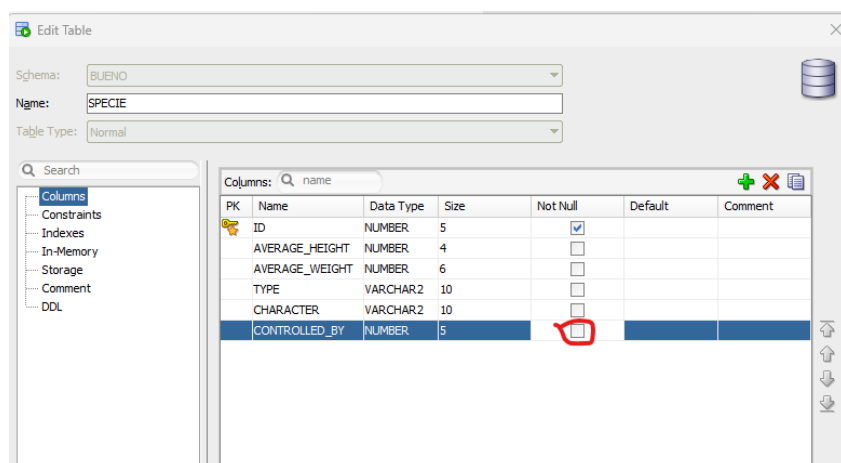
To solve this problem, we must add a column in SPECIES called CONTROLLED\_BY that will have as FOREIGN KEY the key of the entity INTELLIGENT (Go to Edit Table and add a new column with the copy button that copies the primary key of INTELLIGENT that is SPECIE).



IMPORTANT NOTE: If you want to create a column and it cannot have null values. You have to create the column first without constraints, copy the values and then add the constraint. If you don't remove it you will get an alert like this:



So you must remove the check until the data are corrected. Removing the tick in the column NOT NULL for the CONTROLLED\_BY row.



With this you can click OK to modify the table edition.

Now we need to set the data because the column CONTROLLED\_BY is empty.

ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	CHARACTER	CONTROLLED_BY
1 10001	2,3	120,4	SEDENTARY	(null)	(null)
2 20002	1,3	120,4	NOMAD	PACIFIC	(null)
3 30003	1,3	60,4	NOMAD	PACIFIC	(null)
4 40004	2,3	120,4	NOMAD	WARRIOR	(null)
5 50005	0,3	10,4	NOMAD	WARRIOR	(null)
6 60006	4,3	160,4	SEDENTARY	(null)	(null)
7 70007	2,9	90,48	SEDENTARY	(null)	(null)
8 80008	2,6	100,4	SEDENTARY	(null)	(null)
9 90009	2,5	70,4	SEDENTARY	(null)	(null)

To relate the SPECIES with the INTELLIGENT that controls we can make this query (or see it in the table INTELLIGENT).

```
SELECT CONTROL, SPECIE FROM INTELLIGENT;
```

CONTROL	SPECIE
20002	10001
30003	20002
40004	30003
50005	40004
60006	50005
70007	60006
10001	70007

Copy the values that are in the Intelligence table as CONTROLS\_BY. I.e. as 10001 control 20002 then in the row of 20002 we will put controlled by 10001.

Solve the two species that have no relationship created (80008 and 90009) by taking another species that lives on the same planet selecting the INTELLIGENT as the controller. In the LIVES\_IN we can check that 80008 lives in 44444 (where the intelligent species 50005 lives also) and 90009 lives in 77777 (where 60006 lives also). So, we can suppose that 80008 is controlled by 50005 and 90009 is controlled by 60006.

The final values will be:

	ID	AVERAGE_HEIGHT	AVERAGE_WEIGHT	TYPE	CHARACTER	CONTROLLED_BY
1	10001	2, 3	120, 4	SEDENTARY	(null)	70007
2	20002	1, 3	120, 4	NOMAD	PACIFIC	10001
3	30003	1, 3	60, 4	NOMAD	PACIFIC	20002
4	40004	2, 3	120, 4	NOMAD	WARRIOR	30003
5	50005	0, 3	10, 4	NOMAD	WARRIOR	40004
6	60006	4, 3	160, 4	SEDENTARY	(null)	50005
7	70007	2, 9	90, 48	SEDENTARY	(null)	60006
8	80008	2, 6	100, 4	SEDENTARY	(null)	50005
9	90009	2, 5	70, 4	SEDENTARY	(null)	60006

After this, it is possible to select in the species table that the CONTROLLED\_BY attribute is not null. (In Columns) and click ok.

Schema: BUENO  
Name: SPECIE  
Table Type: Normal

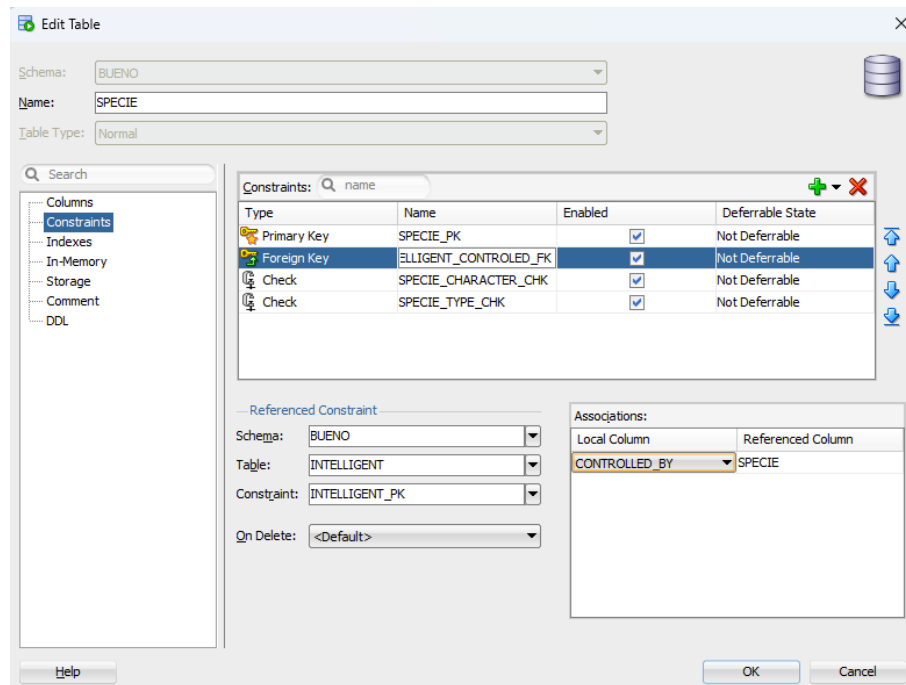
Columns:

PK	Name	Data Type	Size	Not Null	Default	Comment
	ID	NUMBER	5	<input checked="" type="checkbox"/>		
	AVERAGE_HEIGHT	NUMBER	4	<input type="checkbox"/>		
	AVERAGE_WEIGHT	NUMBER	6	<input type="checkbox"/>		
	TYPE	VARCHAR2	10	<input type="checkbox"/>		
	CHARACTER	VARCHAR2	10	<input type="checkbox"/>		
	CONTROLLED_BY	NUMBER	5	<input checked="" type="checkbox"/>		

Help OK Cancel

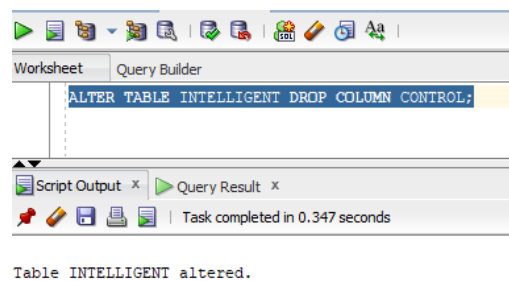
Now we need to add the constraints that CONTROLLED\_BY is FOREIGN KEY in SPECIE as mandatory.





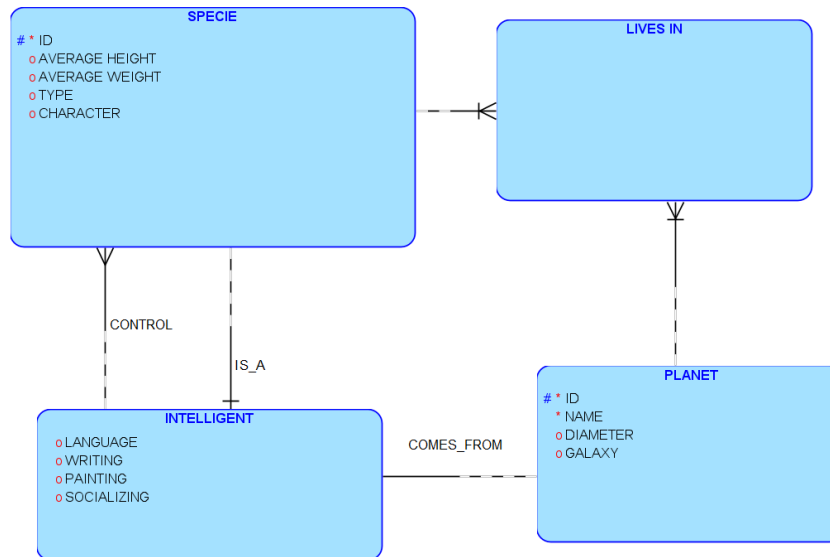
Finally delete the CONTROL in the table INTELLIGENT.

```
ALTER TABLE INTELLIGENT DROP COLUMN CONTROL;
```



#### 4.10 Final Result

The final result is:



#### 4.11 Export from SQL Developer

If we want to export the database SQL Developer Tools→Database Export

