# Lab Test

`ssh` user account:

```
ssh ozh11@linux.cosc.canterbury.ac.nz
```

Run `sqlplus`:

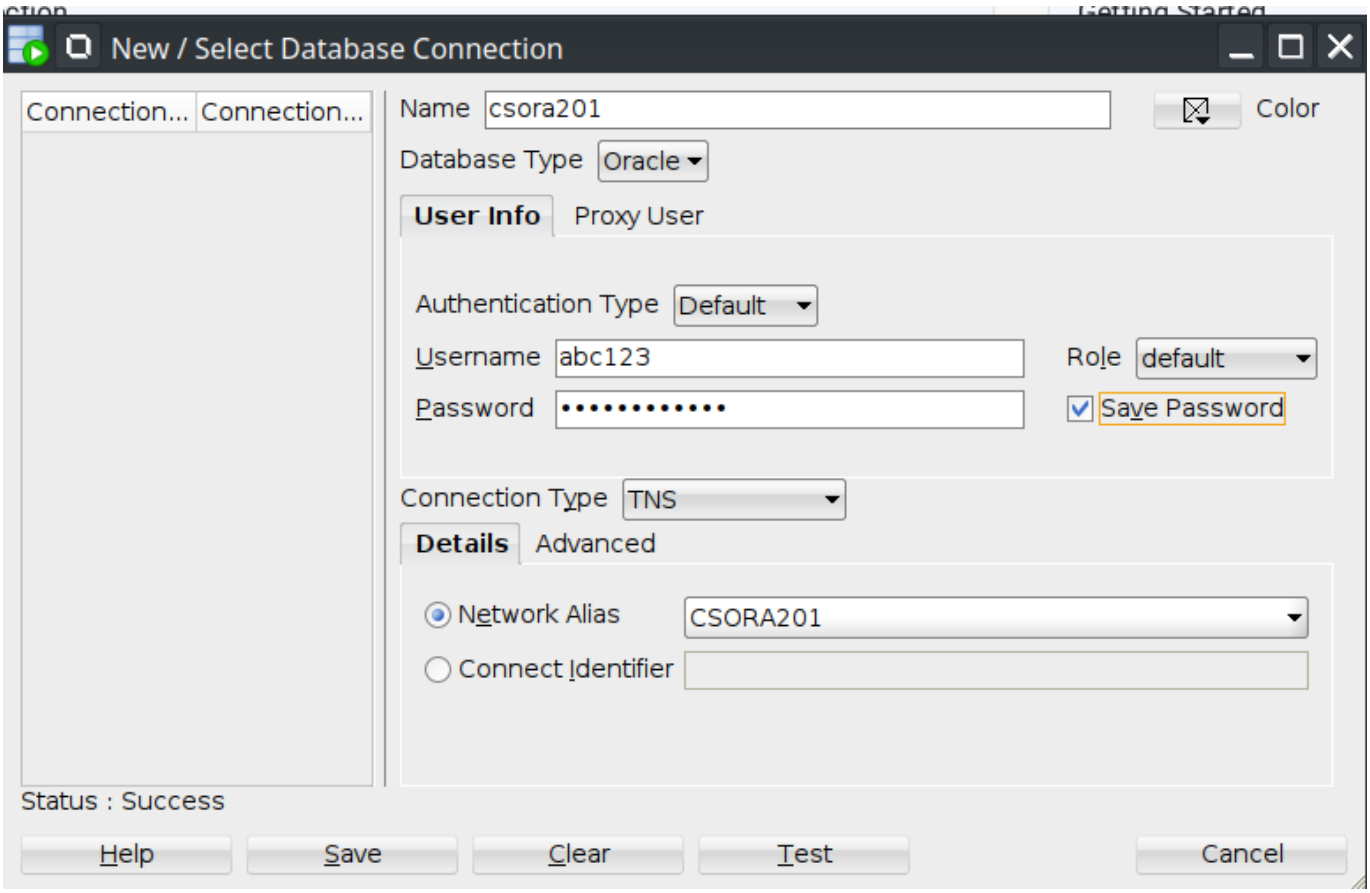
```
sqlplus
username: ozh11@csora201
```

Access CSSE User Account:

- `coschomedir`

Configure SQL Developer:

- Launch SQL Developer
- Select the green + button to add a new connection
- Fill in the following values:
    - Connection name: csora201
    - Username: ozh11
    - Password: (initially student number)
    - Check Save Password
    - Connection Type: TNS
    - Network Alias: CSORA201




- Click Test, then save the connection. Connection csora201 will be added to the Connections panel on the left of the screen.

## Table Commands

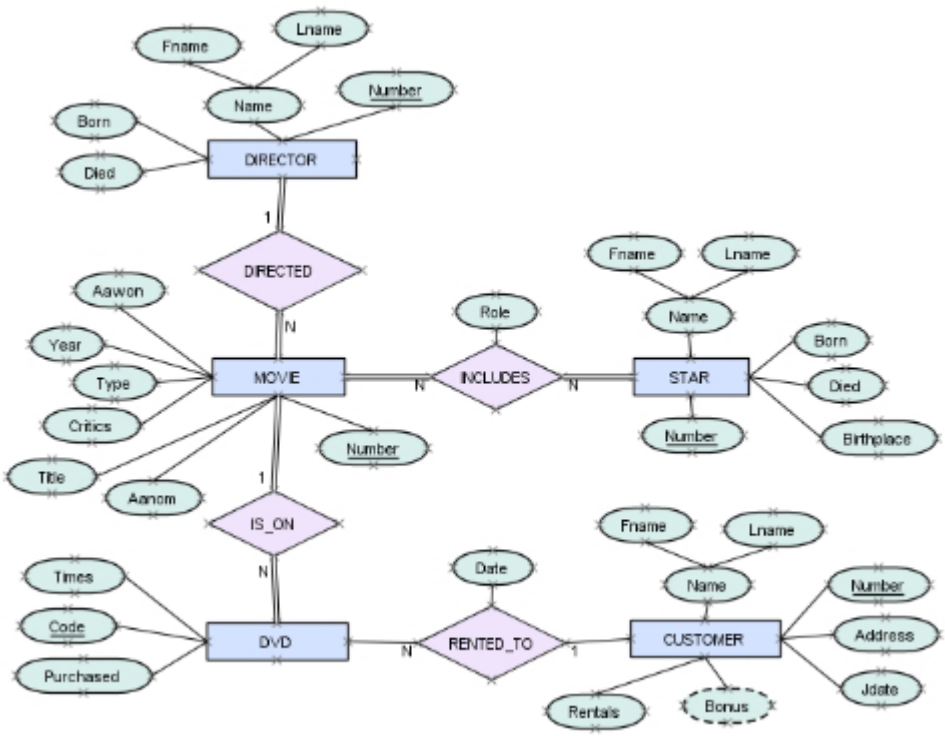
- Describe: `describe tanja.employee`
- View: `select * from tanja.employee`
- Delete: `drop table dvd cascade constaints`
- Create: see below

## MOVIES database

```
drop table dvd;
drop table customer;
drop table stars;
drop table star;
drop table movie;
drop table director;

CREATE TABLE DIRECTOR
(DNUMBER /* Unique number */ INTEGER NOT NULL constraint dir_pk PRIMARY KEY,
LNAME /* Last name */ VARCHAR(16) NOT NULL,
FNAME /* First name */ VARCHAR(15),
BORN /* Year of birth */ INTEGER
                constraint dir_born check (BORN between 1880 and 1990),
DIED /* Year of death */ INTEGER constraint dir_died check (DIED>1930),
constraint corr_years check (born <= died));

CREATE TABLE MOVIE
(MNUMBER /* Unique number for a movie */ INTEGER NOT NULL PRIMARY KEY,
TITLE /* Title */ VARCHAR(50) not null,
TYPE /* Type of the movie */ VARCHAR(15) NOT NULL,
AANOM /* Number of nominations for Academy Awards */ INTEGER,
AAWON /* Number of AA won */ INTEGER,
YEAR /* Year when the movie was made */ INTEGER,
CRITICS /* Critics' rating */ VARCHAR(2),
DIRECTOR /* Director's number */ INTEGER REFERENCES DIRECTOR);

CREATE TABLE STAR
(SNUMBER /* Unique number */ INTEGER NOT NULL,
LNAME /* Last name */ VARCHAR(15) NOT NULL,
FNAME /* First name */ VARCHAR(15),
BORN /* Year of birth */ INTEGER
                constraint check_born check (BORN between 1880 and 2000),
DIED /* Year of death */ INTEGER constraint check_died check (DIED>1930),
CITY /* City of birth */ VARCHAR(15),
constraint corr_syears check (born <= died),
PRIMARY KEY (SNUMBER));

CREATE TABLE CUSTOMER
(LNAME /* Last name */ VARCHAR(15) NOT NULL,
FNAME /* First name */ VARCHAR(15) NOT NULL,
CNUMBER /* Unique number */ INTEGER NOT NULL,
ADDRESS /* Customer's address */ VARCHAR(40),
RENTALS /* The number of DVDs rented */ INTEGER check (rentals>=0),
BONUS /* 1/10 of RENTALS */ INTEGER,
JDATE /* Date of joining the club */ DATE,
PRIMARY KEY (CNUMBER));

CREATE TABLE DVD
(CODE /* Unique number */ INTEGER NOT NULL,
MOVIE /* Movie number */ INTEGER NOT NULL,
```

```
PDATE /* Purchase date */ DATE,
TIMES /* Times rented */ INTEGER default 0,
CUSTOMER /* Number of the customer renting the DVD */ INTEGER,
HIREDATE /* Date of hire */ DATE,
PRIMARY KEY (CODE),
FOREIGN KEY (CUSTOMER) REFERENCES CUSTOMER(CNUMBER),
FOREIGN KEY (MOVIE) REFERENCES MOVIE,
CONSTRAINT CHECK_TIMES CHECK (TIMES >= 0));

CREATE TABLE STARS
(MOVIE /* Movie number */ INTEGER NOT NULL REFERENCES MOVIE(MNUMBER),
STAR /* Star number */ INTEGER NOT NULL REFERENCES STAR(SNUMBER),
ROLE /* Name of the role */ VARCHAR(20) NOT NULL,
PRIMARY KEY (MOVIE,STAR,ROLE));
```

- DIRECTOR (NUMBER, LNAME, FNAME, BORN, DIED)
- MOVIE (NUMBER, TITLE, TYPE, AANOM, AAWON, CRITICS, YEAR, DIRECTOR)
- CUSTOMER (NUMBER, FNAME, LNAME, ADDRESS, RENTALS, BONUS, JDATE)
- STAR (NUMBER, LNAME, FNAME, CITY, BORN, DIED)
- DVD (CODE, PDATE, MOVIE, TIMES, CUSTOMER, HIREDATE)
- STARS (MOVIE, STAR, ROLE)



# REGISTRATION database

```
create table VEHICLE_TYPE
(make /* Make of a vehicle */ varchar(10) not null,
model /* Model of a vehicle */ varchar(10) not null,
power /* Motive power (petrol, gas, diesel) */ char(1)
      constraint check_power check (power in ('p','g','d')),
no_pass /* Number of passengers */ integer
      constraint check_pass check (no_pass between 0 and 8),
cap /* Capacity */ float
      constraint check_cap check (cap >= 0),
cc /* Volume of the motor */ integer
      constraint check_cc check (cc >= 0),
primary key (make,model));

create table OWNER
(dr_lic /* Driver's licence number */ char(8) not null primary key,
IRD /* IRD number of the owner */ char(8),
fname /* Owner's first name */ varchar(15) not null,
init /* Middle initial */ char(1),
lname /* Owner's last name */ varchar(15) not null,
address /* Owner's address */ varchar(50) not null,
bdate /* Owner's birthdate */ date,
gender /* Owner's gender */ char(1),
employer varchar(30),
phone /* Owner's phone number */ varchar(15));

create table EMPLOYEE
(fname /* Employee's first name */ varchar(15) not null,
init /* Employee's middle initial */ char(1),
lname /* Employee's last name */ varchar(15) not null,
IRD /* Employee's IRD number */ varchar(10) not null primary key,
gender /* Employee's gender */ char(1)
  constraint check_gender check (gender in      ('f','m','F','M','x','X')),
bdate /* Employee's birthdate */ date,
office /* Employee's office */ varchar(5),
reg_org /* The number of the registration office the employee works for */
        varchar(10),
sdate /* Starting date in the organization */ date);

create table REG_ORG
(org_number /* The number of the registration organization */ varchar(10)
            not null primary key,
street /* Street name */ varchar(25) not null,
st_num /* Number in the street */ varchar(6) not null,
city /* City */ varchar(15) not null,
manager /* The manager's IRD number */ varchar(10) references employee);
```

```
alter table EMPLOYEE
add foreign key (reg_org) references REG_ORG;
```

```
drop table registration cascade constraints;
drop table color cascade constraints;
drop table owns cascade constraints;
drop table vehicle cascade constraints;

create table VEHICLE
(plates /* Plate number */ varchar(6) not null primary key,
year /* Year of manufacture */ char(4)
      constraint check_year check (year between '1900' and '2019'),
eng_no /* Engine number */ varchar(9) not null unique,
ch_no /* Chassis number */ varchar(7) not null unique,
type /* Type of the vehicle (taxi, private, truck, ...) */ char(1)
      constraint check_veh_type check (type in ('p','m','t','r','l')),
make /* Make of a vehicle */ varchar(10),
model /* Model of a vehicle */ varchar(10),
foreign key (make,model) references vehicle_type);

create table OWNS
(plates /* Owner's plates number */ varchar(6) not null references vehicle,
ownerid /* Owner's id number */ char(8) not null references owner,
purchase_date /* The date of purchase */ date,
drr /* The mileage */ char(6),
DateSold /* The date the vehicle was sold */ date default null,
primary key (plates,ownerid));

create table COLOR
(plates /* The plate number */ varchar(6) not null references vehicle,
color /* Color of the vehicle */ varchar(10) not null,
primary key (plates,color));

create table REGISTRATION
(plates /* Plates */ varchar(6) not null references vehicle,
emp /* IRD of the employee who registered the vehicle */
varchar(10) not null references employee,
reg_org /* Organization number */ varchar(10) not null references reg_org,
reg_date /* Registration date */ date not null,
country /* The country */ varchar(10),
drr /* mileage */ char(6),
amount /* the price */ number,
primary key (plates,emp,reg_org,reg_date));
```

- VEHICLE_TYPE (make, model, power, no_pass, cap, cc)
- OWNER (Dr_Lic, IRD, fname, init, lname, address, bdate, gender, employer, phone)

- EMPLOYEE (fname, init, lname, IRD, gender, bdate, office, reg_org, SDate)
- REG_ORG (number, street, street_no, city, manager)
- VEHICLE (plates, year, Eng_No, Ch_No, type, make, model)
- REGISTRATION (plates, emp, reg_org, reg_date, country, DRR, amount)
- OWNS (plates, ownerid, purchase_date, DRR, DateSold)
- COLOR (plates, color)

## Populate tables

Given `registration-populate.sql`:

```
insert into owner values('BA789256',58743344,'Anna','B','Simmons',
'21 Aorangi Rd Christchurch','21-jan-
58','f',null,'3381275');

/* Inserts for the Vehicle_type table */
insert into vehicle_type (make, model, power, no_pass, cap, cc)
values ('daihatsu', 'charade', 'p', 4, 1, 1000);
insert into vehicle_type (make, model, power, no_pass, cap, cc)
values ('mazda', '121', 'p', 5, 1.2, 1000);
insert into vehicle_type (make, model, power, no_pass, cap, cc)
values ('ford', 'telstar', 'p', 5, 2, 2500);
```

- Populate table: `@path/to/populate-tables`
- Delete all data: `delete from table_name;`

If blank tuple: remove all data from table and copy paste the insert statements directly.

## Commands

- List all tables: `select table_name from user_tables;`
- List privileges: `select * from user_role_privs;`
- List all constraints on a table: `select * from user_constraints where table_name = 'NAMEOFTABLE';`

## Queries

1. Find all types of movies in the database.
   *select distinct type from movie;*
   $\pi_{Type}$(MOVIE)
2. Find all the information about the star whose number is 4.
   *select * from star where snumber = 4;*
   $\sigma_{Number=4}$ (STAR)
3. Find the name, year and city of birth of the star whose number is 50.
   *select fname, lname, born, city*
   *from star*
   *where snumber=50;*
   $\pi_{FName,LName,Born,City}$ ($\sigma_{Number=50}$ (STAR))
4. List the names of all stars born in or after 1950.
   *select fname, lname*
   *from star*
   *where born >= 1950;*
   $\pi_{FName,LName}$ ($\sigma_{Born>=1950}$ (STAR))
5. List the numbers and titles of all movies made between 1965 and 1975.
   *select mnumber, title*
   *from movie*
   *where year between 1965 and 1975;*
   $\pi_{Number,Title}$ ($\sigma_{Year>=1965\ AND\ Year<=1975}$ (MOVIE))
6. List the numbers and titles of all movies whose type is fantasy or romance.
   *select mnumber,title*
   *from movie*
   *where type = 'fantasy' or type='romance';*
   $\pi_{Number,Title}$ ($\sigma_{Type='fantasy'\ OR\ Type='romance'}$ (MOVIE))
7. Find the name, year and city of birth for every star born in 1920s who is still living.
   *select fname, lname, born, city*
   *from star*
   *where born between 1920 and 1929 and died is null;*
   $\pi_{Fname,Lname,Born,City}$ ($\sigma_{Born>=1920\ AND\ Born<=1929\ AND\ Died=NULL}$ (STAR))

8. Produce a list of numbers of all stars that acted in movies number 85 to 91.
   *select distinct star*
   *from stars*
   *where movie between 85 and 91;*
   $\pi_{Star}$ ($\sigma_{Movie>=85\ AND\ Movie<=91}$ (STARS))
9. For all directors who are deceased, list their names and how long they lived.
   *select fname, lname, died-born*
   *from director*
   *where died is not null;*
   $\pi_{Fname,\ Lname,\ died\text{-}born}$ ($\sigma_{Died \neq Null}$ (DIRECTOR))
10. Find the total number of awards won by comedies.
    *select sum(aawon)*

    *from movie*
    *where type='comedy';*
11. $\mathcal{F}_{SUM\ AAWON}$ ( $\sigma_{Type='comedy'}$ (MOVIE))List the titles of all movies and the names of their directors.
    *select title, fname, lname*
    *from movie, director*
    *where director=dnumber;*
    $\pi_{Title,FName,LName}$ (MOVIE $\bowtie_{Director=Director.Number}$ DIRECTOR)
12. Find the name of the star who played Vronsky in the movie entitled 'Anna Karenina'.
    *select fname, lname*
    *from star, stars, movie*
    *where title='Anna Karenina' and role='Vronsky' and movie=mnumber and snumber=star;*
    $\pi_{FName,LName}$ ((($\sigma_{Title='Anna\ Karenina'}$ (MOVIE)) $\bowtie_{Movie=Movie.Number}$ ($\sigma_{Role='Vronsky'}$ (STARS))
    $\bowtie_{Star.Number=Star}$ STAR)

1. Find the different types of vehicle in the database.
   *select distinct type from vehicle;*
   $\pi_{Type}$(VEHICLE)
2. Get plate numbers, makes and models of all cars imported from Japan.
   *select vehicle.plates, make, model*
   *from registration, vehicle*
   *where registration.plates = vehicle.plates and country='Japan';*
   $\pi_{Plates,Make,Model}$ (($\sigma_{Country='Japan'}$ (REGISTRATION)) * VEHICLE)
3. Produce a list of all vehicles, showing only the plate numbers and the year of manufacture. Order the tuples by the year.
   *select plates, year*
   *from vehicle*
   *order by year;*
   $\pi_{Plates,Year}$(VEHICLE)
4. List the names of all owners. Sort the output by last name descending and by first name ascending.
   *select fname, lname*
   *from owner*
   *order by lname desc, fname;*
   $\pi_{Lname,Fname}$(OWNER)
5. For each car, show the plates number and the name of the current owner.
   *select plates, fname, lname*
   *from owner join owns on ownerid=dr_lic*
   *where datesold is null;*

   $\pi_{Plates,Fname,Lname}$ ($\sigma_{Datesold=NULL}$ (OWNS $\bowtie_{Ownerid=Dr\_lic}$ OWNER))

## Complex Queries and Updates

1. Find names of stars who acted in movies directed by Sofia Coppola.

```
select distinct star.fname, star.lname
from star, stars, movie, director
where star=snumber and movie=mnumber and director=dnumber
        and director.fname='Sofia' and director.lname='Coppola';
```

2. Find names of stars who acted in at least two movies directed by Sofia Coppola.

```
select star.fname, star.lname, count(*)
from star, stars, movie, director
where star=snumber and movie=mnumber and director=dnumber
        and director.fname='Sofia' and director.lname='Coppola'
group by star.fname, star.lname, snumber
having count(*)>=2;
```

3. Show types of movies for which there are more than 5 movies in the database. Order the results by decreasing number of movies.

```
select type, count(*) as no_movies
from movie
group by type
having count(*)>5
order by no_movies desc;
```

4. Find the names of all directors who directed at least as many movies as the director number 15.

```
select fname, lname
from director join movie on director=dnumber
group by director, fname, lname
having count(*) >= (select count(*) from movie where director=15);
```

5. Find the director who has directed most dramas.

```
select fname, lname
from director join movie on director = dnumber
where type='drama'
group by director, fname, lname
having count(*) >= all (select count(*) from movie
                        where type='drama' group by director);
```

1. Get full details of all vehicles which were registered during July 2011.

```
select *
from vehicle join  registration R on R.plates = vehicle.plates
where  reg_date between '01-jul-2011' and '31-jul-2011';
```

2. Get the list of vehicles imported from Japan since 1985 which had been registered less than 3 times in New Zealand, listing their plates, makes and models.

```
select vehicle.plates, make, model, count(*) as No_regs
from vehicle, registration R1, registration R2
where R1.plates = vehicle.plates and R1.country = 'Japan' and R1.reg_date>'01-jan-85'
      and R1.plates=R2.plates
group by vehicle.plates, make, model
having count(*)<3;
```

3. Find the names of people who own more than one vehicle.

```
select lname, init, fname
from owner join owns on ownerid=dr_lic
where datesold is null
group by dr_lic ,lname, init, fname
having 1<count(*);
```

4. Write SQL statements to update the REGISTRATION database in the following cases:
   a. Anna Simmons has had her VW golf painted in green.

```
update color
set color='green'
where plates=(select owns.plates
from owns, owner, vehicle
where owns.plates = vehicle.plates and
ownerid=dr_lic and
lname='Simmons' and
fname='Anna' and
make='VW' and model='golf');
```

   b. Write the INSERT statement to add a new registration for the car with plates number TX9283. The car was registered on July 1, 2011 by employee 21321322 who works for registration organization 1352. The DRR reading on the day was 169654 kilometres, and the cost was $137.85.

```
insert into registration
values ('TX9283', 21321322,1352,'01-jul-2011', null,169654,137.85);
```

   c. Delete all registrations for vehicle TX9283.

```
delete from registration
where plates = 'TX9283';
```

5. Create a view EMPS which contains the names and birthdates of all people employed in registration organizations. Try to change the birthdate of some employee.

```
create view emps
```

```
as select lname, fname, bdate
from employee
where reg_org is not null;

update emps
set bdate='25-jul-2011'
where fname='Anna' and lname='Simmons';
```

Oracle allows birthdates to be changes. We have seen that in the standard SQL such a view cannot be updated, as the primary key is not included in the view.

6. Create a view which contains the plates, make and model of each vehicle, and the number of times it has been registered. Using this view, find vehicles that have been registered at least three times. Try deleting information about a specific vehicle. Discuss what happens with DML statements when performed on views. Check what happens both to the view and the base table(s) on which the view is defined.

```
create view veh_info
as select vehicle.plates, make, model, count(*) as regno
        from vehicle join registration on registration.plates=vehicle.plates
        group by vehicle.plates, make, model;

select plates
from veh_info
where regno>3;

delete from veh_info
where plates='FO2341';
```

This view cannot be updated, as it is defined using an aggregate function.

7. Create an index IYEAR on the YEAR attribute of the VEHICLE table. Consider other candidates for indexing.

```
create index IYEAR
on vehicle(year);
```

## Constraints and Triggers

1. Design two attribute and two table constraints for the REGISTRATION database. For example, you may define a constraint on the REGISTRATION table which specifies that the cost of registration has to be over $50. Alter the table(s) and demonstrate the operation of constraints you created.
2. Create the OWNER2 table, which contains the id and name of each car owner, and the total number of vehicles owned by him/her.
   a. Define a trigger that will modify the OWNER2 table in the case when a new owner is added for a car (i.e. insertion to the OWNS table).
   b. Demonstrate the operation of your trigger on a few examples.
3. Create a multitable view MULTIREG, which contains the number of a registration organization, the name of its manager and the number of employees working in it. To do this, first create a copy of the *tanja.reg_org* table, as follows:

   ```
   create table reg_org as select * from tanja.reg_org;
   ```

   Then use your REG_ORG and EMPLOYEE tables to create the MULTIREG view.
   a. Is it possible to update the MULTIREG view directly? Write an UPDATE on the view for the case when John Right becomes the manager of the registration organization number 1303.
   b. Define a trigger that will update all the underlying tables in the case of an UPDATE statement being run on the view that changes the name of the manager of the registration organization whose number is given. You may assume that the data about the new manager already exists in the EMPLOYEE table.
   c. Demonstrate the operation of your trigger when John Right becomes the manager of the registration organization number 1303.

1. Attribute constraints are constraints that are defined within the CREATE TABLE statement. An attribute constraint is defined in the same line as the attribute itself. For example, in the solutions for lab 1 there is a constraint *check_type*, which is an attribute constraint. An attribute constraint is specified on a single attribute only.
   If a constraint involves two attributes from the same table, it must be defined as a table constraint. Such a constraint can be defined within CREATE TABLE, after all attributes have been defined. See the CREATE TABLE statement given in the lecture handout for the DIRECTOR table; there are two table constraints, *dir_died* and *corr_years*. In the solutions for Lab1, the definitions of primary keys for the OWNS, COLOR and REGISTRATION tables are also examples of table constraints.

   If a constraint is based on a single attribute, it can be defined either as an attribute or a table constraint. However, if the constraint is defined on two attributes from the same table, it must be defined as a table constraint. Please note that in Oracle it is not possible to define a constraint which uses attributes from more than one table (in that case, it is necessary to use a trigger).

   Table constraints can also be added via ALTER TABLE – for example:

   *alter table vehicle_type*
   *add constraint constr_example check (no_pass between 0 and 6);*

2. The following statement creates the OWNER2 table:

```
create table owner2 as
select dr_lic, lname, fname, count(*) as no_cars
from owns join owner on ownerid=dr_lic
where datesold is null
group by dr_lic,fname,lname;
```

a. When a new owner is added for a car, we need to check whether the owner has some other cars or not.

```
create or replace trigger change_owner2
after insert on owns
for each row
when (new.datesold is null)
declare
  check_tuple integer :=0;
  olname varchar(15);
  ofname varchar(15);
begin
  select count(*) into check_tuple
  from owner2
  where dr_lic=:new.ownerid;
  if check_tuple = 0 then        -- this is a new owner, not appearing in OWNER2 yet


    select lname, fname into olname, ofname  -- find the name of the owner
    from owner
    where dr_lic = :new.ownerid;
    insert into owner2
    values(:new.ownerid,olname,ofname,1);
  else
    update owner2     -- existing owner, add one more car
    set no_cars=no_cars+1
    where :new.ownerid=dr_lic;
  end if;
end;
/
```

b. Whenever an insert is run on OWNS, the OWNER2 table is modified accordingly. Please see the example below.

```
SQL> select * from owner2;

DR_LIC   LNAME           FNAME              NO_CARS
-------- --------------- --------------- ----------
DB125699 Martin          Jennie                   1
BA789256 Simmons         Anna                     1
HD543235 Jason           King                     2
HD293847 Lin             Mary                     1
GR153856 Roberts         Steven                   1
FF849583 Austin          Jane                     2
IA192837 Mouse           Minnie                   2
JA264818 Holland         Peter                    1

SQL> insert into owns
  2  values ('PA9485','HD543235','25-jul-2010',58920,null);

1 row created.

SQL> select * from owner2;

DR_LIC   LNAME           FNAME              NO_CARS
-------- --------------- --------------- ----------
DB125699 Martin          Jennie                   1
BA789256 Simmons         Anna                     1
HD543235 Jason           King                     3
HD293847 Lin             Mary                     1
GR153856 Roberts         Steven                   1
FF849583 Austin          Jane                     2
IA192837 Mouse           Minnie                   2
JA264818 Holland         Peter                    1

8 rows selected.
```

3. create view multireg
as select org_number, M.lname, M.fname, count(*) as total_emp
from reg_org, employee E, employee M
where manager=M.ird and E.reg_org=org_number
group by org_number, M.lname, M.fname;

a. It is not possible to update the view directly. In the case of the update statement given below, Oracle returns an error because the view is not updatable.

```
update multireg
set lname='Right', fname='John'
where org_number='1303';
```

b. The update_view trigger to change the manager:

```
create trigger update_view
instead of update on multireg
for each row
declare
  managerno char(8);
begin
  select ird into managerno
  from employee
  where lname=:new.lname and fname=:new.fname;

  update reg_org
  set manager=managerno
  where org_number=:new.org_number;
end;
/
```

c. When the same UPDATE is executed with the existing trigger, the changes are made to the underlying tables. The following is an excerpt from the SQL Plus session:

```
SQL> select * from multireg;

ORG_NUMBER LNAME           FNAME             TOTAL_EMP
---------- --------------- --------------- ----------
1303       Tay             Angela                  2
1352       Simmons         Anna                    3

SQL> create trigger update_view
  2  instead of update on multireg
  3  for each row
  4  declare
  5    managerno char(8);
  6  begin
  7    select ird into managerno
  8    from employee
  9    where lname=:new.lname and fname=:new.fname;
 10
 11    update reg_org
 12    set manager=managerno
 13    where org_number=:new.org_number;
 14  end;
 15  /

Trigger created.

SQL> update multireg
  2  set lname='Right', fname='John'
  3  where org_number='1303';

1 row updated.

SQL> select * from multireg;

ORG_NUMBER LNAME           FNAME             TOTAL_EMP
---------- --------------- --------------- ----------
1352       Simmons         Anna                    3
1303       Right           John                    2
```

## Test

- The ***Patient*** relation stores information about patients, such as the unique patient number, first and last name, email address, birthdate, phone number, and the account balance.

  PATIENT (PNum, FName, LName, BDate, Street, City, Email, Phone, Balance)

- The ***Therapist*** relation tracks information about each therapist, including a unique id, name, and address.

  THERAPIST (UserID, FName, LName, Address)

- The ***Therapy*** relation stores a unique number for each therapy, description, and the billable units of time (in minutes).

  THERAPY (Code, Description, Duration)

```
create table PATIENT
(
  PNum integer not null primary key,
  FName varchar(15) not null,
  LName varchar(15) not null,
  email varchar(20) not null,
  BDate date not null,
  Phone varchar(15) not null,
  Balance number not null
);

create table THERAPY
(
  Code integer not null primary key,
  Description varchar(90) not null,
  Duration integer
);

create table THERAPIST
(
  TNum char(5) not null primary key,
  FName varchar(15) not null,
  LName varchar(15) not null,
  Address varchar(30) not null
);
```

**Question 1 (20 marks for the whole question)**

a) Create the **Appointment** relation, which stores information about the sessions patients have had, or have booked. For each session, the table stores a unique session number, the date, start time, patient number and the therapist number.

```
create table appointment
(SessionNo integer not null primary key,
 Patient integer not null references Patient,
 Therapist char(5) not null references Therapist,
 SDate date not null,
 Stime char(5) not null);
```

b) Populate the table you created with data provided in the *appointment-inserts.sql* file. Write an SQL statement that shows the number of tuples in your table.

```
select count(*) from Appointment;
Returns 90 tuples.
```

c) Create the **Included** relation, which stores information about the different therapies planned for or conducted during a session. A session can contain several therapies, but a therapy can only appear once in a session.

```
create table INCLUDED
(SessionNo integer not null references Appointment,
 Therapy integer not null references Therapy,
 Primary key (SessionNo, Therapy));
```

d) Populate the table you created with data provided in the *scheduled-inserts.sql* file. Write an SQL statement that shows the number of tuples in your table.

```
select count(*) from Included;
Returns 189 tuples.
```

**Question 2 (40 marks for the whole question) Write a single SQL statement for each of th following situations:**

a) Show the code and description of the most frequently used therapy, excluding Assessment (Code = 1).

```
Select code, description
From therapy join included on therapy = code
Where therapy <> 1
Group by code, description
Having count(*) >= ALL(select count(code)
                       From therapy join included on code = therapy
                       Where therapy!=1
                       Group by code);
```

(Answer: 9, Massage)

b) How many patients are receiving the infrared therapy?

```
Select count(distinct patient)
From therapy join included on therapy = code join appointment
     On Appointment.SessionNo = Included.SessionNo
Where description = 'Infrared';
```

(Answer: 3 patients)

c) How many therapies refer to pain in their description?

```
Select count(description)
From therapy
Where description like '%pain%';
```

(Answer: 0)

d) Modify the database to enforce a condition that a patient can only have one session on a given day. Show that your modification works.

This can be achieved by having a constraint on the Appointment table, or by defining a trigger. The following INSERT statement can be used to show that the modification works:

```
insert into Appointment
values(91,156465,'JR085','6/10/2022','14:00');

ALTER TABLE Appointment add constraint one_PerDay
unique(Patient,SDate);

create or replace trigger One_PerDay
BEFORE INSERT on Appointment
for each row
declare No_Sessions integer;
  begin
     select count(*) into No_Sessions
```

```
from appointment
where Patient = :new.Patient and SDate = :new.SDate;
if (No_Sessions > 0) then
      raise_application_error (num => -20009,
      msg => 'There is already a session on that date!');
end if;
end;
/
```

**Question 3 (30 marks for the whole question) Write a single SQL statement for each of the following situations:**

a) Create a view which will contain the following information about all appointments planned for today: the names of the patient and the therapist, start time, and the appointment length. Order the tuples by the start time. The query should be able to be executed on any day.

```
Create view SCHEDULE as
Select P.Fname, P.Lname, TNum, Stime, sum(Duration) as Duration
From appointment A, patient P, therapist T, included, therapy
Where A.SessionNo = included.SessionNo and
      sdate = (select sysdate from dual) and
      therapist = TNum and A.patient = PNum and
      Code = therapy
Group by P.Fname, P.Lname, T.FName,T.LName,Stime
Order by stime;

24 rows
```

b) Modify the database to enforce a condition that the total length of a session cannot be longer than 90 minutes. Provide an example to show that your modification works.

```
create or replace trigger Add_Therapy
BEFORE INSERT on INCLUDED
for each row

declare
   Existing_Length integer := 0;
   Therapy_Length integer :=0;

begin
   /* get the duration of the session before insert */
   select sum(Duration) into Existing_Length
   from therapy join included on included.therapy = code
   where SessionNo = :new.SessionNo;
/* get the duration of the new therapy */
   select Duration into Therapy_Length
   from therapy
   where Code = :new.therapy;

If ((Existing_Length + Therapy_Length) > 90) then
    raise_application_error (num => -20099,
```

```
                msg => 'Sessions cannot be longer than 90 minutes!');

        end if;
      end;
      /
```

**Example**: The current length of session 67 is 65 minutes. Therapy 21 is 30 minutes long. If we have:

```
Insert into included
Values(67,21);
```

```
SQL> Insert into included
  2  Values(67,21);
Insert into included
       *
ERROR at line 1:
ORA-20099: Sessions cannot be longer than 90 minutes!
ORA-06512: at "TANJA.ADD_THERAPY", line 16
ORA-04088: error during execution of trigger 'TANJA.ADD_THERAPY'
```

c) (4 marks) A new public holiday has been declared for October 14. Write a statement to cancel all appointments on that date.

```
SQL > delete from appointment
  2       where sdate = '14/10/2022';
```

**Question 5 (**No marks; a correct answer will be awarded a chocolate fish)
Write a single SELECT statement to show the names of those therapists who perform all kinds of therapies.

```
select fname,lname,count(distinct therapy)
from included join appointment on included.SessionNo = appointment.sessionno
join therapist on therapist = tnum
group by therapist, fname,lname
having count(distinct therapy) = (select count(code) from therapy);
```

Alternative solution:

```
select fname, lname
from therapist
where not exists
       (select code
        from therapy
        where not exists
                    (select *
                     from included join appointment on
                          appointment.sessionno = included.sessionno
                     where therapy.code = Included.therapy
                           and therapist = Tnum));
```