Table of contents

Vitalii Naumov

FY46IN

Data Models and DB

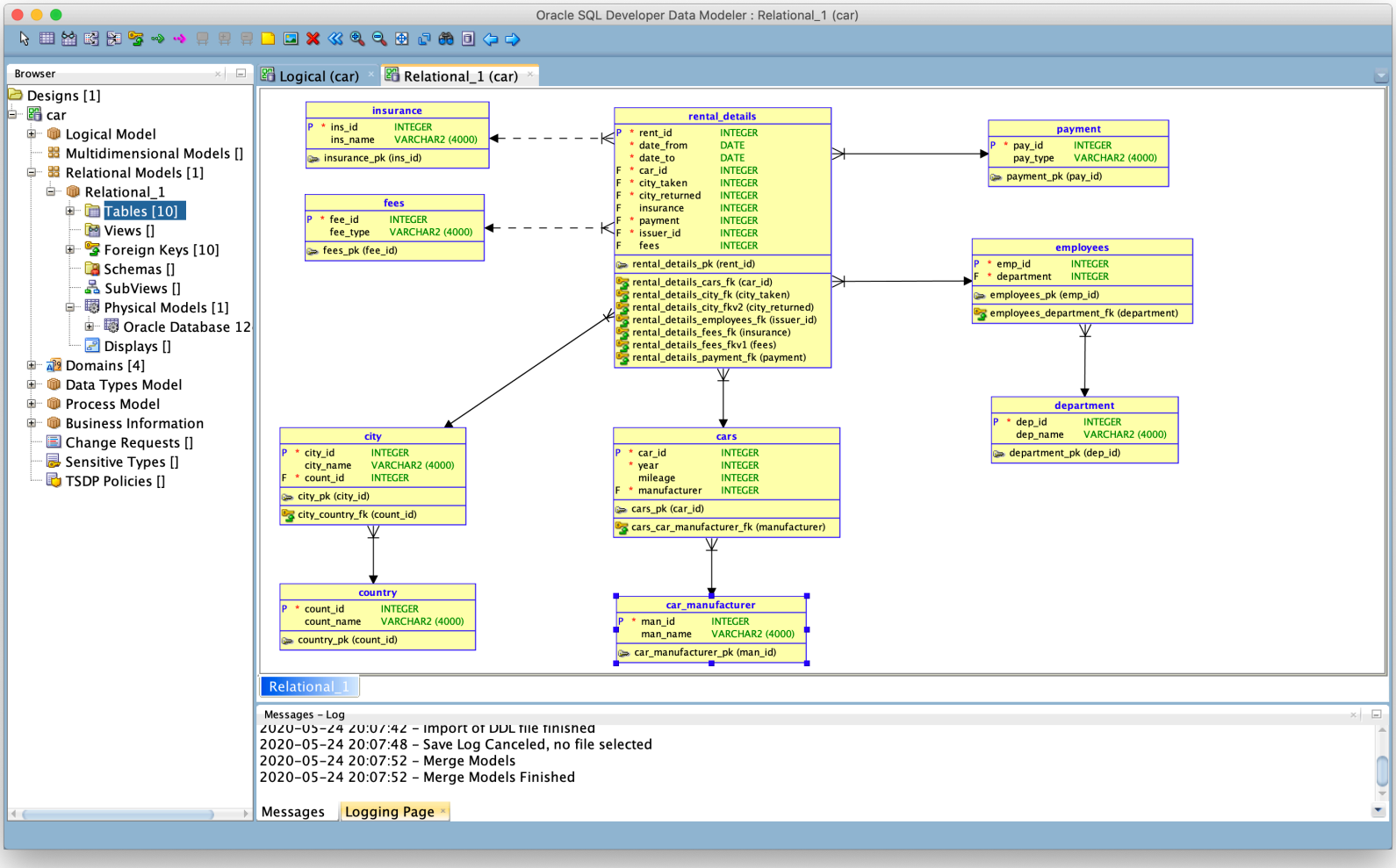
May 24, 2020

Oracle project

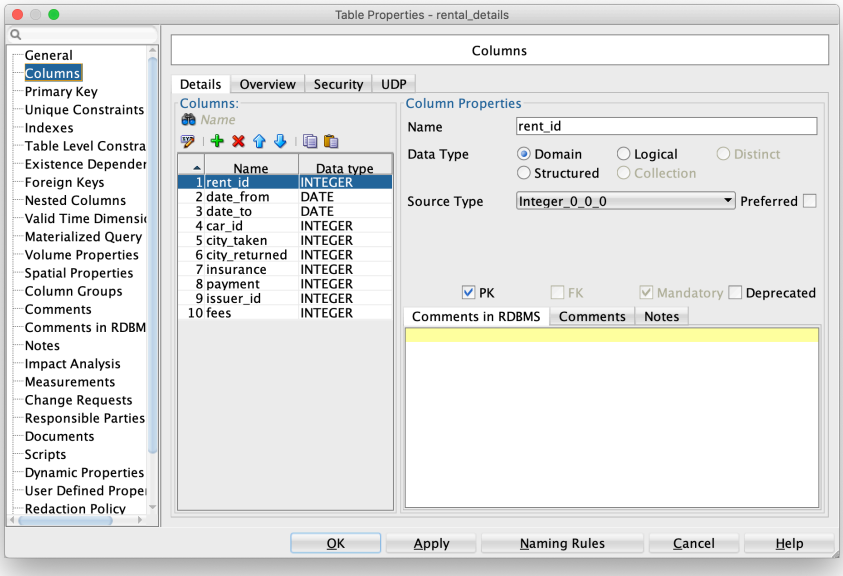
Database modeling

First step was to decide which project to work on. Since I have a little understanding in a car rental field, I decided to stick with that idea. Next, I was relying on the given criteria of 10 tables.

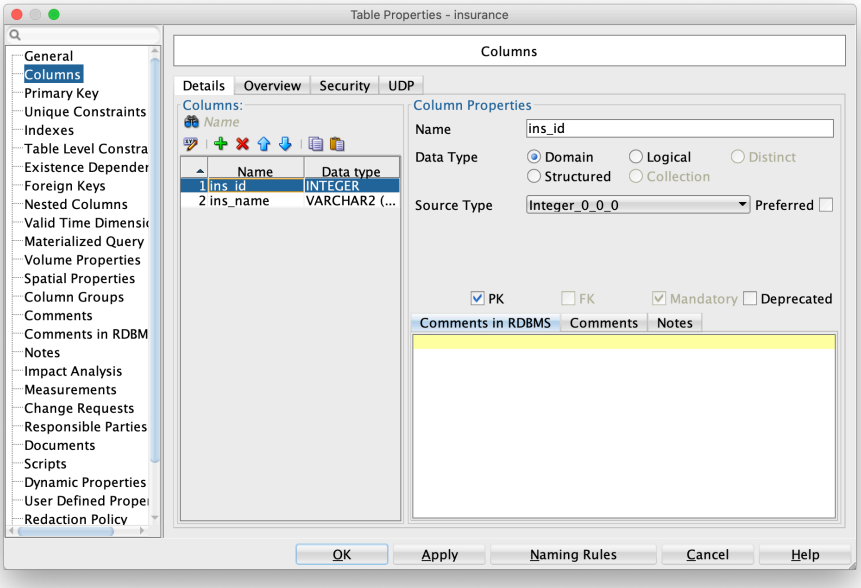
The modeling of the Data Base I decided to start in SQL Developer Data Modeler as you suggested. The model I ended up with have the following structure:



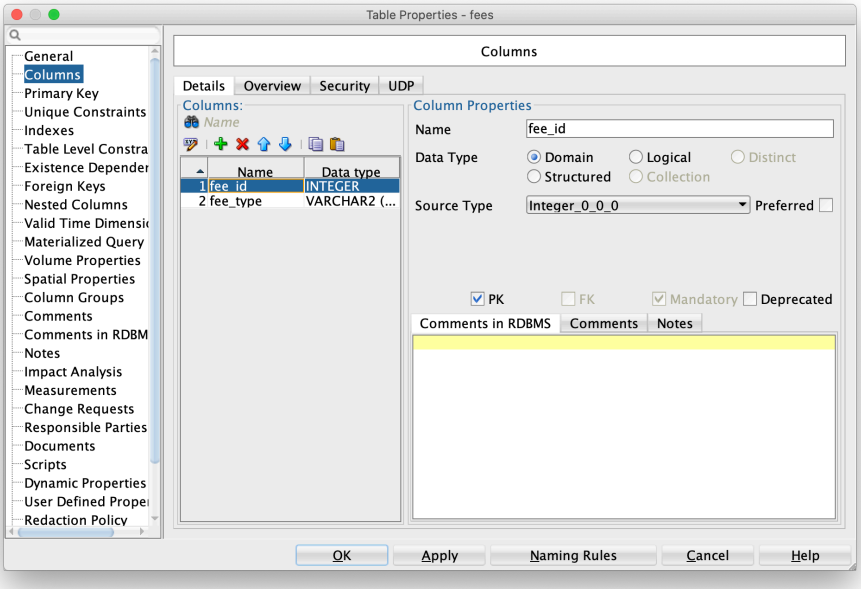
As you can see one of the most important tables is RENTAL_DETAILS. It contains following columns: Rent_id - integer, obligatory, primary key of the table, basically incrementing counter of the rows; Date_from - date type, obligatory, information when the booking has started, to simplify the data I gave up on the time which will be quite important on a real world solution; Date_to - date type, obligatory, similarly to the previous one but indicates when the car was returned; Car_id - integer, obligatory, foreign key, references to Cars table with the car data; City_taken - integer, obligatory, foreign key, references to City table with city names; City_returned - integer, obligatory, foreign key, similar to the previous one; Insurance - integer, optional, foreign key, indicates whether a client selected any insurance, references to Insurance table with all kinds of insurance; Payment - integer, obligatory, foreign key, indicates the type of payment client choose, references to table Payment; Issuer_id - integer, obligatory, foreign key, indicates which employee was operating this rental, references to Employees table; Fees - integer, optional, foreign key, depending on the result of rental which caused any problems might refer to the table Fees with description of the problem.



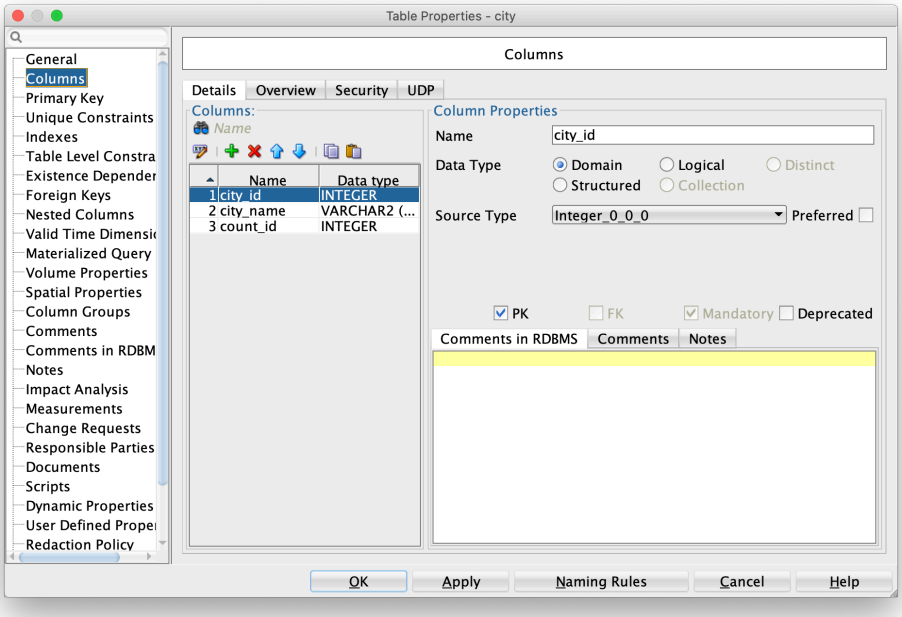
Next is an Insurance table which contains Ins_id - integer, obligatory, primary key - ID of insurance and Ins_name - varchar2, obligatory, description of the insurance a client might have purchased.



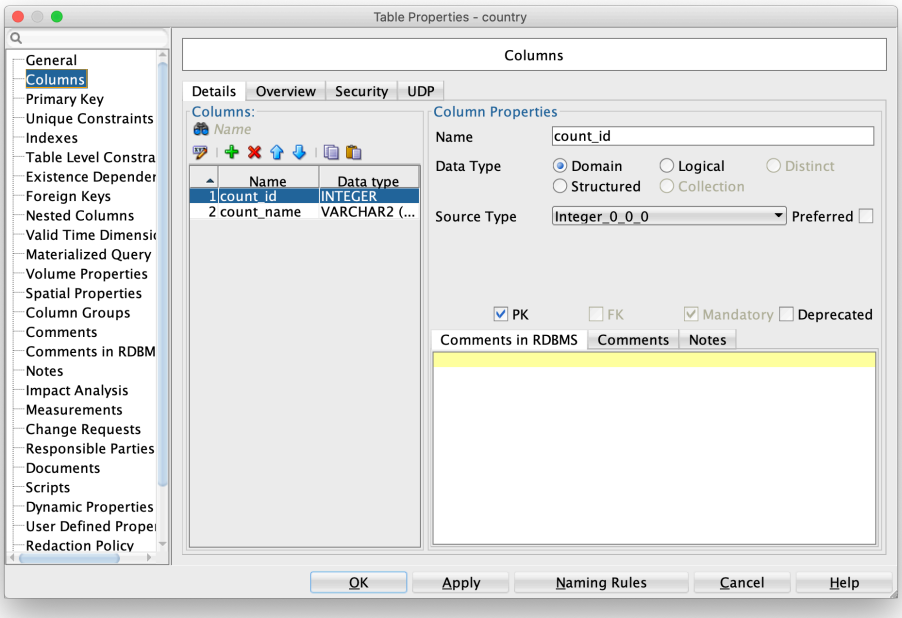
After that is Fees table. It has Fee_id - integer, obligatory, primary key, an ID of the fee might be applied for the rental and Fee_type - varchar2, obligatory - description of the fee applied.



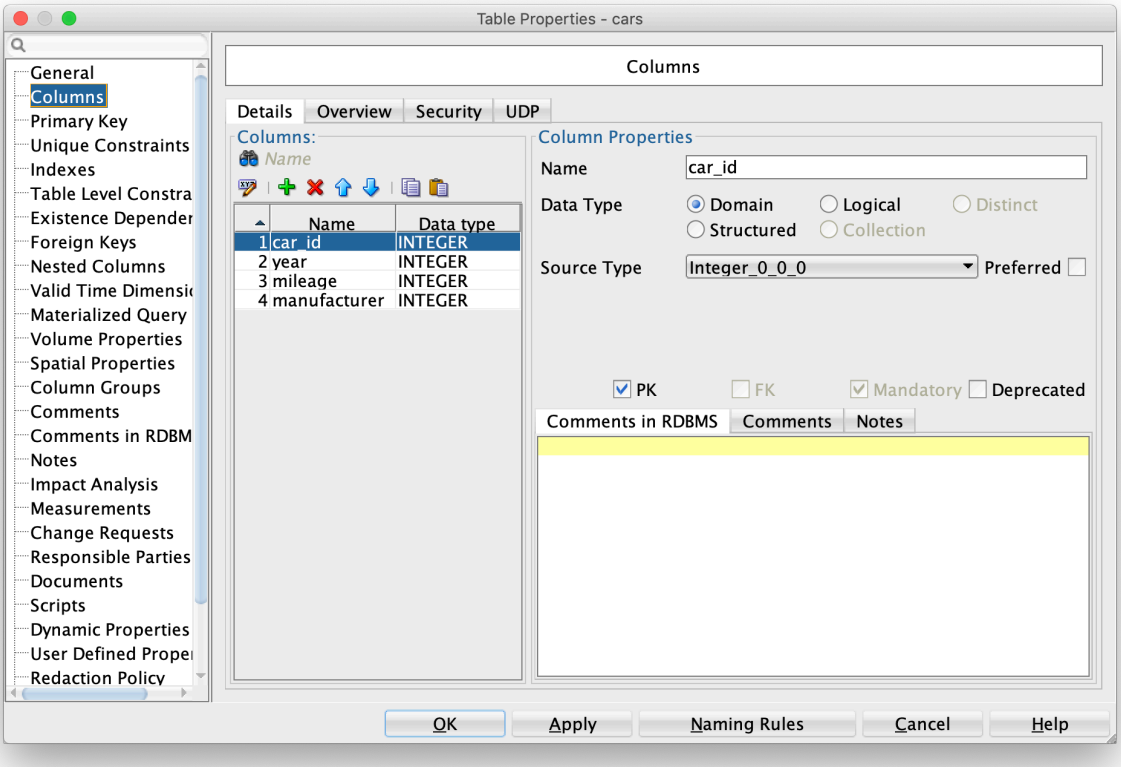
Follows up City table including columns: City_id - integer, obligatory, primary key, an ID of the city; City_name - varchar2, obligatory, basically a name of the city; Count_id - integer, obligatory, foreign key, references to Country table, which is saying in which country this city is.



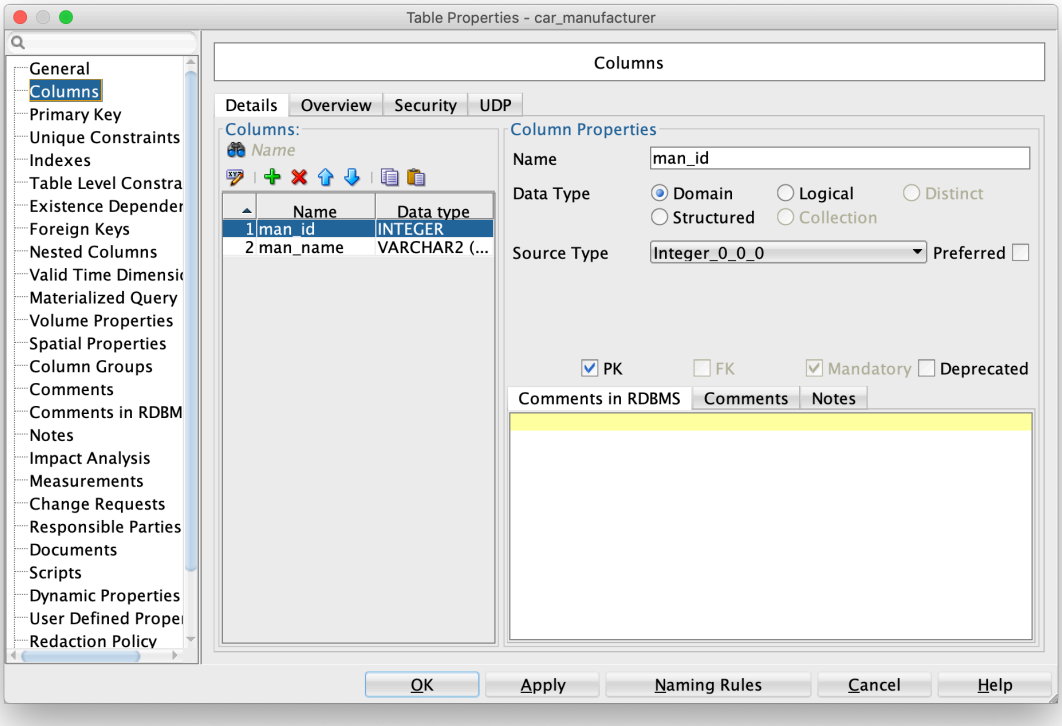
Next is Country table. It has: Count_id - integer, obligatory, primary key, an ID of the country, Count_name - varchar2, obligatory, a name of the country.



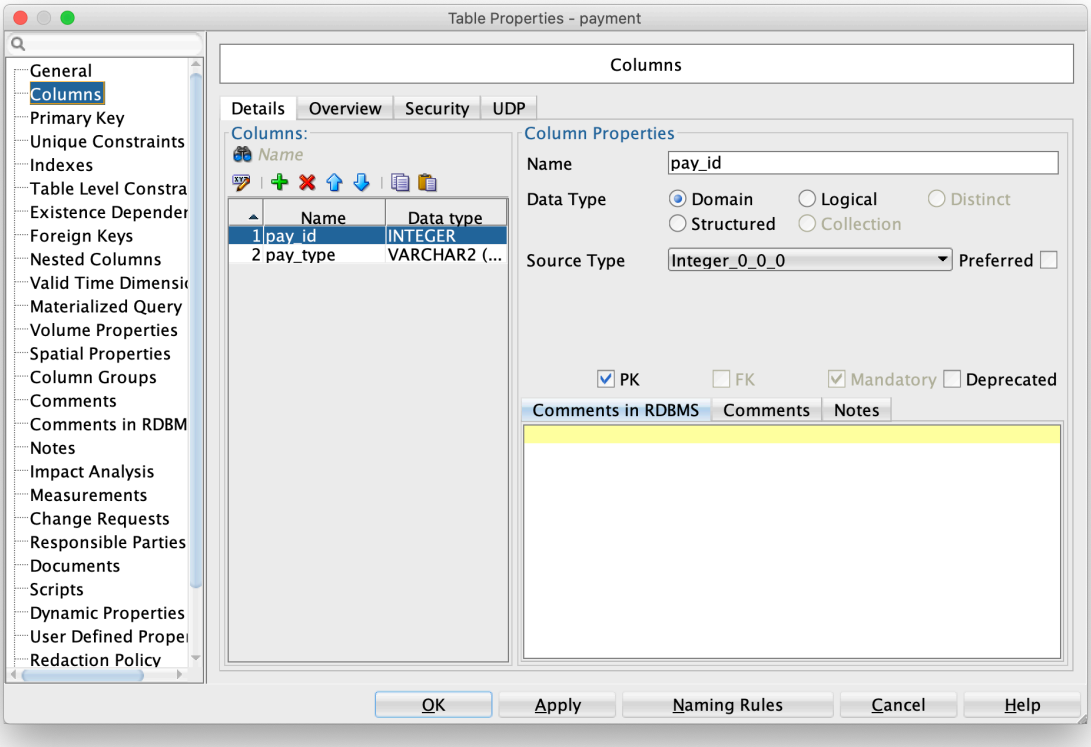
After that is Cars table with the following columns: Car_id - integer, obligatory, primary key, an ID of each car in rental; Year - integer, obligatory, a year of manufacture of the car; Mileage - integer, obligatory, a number miles this car ran; Manufacturer - integer, obligatory, foreign key - references to the Car_manufacturer table with the names of manufacturers.



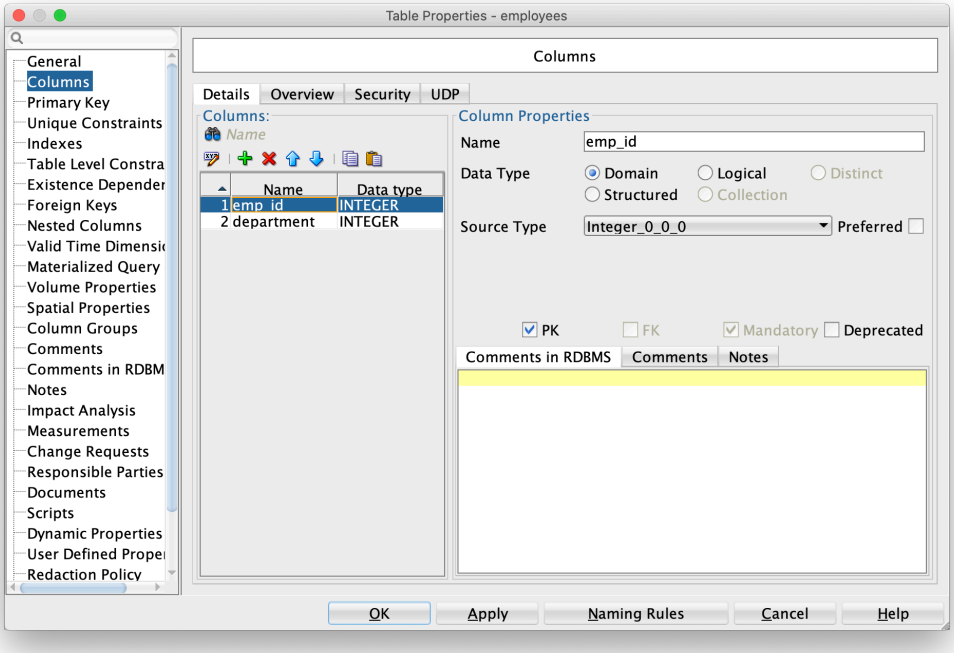
Next is Car_manufacturer table. Two columns are in here: Man_id - integer, obligatory, primary key, an ID of the manufacturer; Man_name - varchar2, obligatory, a name of manufacturer.



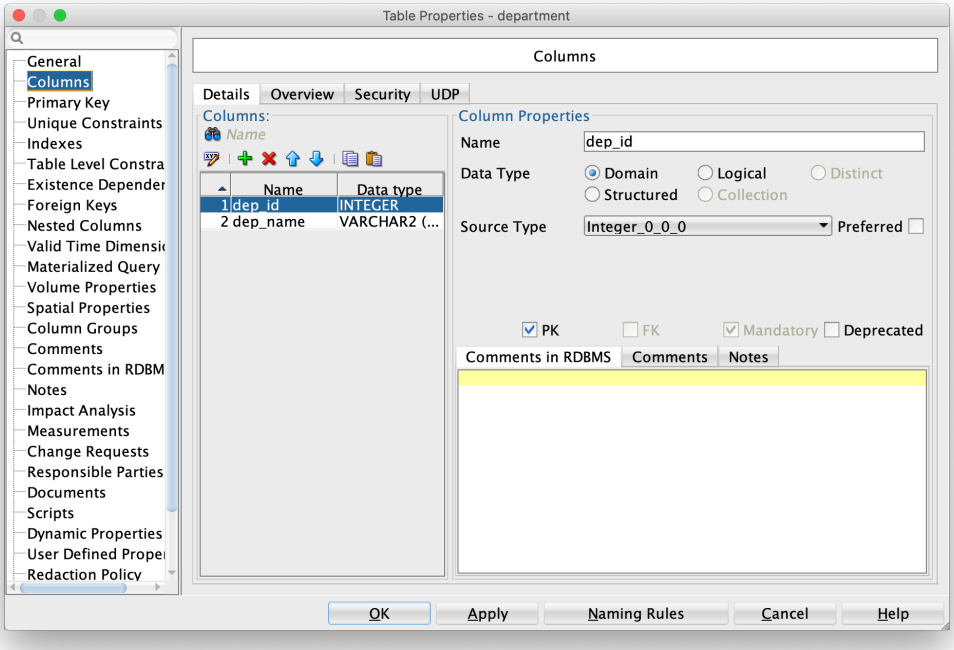
Payment table contains of Pay_id - integer, obligatory, primary key, an Id of type of payment; Pay_type - varchar2, obligatory, explains what type of payment has been used by customer.



Employees table: Emp_id - integer, obligatory, primary key, an ID of each employee; Department - integer, obligatory, foreign key, references to Department table and indicates in which department person works.



The last table is Departments one. It has Dep_id - integer, obligatory, primary key, an ID of each department; Dep_name - varchar2, obligatory, unique list of name related to each department.



Data generation

For this step I decided to generate data using Python.

Beneath is the code I used for generation data for Cars table:

In [8]:
```
import numpy as np
import pandas as pd
```
executed in 5ms, finished 21:53:01 2020-05-22

In [4]:
```
df = pd.DataFrame(columns=['CAR_ID','YEAR','MILEAGE','MANUFACTURER'])
```
executed in 56ms, finished 21:50:06 2020-05-22

In [25]:
```
for i in range(1, 110000):
    year = np.random.randint(1950,2020)
    mil = np.random.randint(0,100000)
    man = np.random.randint(1,15)
    df.loc[i-1] = {'CAR_ID':i, 'YEAR':year, 'MILEAGE':mil, 'MANUFACTURER':man}
```
executed in 9m 28s, finished 22:28:29 2020-05-22

In [28]:
```
df.head()
```
executed in 20ms, finished 22:28:59 2020-05-22

Out[28]:

	CAR_ID	YEAR	MILEAGE	MANUFACTURER
0	1	1999	92313	8
1	2	1962	52923	3
2	3	1954	583	1
3	4	1983	45924	6
4	5	1990	81769	12

In [30]:
```
df.to_csv("/Users/vitalij/Desktop/cars.csv", index=False)
```

Similarly for Employees table:

In [31]:
```
emp = pd.DataFrame(columns=['EMP_ID','DEPARTMENT'])
```
executed in 27ms, finished 22:43:06 2020-05-22

In [34]:
```
for i in range(1, 500):
    man = np.random.randint(1,4)
    emp.loc[i-1] = {'EMP_ID':i, 'DEPARTMENT':man}
```
executed in 1.15s, finished 22:45:59 2020-05-22

In [35]:
```
emp.to_csv("/Users/vitalij/Desktop/emp.csv", index=False)
```
executed in 12ms, finished 22:46:01 2020-05-22

The most difficult generating was for Rent_details table because it has data type columns:

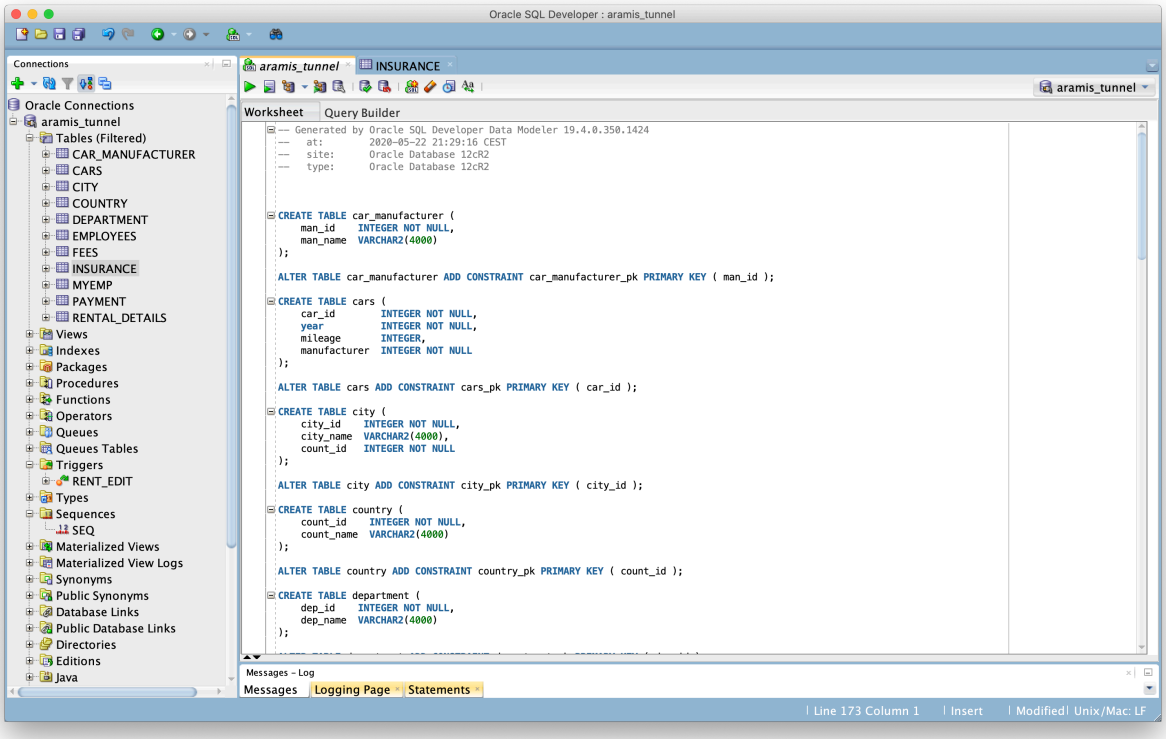
In [42]:
```
det = pd.DataFrame(columns=['RENT_ID','DATE_FROM','DATE_TO','CAR_ID',
                            'CITY_TAKEN','CITY_RETURNED','INSURANCE','PAYMENT','ISSUER_ID','FEES'])
```
executed in 10ms, finished 23:03:43 2020-05-22

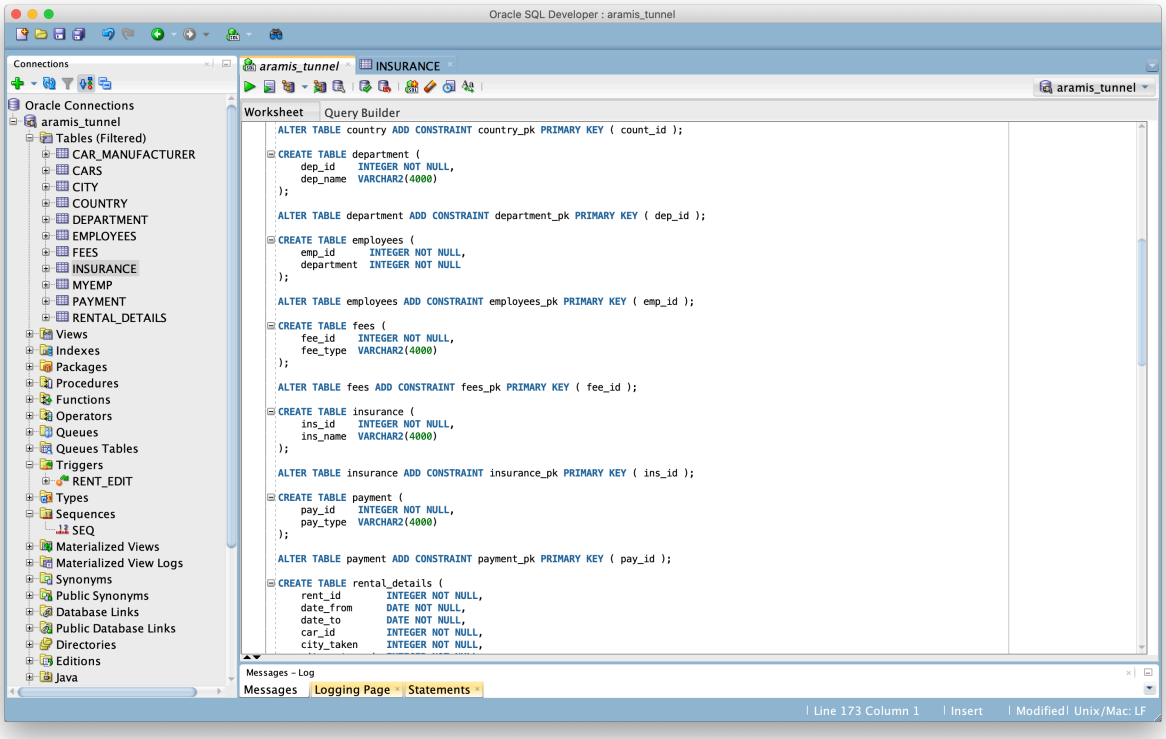
In [61]:
```
for i in range(1, 150000):
    d1 = np.random.randint(2010,2020)
    d11 = np.random.randint(1,12)
    d111 = np.random.randint(1,28)
    d2 = np.random.randint(2010,2020)
    d22 = np.random.randint(1,12)
    d222 = np.random.randint(1,28)
    car = np.random.randint(1,110000)
    city1 = np.random.randint(1,8)
    city2 = np.random.randint(1,8)
    ins =  np.random.randint(1,4)
    pay =  np.random.randint(1,4)
    emp = np.random.randint(1,500)
    fee = np.random.randint(1,5)
    date1 = str(min(d1,d2))+'-'+str(min(d11,d22))+'-'+str(min(d111,d222))
    date2 = str(max(d1,d2))+'-'+str(max(d11,d22))+'-'+str(max(d111,d222))
    det.loc[i-1] = {'RENT_ID':i, 'DATE_FROM':date1,'DATE_TO':date2, 'CAR_ID':car, 'CITY_TAKEN':city1,
                    'CITY_RETURNED':city2, 'INSURANCE':ins, 'PAYMENT':pay, 'ISSUER_ID':emp, 'FEES':fee  }
```
executed in 53m 15s, finished 00:56:26 2020-05-23

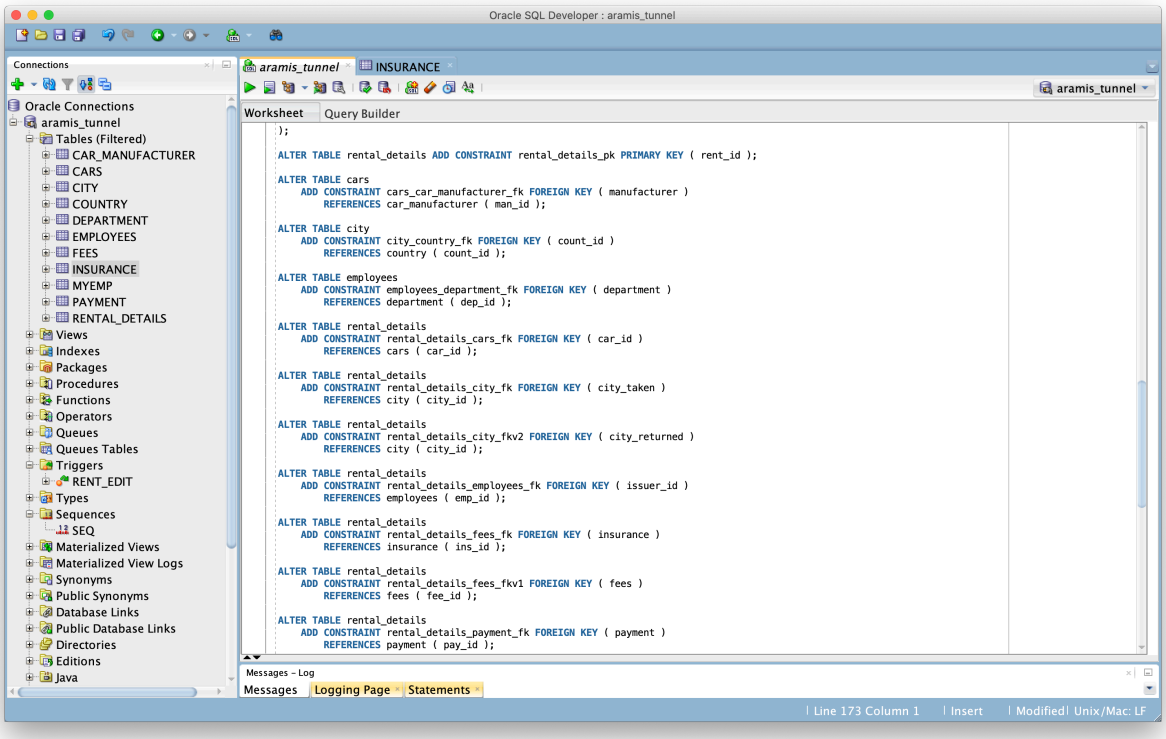
In [62]:
```
det.to_csv("/Users/vitalij/Desktop/det.csv", index=False)
```
executed in 666ms, finished 01:22:59 2020-05-23

Data import

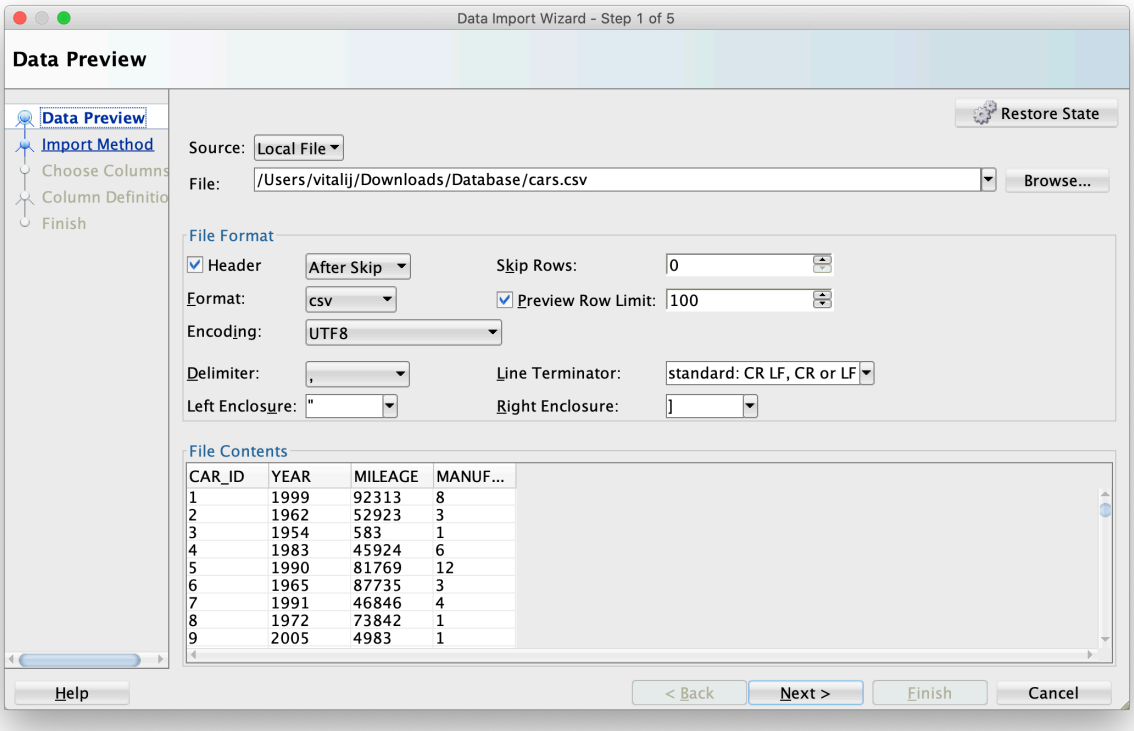
Next step was to import all the data. First, from SQL Modeler to SQL Developer using import as DDL:



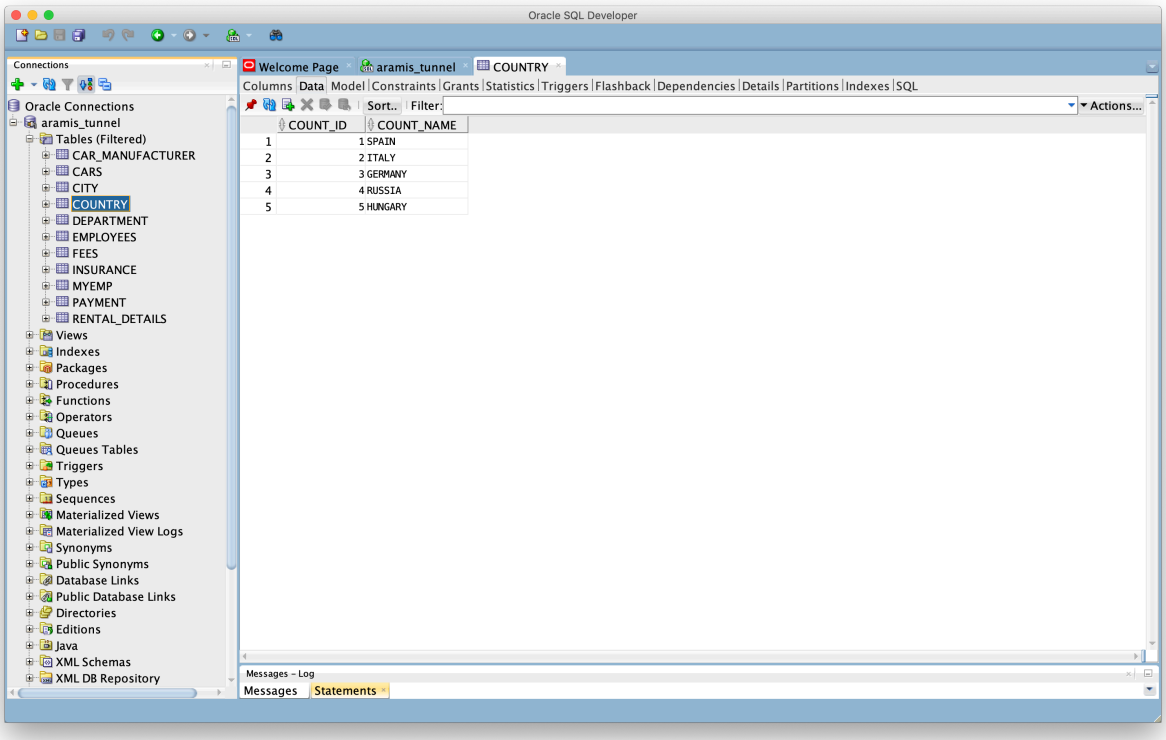


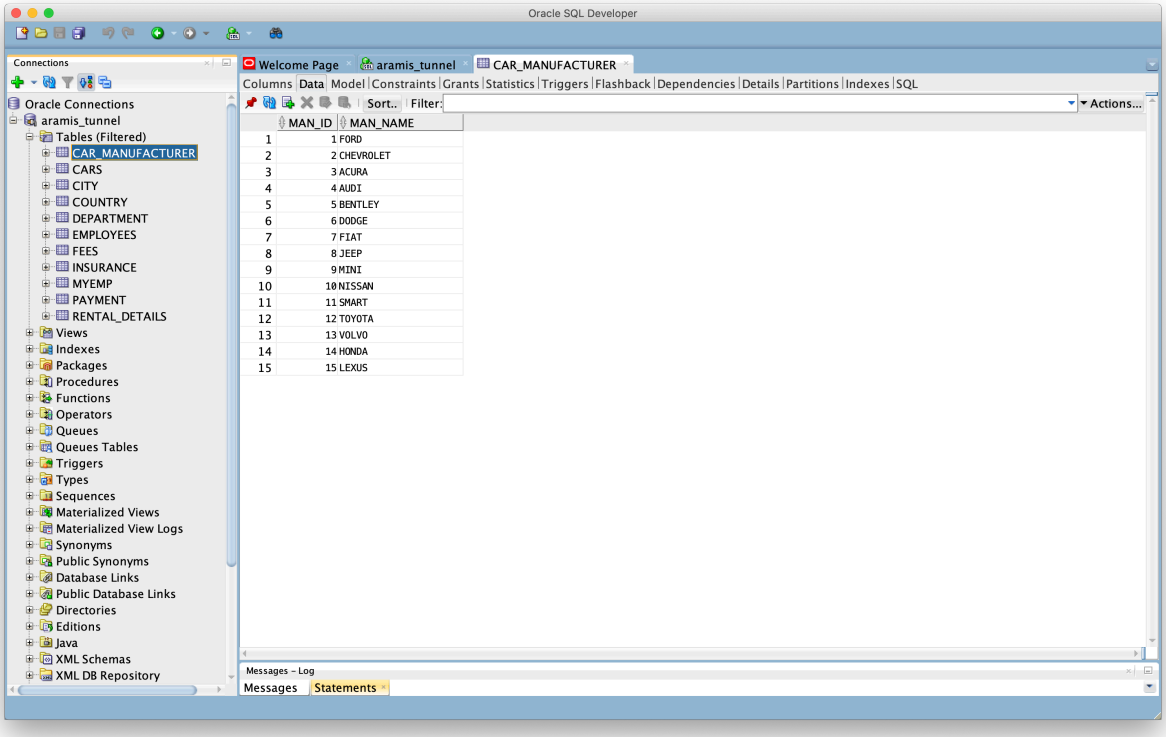


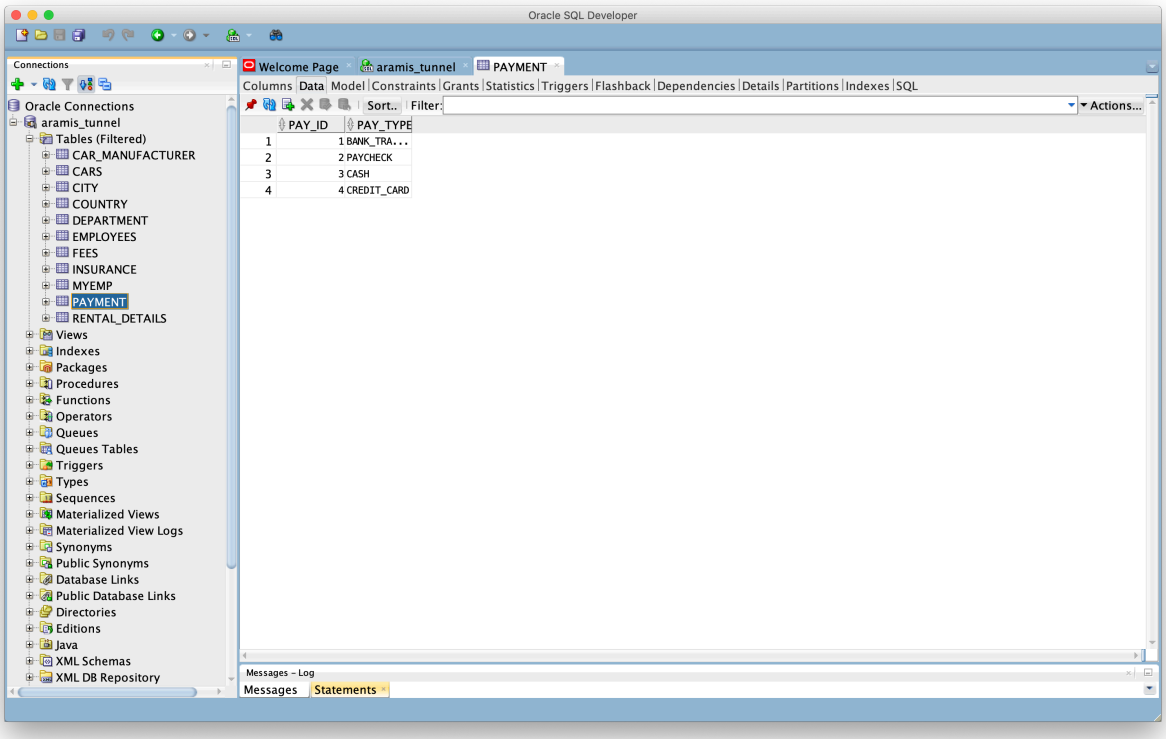
Next step is to import csv files I have generated before using Import Data by the right click on the newly generated tables Cars, Rental_details, Employees:

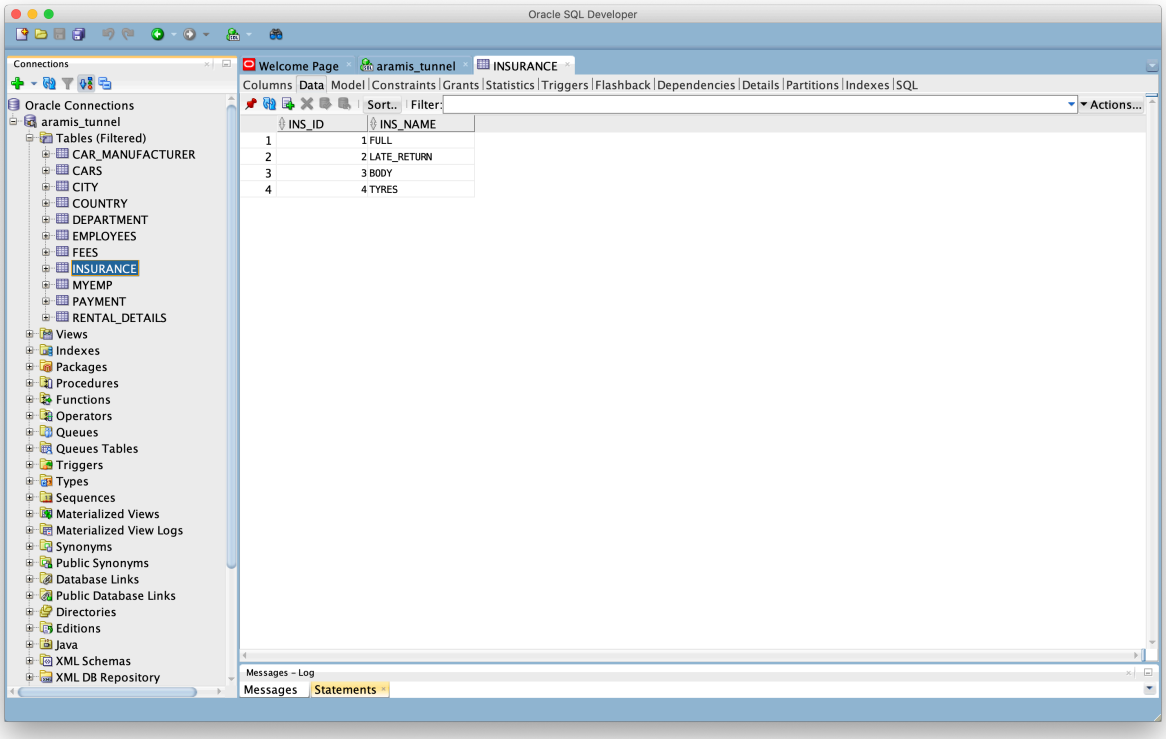


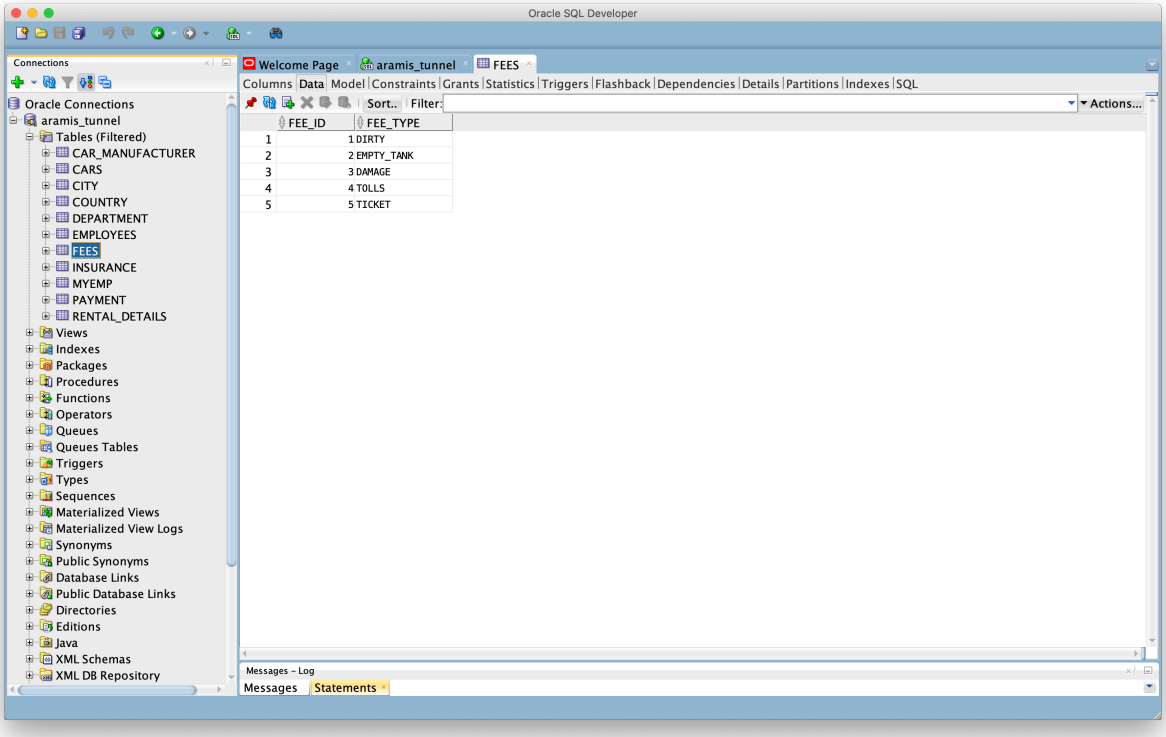
For all other tables I added data manually:

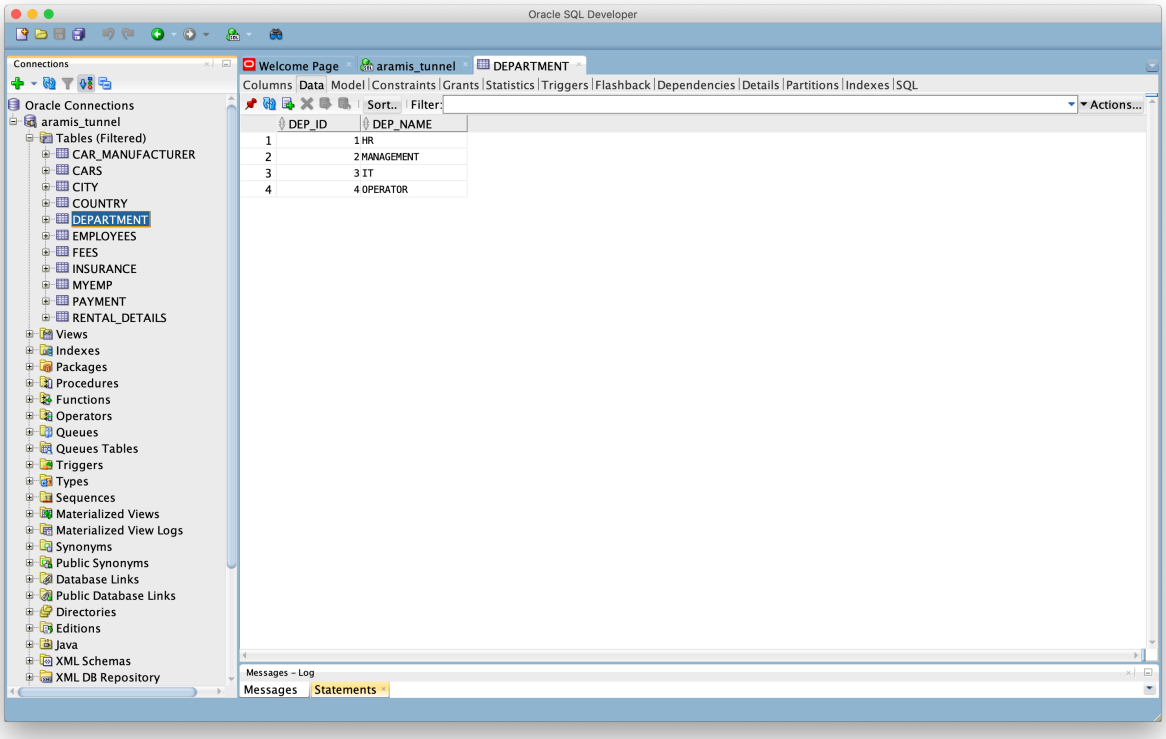


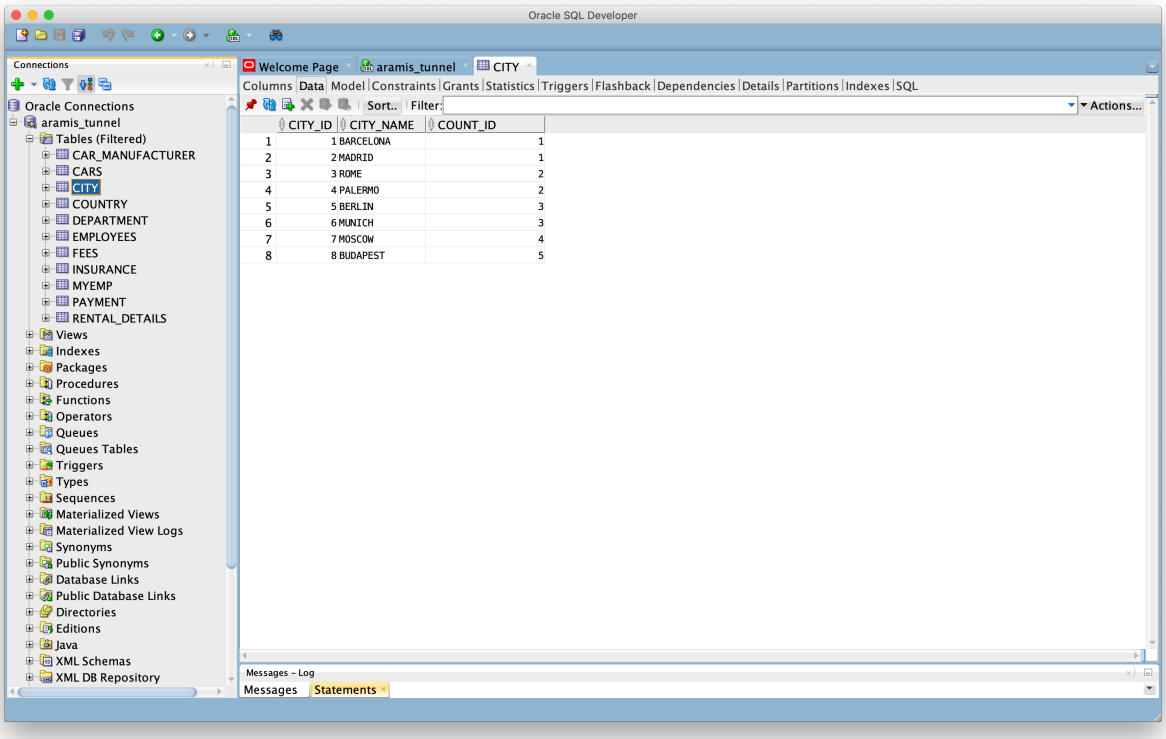



Oracle SQL Developer
Connections
Oracle Connections
aramis_tunnel
Tables (Filtered)
CAR_MANUFACTURER
CARS
CITY
COUNTRY
DEPARTMENT
EMPLOYEES
FEES
INSURANCE
MYEMP
PAYMENT
RENTAL_DETAILS
Views
Indexes
Packages
Procedures
Functions
Operators
Queues
Queues Tables
Triggers
Types
Sequences
Materialized Views
Materialized View Logs
Synonyms
Public Synonyms
Database Links
Public Database Links
Directories
Editions
Java
XML Schemas
XML DB Repository
Welcome Page
aramis_tunnel
PAYMENT
Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL
Sort..
Filter:
Actions...
PAY_ID
PAY_TYPE
1 1 BANK_TRA...
2 2 PAYCHECK
3 3 CASH
4 4 CREDIT_CARD
Messages - Log
Messages
Statements

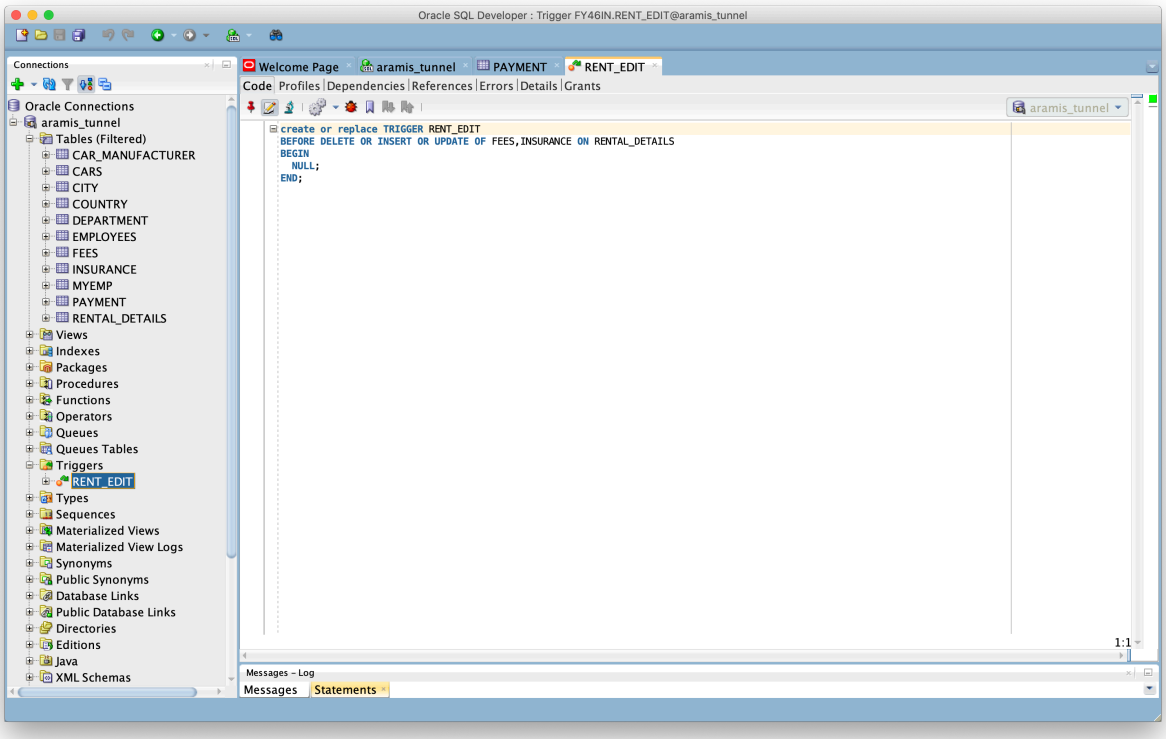

Oracle SQL Developer
Connections
Oracle Connections
aramis_tunnel
Tables (Filtered)
CAR_MANUFACTURER
CARS
CITY
COUNTRY
DEPARTMENT
EMPLOYEES
FEES
INSURANCE
MYEMP
PAYMENT
RENTAL_DETAILS
Views
Indexes
Packages
Procedures
Functions
Operators
Queues
Queues Tables
Triggers
Types
Sequences
Materialized Views
Materialized View Logs
Synonyms
Public Synonyms
Database Links
Public Database Links
Directories
Editions
Java
XML Schemas
XML DB Repository
Welcome Page
aramis_tunnel
INSURANCE
Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL
Sort..
Filter:
Actions...
INS_ID
INS_NAME
1 1 FULL
2 2 LATE_RETURN
3 3 BODY
4 4 TYRES
Messages - Log
Messages
Statements

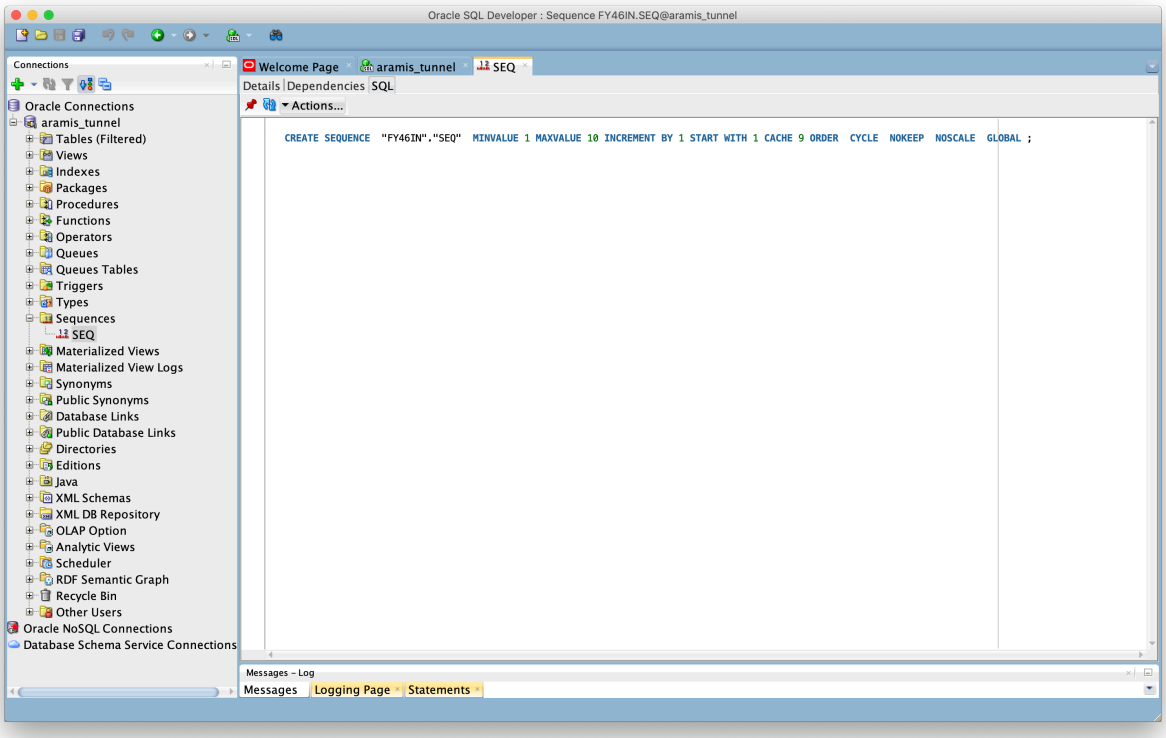

Oracle SQL Developer
Connections
Oracle Connections
aramis_tunnel
Tables (Filtered)
CAR_MANUFACTURER
CARS
CITY
COUNTRY
DEPARTMENT
EMPLOYEES
FEES
INSURANCE
MYEMP
PAYMENT
RENTAL_DETAILS
Views
Indexes
Packages
Procedures
Functions
Operators
Queues
Queues Tables
Triggers
Types
Sequences
Materialized Views
Materialized View Logs
Synonyms
Public Synonyms
Database Links
Public Database Links
Directories
Editions
Java
XML Schemas
XML DB Repository
Welcome Page
aramis_tunnel
FEES
Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL
Sort..
Filter:
Actions...
FEE_ID FEE_TYPE
1 1 DIRTY
2 2 EMPTY_TANK
3 3 DAMAGE
4 4 TOLLS
5 5 TICKET
Messages – Log
Messages
Statements


Oracle SQL Developer
Connections
Oracle Connections
aramis_tunnel
Tables (Filtered)
CAR_MANUFACTURER
CARS
CITY
COUNTRY
DEPARTMENT
EMPLOYEES
FEES
INSURANCE
MYEMP
PAYMENT
RENTAL_DETAILS
Views
Indexes
Packages
Procedures
Functions
Operators
Queues
Queues Tables
Triggers
Types
Sequences
Materialized Views
Materialized View Logs
Synonyms
Public Synonyms
Database Links
Public Database Links
Directories
Editions
Java
XML Schemas
XML DB Repository
Welcome Page
aramis_tunnel
DEPARTMENT
Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL
Sort..
Filter:
Actions...
DEP_ID DEP_NAME
1 1 HR
2 2 MANAGEMENT
3 3 IT
4 4 OPERATOR
Messages – Log
Messages
Statements

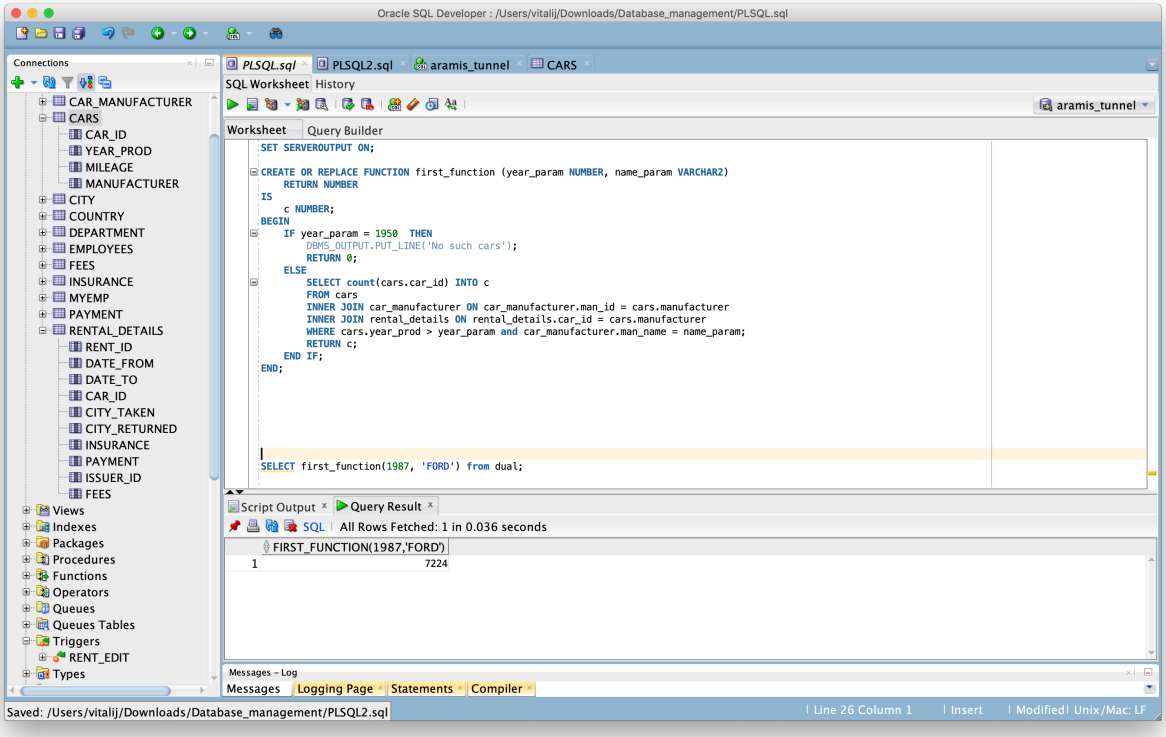


And some triggers and sequences:

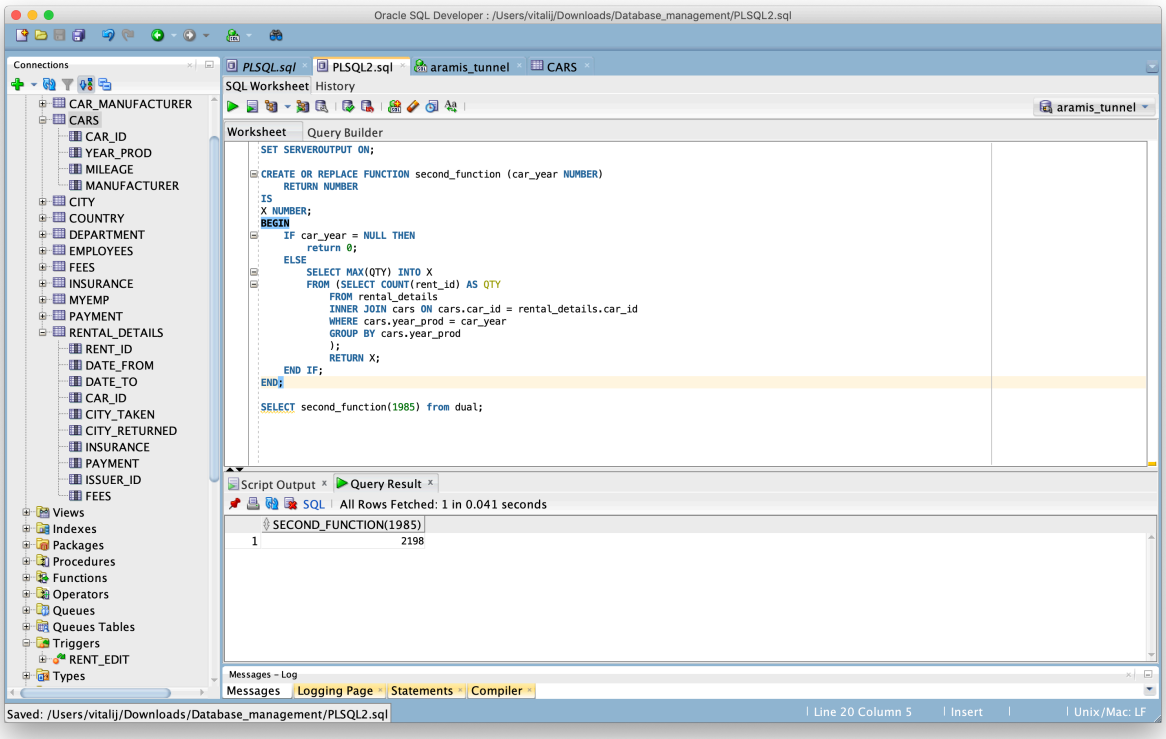




PLSQL functions



This function accepts year and name as parameters and looking in Cars table for a quantity of the cars which are newer than given year and also these cars should be from the manufacturer given as name parameter.



The second function calculates the maximum quantity of usage of the cars produced in given year which are in Rental_details table.

NoSQL database

I decided to work with MongoDB as NoSQL Database because it is very easy to install and setup the MongoDB, the very basic feature of MongoDB is that it is a schema-less database compare to Oracle one. Also one of the main factor was a good amount of available documentation and MongoDB does not require a VM to be run. Since, it is a NOSQL database, then it is obviously secure because no sql injection can be made.

First step after installation is to load the data into DB:

```
(base) Vitalijs-MacBook-Pro:~ vitalij$ mongoimport --db rent --collection rent --type csv --file /Users/vitalij/Downloads/Database/det.
csv --headerline
2020-05-24T15:27:34.924+0200    connected to: mongodb://localhost/
2020-05-24T15:27:37.002+0200    149999 document(s) imported successfully. 0 document(s) failed to import.
(base) Vitalijs-MacBook-Pro:~ vitalij$ mongoimport --db rent --collection cars --type csv --file /Users/vitalij/Downloads/Database/cars
.csv --headerline
2020-05-24T15:28:26.499+0200    connected to: mongodb://localhost/
2020-05-24T15:28:27.583+0200    109999 document(s) imported successfully. 0 document(s) failed to import.
(base) Vitalijs-MacBook-Pro:~ vitalij$ mongoimport --db rent --collection manufact --type csv --file /Users/vitalij/Downloads/Database/
manufact.csv --headerline
2020-05-24T15:28:56.069+0200    connected to: mongodb://localhost/
2020-05-24T15:28:56.097+0200    15 document(s) imported successfully. 0 document(s) failed to import.
```

I have uploaded 3 collections (tables) from the previous exercise. And I had to switch to the current DB I was going to work with:

```
> use rent
switched to db rent
> show collections
cars
manufact
rent
> db.rent.cars.count()
0
> db.rent.count()
149999
> db.cars.count()
109999
> db.manufact.count()
15
```

Looks like everything was loaded.

NoSQL queries

1. Counting number of cars which were taken or returned from city with index 2

```
> db.rent.find({$or: [{CITY_TAKEN: 2},{CITY_RETURNED: 2}]}).count()
39554
```

2. Picking one row where city the car was taken from has index 5

```
> db.rent.findOne({CITY_TAKEN : 5})
{
        "_id" : ObjectId("5eca7646bf85c7fc0bfe179c"),
        "RENT_ID" : 3,
        "DATE_FROM" : "2013-4-3",
        "DATE_TO" : "2014-10-17",
        "CAR_ID" : 94805,
        "CITY_TAKEN" : 5,
        "CITY_RETURNED" : 7,
        "INSURANCE" : 3,
        "PAYMENT" : 2,
        "ISSUER_ID" : 473,
        "FEES" : 3
}
```

3. Counting number of cars with a mileage more than 90000 km

```
> db.cars.find({MILEAGE: {$gt: 90000 }}).count()
10740
```

4. Counting number of cars older than 2000 and with a manufacturer index 8

```
> db.cars.find({$and: [{YEAR: {$lt: 2000 }},{MANUFACTURER: 8}]}).count()
5567
```

5. The following aggregation operation selects documents with INSURANCE equal to 1 and sorts the results by the FEES field in descending order

```
> db.rent.aggregate([{$match: {INSURANCE: 1}},{$sort: {FEES: -1}}])
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe179a"), "RENT_ID" : 8, "DATE_FROM" : "2013-9-5", "DATE_TO" : "2014-10-6", "CAR_ID" : 85003, "CITY_TAKEN" : 1, "CITY_RETURNED" : 7, "INSURANCE" : 1, "PAYMENT" : 2, "ISSUER_ID" : 444, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe179d"), "RENT_ID" : 2, "DATE_FROM" : "2010-5-2", "DATE_TO" : "2019-8-23", "CAR_ID" : 53740, "CITY_TAKEN" : 5, "CITY_RETURNED" : 5, "INSURANCE" : 1, "PAYMENT" : 2, "ISSUER_ID" : 67, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe17a1"), "RENT_ID" : 16, "DATE_FROM" : "2011-6-13", "DATE_TO" : "2014-10-21", "CAR_ID" : 24077, "CITY_TAKEN" : 6, "CITY_RETURNED" : 2, "INSURANCE" : 1, "PAYMENT" : 3, "ISSUER_ID" : 481, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe17a4"), "RENT_ID" : 19, "DATE_FROM" : "2013-10-12", "DATE_TO" : "2019-11-27", "CAR_ID" : 37691, "CITY_TAKEN" : 2, "CITY_RETURNED" : 3, "INSURANCE" : 1, "PAYMENT" : 3, "ISSUER_ID" : 375, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe17a5"), "RENT_ID" : 20, "DATE_FROM" : "2010-6-4", "DATE_TO" : "2019-8-16", "CAR_ID" : 74925, "CITY_TAKEN" : 2, "CITY_RETURNED" : 6, "INSURANCE" : 1, "PAYMENT" : 2, "ISSUER_ID" : 239, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe17c5"), "RENT_ID" : 49, "DATE_FROM" : "2011-4-4", "DATE_TO" : "2018-11-7", "CAR_ID" : 34942, "CITY_TAKEN" : 7, "CITY_RETURNED" : 1, "INSURANCE" : 1, "PAYMENT" : 3, "ISSUER_ID" : 8, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe17d5"), "RENT_ID" : 65, "DATE_FROM" : "2010-7-8", "DATE_TO" : "2011-10-11", "CAR_ID" : 62299, "CITY_TAKEN" : 7, "CITY_RETURNED" : 2, "INSURANCE" : 1, "PAYMENT" : 1, "ISSUER_ID" : 385, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe17ea"), "RENT_ID" : 87, "DATE_FROM" : "2015-2-26", "DATE_TO" : "2016-4-27", "CAR_ID" : 13932, "CITY_TAKEN" : 1, "CITY_RETURNED" : 5, "INSURANCE" : 1, "PAYMENT" : 3, "ISSUER_ID" : 297, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe17f1"), "RENT_ID" : 90, "DATE_FROM" : "2011-5-4", "DATE_TO" : "2012-7-13", "CAR_ID" : 29711, "CITY_TAKEN" : 6, "CITY_RETURNED" : 5, "INSURANCE" : 1, "PAYMENT" : 1, "ISSUER_ID" : 361, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe17f2"), "RENT_ID" : 93, "DATE_FROM" : "2010-4-5", "DATE_TO" : "2018-10-7", "CAR_ID" : 9073, "CITY_TAKEN" : 4, "CITY_RETURNED" : 6, "INSURANCE" : 1, "PAYMENT" : 2, "ISSUER_ID" : 461, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe17ff"), "RENT_ID" : 107, "DATE_FROM" : "2010-4-20", "DATE_TO" : "2013-9-24", "CAR_ID" : 78179, "CITY_TAKEN" : 3, "CITY_RETURNED" : 2, "INSURANCE" : 1, "PAYMENT" : 3, "ISSUER_ID" : 47, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe1803"), "RENT_ID" : 111, "DATE_FROM" : "2010-7-22", "DATE_TO" : "2015-8-24", "CAR_ID" : 7430, "CITY_TAKEN" : 2, "CITY_RETURNED" : 7, "INSURANCE" : 1, "PAYMENT" : 2, "ISSUER_ID" : 251, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe1804"), "RENT_ID" : 113, "DATE_FROM" : "2015-7-1", "DATE_TO" : "2019-10-4", "CAR_ID" : 67392, "CITY_TAKEN" : 1, "CITY_RETURNED" : 7, "INSURANCE" : 1, "PAYMENT" : 3, "ISSUER_ID" : 446, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe1805"), "RENT_ID" : 112, "DATE_FROM" : "2015-2-16", "DATE_TO" : "2017-6-18", "CAR_ID" : 75177, "CITY_TAKEN" : 6, "CITY_RETURNED" : 1, "INSURANCE" : 1, "PAYMENT" : 1, "ISSUER_ID" : 470, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe180c"), "RENT_ID" : 120, "DATE_FROM" : "2013-5-19", "DATE_TO" : "2019-8-21", "CAR_ID" : 61041, "CITY_TAKEN" : 6, "CITY_RETURNED" : 4, "INSURANCE" : 1, "PAYMENT" : 2, "ISSUER_ID" : 494, "FEES" : 4 }
{ "_id" : ObjectId("5eca7646bf85c7fc0bfe1814"), "RENT_ID" : 128, "DATE_FROM" : "2011-6-16", "DATE_TO" : "2016-11-23", "CAR_ID" : 79226, "CITY_TAKEN" : 6, "CITY_RETURNED" : 5, "INSURANCE" : 1, "PAYMENT" : 1, "ISSUER_ID" : 144, "FEES" : 4 }
```

6. Groups the documents by the PAYMENT field and calculates the total for each FEES field from the sum of the amount field

```
> db.rent.aggregate({$group: {_id: "$PAYMENT", total:{$sum: "$FEES"}}})
{ "_id" : 3, "total" : 125252 }
{ "_id" : 1, "total" : 125029 }
{ "_id" : 2, "total" : 124858 }
```

7. Groups the cars by YEAR which calculates the total age of cars which were manufactured in the same year in ascending order

```
> db.cars.aggregate([{$group: {_id:"$YEAR", total:{$sum: "$YEAR"}}},{ $sort: { total: 1 } },{$limit:3}])
{ "_id" : 1964, "total" : 2947964 }
{ "_id" : 1996, "total" : 2950088 }
{ "_id" : 1950, "total" : 2950350 }
```

8. Finds one row with YEAR = 2000 and MANUFACTURER = 8 and delete it

```
> db.cars.findOneAndDelete({YEAR:2000, MANUFACTURER:8})
{
        "_id" : ObjectId("5eca767a1393a73126e7a138"),
        "CAR_ID" : 594,
        "YEAR" : 2000,
        "MILEAGE" : 42741,
        "MANUFACTURER" : 8
}
```

9. Finds one row with the.given query YEAR=2019 and then resets the MILEAGE field to 0

```
> db.cars.updateOne({YEAR:2019},{$set: {MILEAGE: 0}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

10. Finds one row with MANUFACTURER = 8 and replaces with new attributes such as CAR_ID, YEAR, resets MILEAGE and MANUFACTURER

```
> db.cars.replaceOne({MANUFACTURER: 8},{CAR_ID:99999999, YEAR:2020, MILEAGE:0, MANUFACTURER: 16})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Appendix

Script to create tables:

```
CREATE TABLE car_manufacturer (
    man_id    INTEGER NOT NULL,
    man_name  VARCHAR2(4000)
);

ALTER TABLE car_manufacturer ADD CONSTRAINT car_manufacturer_pk PRIMARY KEY ( man_id );

CREATE TABLE cars (
    car_id        INTEGER NOT NULL,
    year          INTEGER NOT NULL,
    mileage       INTEGER,
    manufacturer  INTEGER NOT NULL
);

ALTER TABLE cars ADD CONSTRAINT cars_pk PRIMARY KEY ( car_id );

CREATE TABLE city (
    city_id    INTEGER NOT NULL,
    city_name  VARCHAR2(4000),
    count_id   INTEGER NOT NULL
);
```

```
ALTER TABLE city ADD CONSTRAINT city_pk PRIMARY KEY ( city_id );

CREATE TABLE country (
    count_id    INTEGER NOT NULL,
    count_name  VARCHAR2(4000)
);

ALTER TABLE country ADD CONSTRAINT country_pk PRIMARY KEY
( count_id );

CREATE TABLE department (
    dep_id    INTEGER NOT NULL,
    dep_name  VARCHAR2(4000)
);

ALTER TABLE department ADD CONSTRAINT department_pk PRIMARY KEY
( dep_id );

CREATE TABLE employees (
    emp_id      INTEGER NOT NULL,
    department  INTEGER NOT NULL
);
```

```
ALTER TABLE employees ADD CONSTRAINT employees_pk PRIMARY KEY
( emp_id );

CREATE TABLE fees (
    fee_id    INTEGER NOT NULL,
    fee_type  VARCHAR2(4000)
);

ALTER TABLE fees ADD CONSTRAINT fees_pk PRIMARY KEY ( fee_id );

CREATE TABLE insurance (
    ins_id    INTEGER NOT NULL,
    ins_name  VARCHAR2(4000)
);

ALTER TABLE insurance ADD CONSTRAINT insurance_pk PRIMARY KEY ( in-
s_id );

CREATE TABLE payment (
    pay_id    INTEGER NOT NULL,
    pay_type  VARCHAR2(4000)
);
```

```
ALTER TABLE payment ADD CONSTRAINT payment_pk PRIMARY KEY ( pay_id
);

CREATE TABLE rental_details (
    rent_id        INTEGER NOT NULL,
    date_from      DATE NOT NULL,
    date_to        DATE NOT NULL,
    car_id         INTEGER NOT NULL,
    city_taken     INTEGER NOT NULL,
    city_returned  INTEGER NOT NULL,
    insurance      INTEGER,
    payment        INTEGER NOT NULL,
    issuer_id      INTEGER NOT NULL,
    fees           INTEGER
);

ALTER TABLE rental_details ADD CONSTRAINT rental_details_pk PRIMARY
KEY ( rent_id );

ALTER TABLE cars
    ADD CONSTRAINT cars_car_manufacturer_fk FOREIGN KEY ( manufacturer )
        REFERENCES car_manufacturer ( man_id );
```

```
ALTER TABLE city
   ADD CONSTRAINT city_country_fk FOREIGN KEY ( count_id )
      REFERENCES country ( count_id );

ALTER TABLE employees
   ADD CONSTRAINT employees_department_fk FOREIGN KEY ( department )
      REFERENCES department ( dep_id );

ALTER TABLE rental_details
   ADD CONSTRAINT rental_details_cars_fk FOREIGN KEY ( car_id )
      REFERENCES cars ( car_id );

ALTER TABLE rental_details
   ADD CONSTRAINT rental_details_city_fk FOREIGN KEY ( city_taken )
      REFERENCES city ( city_id );

ALTER TABLE rental_details
   ADD CONSTRAINT rental_details_city_fkv2 FOREIGN KEY ( city_returned )
      REFERENCES city ( city_id );

ALTER TABLE rental_details
   ADD CONSTRAINT rental_details_employees_fk FOREIGN KEY ( issuer_id )
```

```
        REFERENCES employees ( emp_id );

ALTER TABLE rental_details
   ADD CONSTRAINT rental_details_fees_fk FOREIGN KEY ( insurance )
      REFERENCES insurance ( ins_id );

ALTER TABLE rental_details
   ADD CONSTRAINT rental_details_fees_fkv1 FOREIGN KEY ( fees )
      REFERENCES fees ( fee_id );

ALTER TABLE rental_details
   ADD CONSTRAINT rental_details_payment_fk FOREIGN KEY ( payment )
      REFERENCES payment ( pay_id );
```

Script to create triggers:

```
create or replace TRIGGER RENT_EDIT
BEFORE DELETE OR INSERT OR UPDATE OF FEES,INSURANCE ON
RENTAL_DETAILS
BEGIN
  NULL;
END;
```

Script to create sequences:

CREATE SEQUENCE "FY46IN"."SEQ" MINVALUE 1 MAXVALUE 10 INCREMENT BY 1 START WITH 1 CACHE 9 ORDER CYCLE NOKEEP NOSCALE GLOBAL ;