**For every auctioned item that was sold for a price greater than its initial price, show the item name together with the name of its buyer and the price they paid:**

SELECT auction\_item.name, bidder.name, bidder.final\_price

FROM auction\_item inner Join bidder

ON auction\_item.id = bidder.bought\_item\_id

WHERE auction\_item.initial\_price < bidder.final\_price

**Show the**name**,**initial price**, and**final price**of every item with initial price**greater than $300**. If the item wasn't sold (there was no bidding person), show the item's name anyway.**

SELECT auction\_item.name, auction\_item.initial\_price, bidder.final\_price

FROM auction\_item left join bidder

ON auction\_item.id = bidder.bought\_item\_id

WHERE auction\_item.initial\_price > 300

**Show all information only for those items that**weren't**auctioned off**

SELECT \*

FROM auction\_item left join bidder

ON auction\_item.id = bidder.bought\_item\_id

where bidder.bought\_item\_id IS NULL

**List all products that have fewer than 150 calories.**

**For each product show its name (rename the column to product) and the department in which it can be found (name the column department).**

SELECT product.name as product, department.name as department

FROM product JOIN department

ON product.department\_id = department.id

JOIN nutrition\_data

ON nutrition\_data.product\_id = product.id

WHERE nutrition\_data.calories < 150

**For each fruit (found in the fruits department) show the product's shelf id, name, price, and number of calories.**

SELECT product.shelf\_id, product.name, product.price, nutrition\_data.calories

FROM product JOIN nutrition\_data

ON product.id = nutrition\_data.product\_id

WHERE product.department\_id = 1

**Let's make use of our new table straight away. For each product show its name, price, department name, and producer name.**

**Alias the columns as product\_name, product\_price, department\_name, and producer\_name, respectively**

SELECT product.name as product\_name, product.price as product\_price,

department.name as department\_name, producer.name as producer\_name

FROM product join department

ON product.department\_id = department.id

JOIN producer

ON producer.id = product.producer\_id

**For each product display:**

**the name of the company that produced it (alias the column to producer\_name),**

**the name of the department where the product is located (alias it to department\_name),**

**the name of the product (alias it to product\_name),**

**the total number of carbohydrates in the product.**

**Your query should still consider products with no information about producer\_id or department\_id.**

SELECT

prod.name AS producer\_name,d.name AS department\_name,p.name AS product\_name,nd.carbohydrate

FROM product p LEFT JOIN producer prod ON prod.id=p.producer\_id

LEFT JOIN department d ON d.id=p.department\_id

LEFT JOIN nutrition\_data nd ON nd.product\_id=p.id

**For every day in the sales\_history table show the sales of the products with ID's 13, 18, and 15.**

**Display the date, name, amount, and calorific value of a product.**

**Order the results by date and then by calorific value.**

SELECT product.name, sales\_history.date, sales\_history.amount, nutrition\_data.calories

FROM product join sales\_history

on product.id = sales\_history.product\_id

join nutrition\_data

ON sales\_history.product\_id = nutrition\_data.product\_id

where sales\_history.product\_id in ('13', '18', '15')

order by sales\_history.date, nutrition\_data.calories

**For products that weren't sold even once between 2015-02-01 and 2015-02-05, show the product name (rename the column to product\_name), it's price and the producer's name (rename the column to company\_name). You should display all products that haven't been sold in this interval, also the ones that don't belong to any company.**

SELECT product.name as product\_name, product.price, producer.name as company\_name

FROM product left join sales\_history

ON sales\_history.product\_id = product.id and sales\_history.date between '2015-02-01' and '2015-02-05'

left join producer

ON producer.id = product.producer\_id

WHERE sales\_history.product\_id is Null;

**Show the name and price of each product in the 'fruits' and 'vegetables' departments. Consider only those products that are not produced by 'GoodFoods'**

SELECT p.name, p.price

FROM product p

LEFT JOIN producer prod

ON p.producer\_id = prod.id

JOIN department d

ON d.id = p.department\_id

WHERE d.name IN ('fruits', 'vegetables')

AND (prod.name != 'GoodFoods' OR prod.id IS NULL);

**Show only worker's name, and specialization. Alias column with the name as apprentice. Consider only those workers whose supervisors have more than 14 years of experience**.

SELECT w.name as apprentice, w.specialization

FROM workshop\_workers as w join workshop\_workers as s

ON w.master\_id = s.id

WHERE s.experience > 14

**Show the name of the apprentice, their specialization and the ID of the project if the project they are working on is not directly supervised by their master, i.e. the master currently works on another project**

SELECT w.name, w.specialization, w.project\_id

FROM workshop\_workers as w join workshop\_workers as s

ON w.master\_id = s.id

where w.project\_id != s.project\_id; (We are considering project\_Id of worker that is different from project\_id of master)

**Show each dictionary entry (name the column entry) together with the name of its direct hypernym (name the column hypernym) and the name of that hypernym's hypernym (name the column grandhypernym).**

**Only include words that have both a direct hypernym and a "grandhypernym".**

SELECT entry.word as entry, hypernym.word as hypernym, grandhypernym.word as grandhypernym

FROM dictionary as entry join dictionary as hypernym

ON entry.hypernym\_id = hypernym.entry\_id

JOIN dictionary as grandhypernym

ON grandhypernym.entry\_id = hypernym.hypernym\_id

**For each entry in the dictionary show the word, its direct hypernym and its hypernym's hypernym. Name the columns entry, hypernym and grandhypernym, respectively. Show all such entries, even those that don't have direct hypernyms or grandhypernyms.**

SELECT entry.word as entry, hypernym.word as hypernym, grandhypernym.word as grandhypernym

FROM dictionary as entry left join dictionary as hypernym

ON entry.hypernym\_id = hypernym.entry\_id

left join dictionary as grandhypernym

ON grandhypernym.entry\_id = hypernym.hypernym\_id

**List all pairs of full siblings. Full siblings are people who have the same mother and the same father. Name the columns with sibling names younger\_sibling and older\_sibling (use column year\_born to identify who is younger).**

**You can assume that in our database no two siblings were born in the same year and there are no more than two siblings per family.**

SELECT young.name as younger\_sibling, older.name as older\_sibling

FROM person as older join person as young

ON young.mother\_id = older.mother\_id

and young.father\_id = older.father\_id

WHERE young.year\_born > older.year\_born

**We want to know if there are any patients and physicians who have the same email in our clinic. Perhaps physicians also attend our clinic or former patients were hired by the clinic?**

**List the first and last name of each patient (name the columns patient\_first\_name and patient\_last\_name) and the first and last name of each physician (name the columns physician\_first\_name and physician\_last\_name) for patients and physicians who share the same email address.**

SELECT patient.first\_name as patient\_first\_name, patient.last\_name as patient\_last\_name,

physician.first\_name as physician\_first\_name, physician.last\_name as physician\_last\_name

FROM patient join physician

ON patient.email = physician.email

**For each patient and physician matching this condition, show the patient's ID (name the column patient\_id), the ID of a physician (name the column physician\_id), their shared first and last name, and the physician's years of experience. Only list physicians who have more than five years of experience**.

SELECT patient.id as patient\_id, physician.id as physician\_id, patient.first\_name, physician.last\_name, physician.experience

FROM patient join physician

ON patient.email = physician.email

WHERE physician.experience > 5

**For each patient, list the treatment that they have been recommended. Select the ID and first and last name of each patient. Additionally, show the type of treatment that the patient has been recommended (name it type) along with a list**

**of the names of available treatment sessions. Only list treatments that are cheaper than $100.**

SELECT patient.id , patient.first\_name, patient.last\_name,

treatment.name, therapy.treatment\_type as type

FROM patient left join therapy

ON patient.id = therapy.patient\_id

JOIN treatment

ON treatment.type = therapy.treatment\_type

WHERE treatment.price < 100

**Show the names and types of treatments (name this column type) that were recommended by a physician named James Ferris. Additionally, show the IDs of patients (as patient\_id) for whom these recommendations were made. Exclude treatments of the type relaxation**

SELECT name, type, patient\_id

FROM therapy JOIN treatment

ON treatment.type = treatment\_type

JOIN physician ON physician\_id = physician.id

WHERE type != 'relaxation'

AND physician.first\_name = 'James'

AND physician.last\_name = 'Ferris';

**Show the name of every buyer together with the name of every item that they can afford.**

SELECT buyer.name, item.name

FROM item JOIN buyer

ON item.price <= buyer.funds

**For every buyer that has funds greater than 10 000, show the name of the buyer and the names of the items they can afford that are not of the 'painting' type.**

SELECT buyer.name, item.name

FROM item join buyer

on item.price <= buyer.funds

where item.type != 'painting' and buyer.funds > 10000

**For every couple that wants an apartment, show their name together with the IDs of all apartments within their price range**.

SELECT couples.couple\_name, apartments.id

FROM couples join apartments

ON apartments.price between couples.min\_price and couples.max\_price

**For each couple that wants an apartment, show their name with matching apartment IDs and locations for all apartments that are within their price range and in their preferred location**

SELECT couples.couple\_name,apartments.id, apartments.location

FROM couples join apartments

ON apartments.price between couples.min\_price and max\_price

and apartments.location = couples.pref\_location

**For each order placed during the promotional period, show its ID, the ID of the ordered product, the date when the order was placed, and the total amount that was paid for the order (amount \* price). Name the column total\_amount**

SELECT o.ID, o.product\_id, o.order\_date, o.amount \* p.price as total\_amount

FROM orders as o join product\_price as p

ON o.product\_id = p.product\_id

and o.order\_date between start\_date and end\_date

**Show the name of each treatment and the first and last names of the patients to whom it was recommended for all therapies recommended by physicians with the surname Core or Calderwood.**

**Consider only patients with odd ID numbers**

SELECT treatment.name, patient.first\_name, patient.last\_name

FROM patient join therapy

ON patient.id = therapy.patient\_id

JOIN treatment

ON treatment.type = therapy.treatment\_type

Join physician

ON physician.id = therapy.physician\_id

WHERE (physician.last\_name = 'Core' or physician.last\_name = 'Calderwood')

and patient.id % 2 = 1

**Show the name of each couple, their preferred apartment location, and the apartment ID for all apartments that are neither in their price range nor in their preferred location.**

SELECT couples.couple\_name, couples.pref\_location, apartments.id

FROM apartments join couples

ON apartments.price not between couples.min\_price and couples.max\_price

and (couples.pref\_location != apartments.location)

**For each buyer with funds less than 8000, show their name together with the names of all paintings and sculptures that are outside of their price range**

SELECT buyer.name, item.name

FROM buyer join item

ON buyer.funds < item.price

WHERE item.type in ('painting', 'sculpture') and buyer.funds < 8000

**Show the name of each lecturer together with the ID of the group they teach, the language they teach, and the level of the group they teach**.

SELECT lecturer.name, student\_group.id, lecturer.language, student\_group.group\_level

FROM lecturer join student\_group

on lecturer.id = student\_group.lecturer\_id

**Show each group ID, number of members, level, language and tuition cost. Consider only groups with more than 10 people and whose lecturer is not 'Maria Garcia'.**

SELECT student\_group.id, student\_group.person\_no, student\_group.group\_level, lecturer.language, tuition.cost

FROM student\_group join tuition

ON tuition.group\_id = student\_group.id

JOIN lecturer

ON lecturer.id = student\_group.lecturer\_id

WHERE student\_group.person\_no > 10 and lecturer.name != 'Maria Garcia'

**Show each student group's ID (use the group\_id column), room ID, lecture day, and lecture hour for those groups that do not have both lectures in the same room**

SELECT t1.group\_id, t1.room\_id, t1.day, t1.hour

FROM timetable t1 inner join timetable t2

ON t1.room\_id <> t2.room\_id

and t1.group\_id = t2.group\_id

**For each group, show its ID (as group\_id), language, and level, together with the ID of the room they're in and the days when they have lectures. Consider only those groups whose lectures are always held in the same room.**

SELECT student\_group.id as group\_id, student\_group.language, student\_group.group\_level, timetable.room\_id,timetable.day

FROM student\_group join timetable

ON student\_group.id = timetable.group\_id

JOIN timetable as t2

ON student\_group.id = t2.group\_id

and timetable.room\_id = t2.room\_id

and timetable.day <> t2.day

**For all groups taught in the La Lingua school, show the following information:**

**the ID of a given group,**

**the name of the lecturer who teaches that group,**

**the language that the group learns,**

**the level of the group, and**

**the days and hours when the group has lectures.**

**However, show this information only for those groups that have both their lectures at the same hour and whose tuition is greater than 800 dollars**

SELECT student\_group.id, lecturer.name, student\_group.language, student\_group.group\_level, tt.day, tt.hour

FROM student\_group JOIN lecturer ON lecturer\_id = lecturer.id

JOIN timetable tt ON tt.group\_id = student\_group.id

JOIN timetable tt2 ON tt.group\_id = tt2.group\_id

JOIN tuition ON tuition.group\_id = student\_group.id

WHERE tt.day != tt2.day AND tt.hour = tt2.hour

AND tuition.cost > 800

**Show model, brand, mileage and production year from the car table for these cars that were produced after the 'Roadmaster' (id = 2) and whose original price is greater than original\_price of 'Speedster' (id = 1)**

SELECT model, brand, mileage, prod\_year

FROM car

WHERE id > 2 and original\_price > 80000;

* OR

SELECT c1.model, c1.brand, c1.mileage, c1.prod\_year  
FROM car as c1 INNER JOIN car as c2  
ON c1.prod\_year > c2.prod\_year  
AND c1.original\_price > c2.original\_price  
WHERE c2.model = 'Roadmaster'  
AND c1.model != 'Speedster';

**Show the model, brand, and final price of each car sold at the auction. Consider only those sold cars that have more mileage than the car with id = 4**

SELECT model, brand, charity\_auction.final\_price

FROM car inner join charity\_auction

ON car.id = charity\_auction.car\_id

and charity\_auction.final\_price is not null

where mileage > (Select mileage

FROM car

where id = 4)

**Show the model and brand of each car whose initial\_price was in the price range of the buyer with id = 3, but whose final\_price was outside the buyer's price range.**

**Show me the answer**

SELECT model, brand

FROM car join charity\_auction

ON car.id = charity\_auction.car\_id

JOIN buyer

ON buyer.id = charity\_auction.buyer\_id

where buyer.id = 3

and charity\_auction.initial\_price <= buyer.funds

**Show each buyer's ID (rename the column to buyer\_id) together with the model and mileage of the car they bought.**

**If a potential buyer didn't buy any car, show their ID anyway.**

SELECT b.id as buyer\_id , model, mileage

FROM buyer as b left join charity\_auction as c

on b.id = c.buyer\_id

left join car

on car.id = c.car\_id

**Show the ID and funds of each buyer who didn't buy anything at the auction**.

SELECT id, funds

from buyer left join charity\_auction as c

on buyer.id = c.buyer\_id

where c.final\_price is null

**Show all information about all cities which have the same area as Paris**

SELECT \*

FROM city

WHERE area = (SELECT area

FROM city

WHERE name = 'Paris')

**Find the names of all cities which have a population lower than Madrid.**

SELECT name

FROM city

WHERE population < (SELECT population

FROM city

WHERE name = 'Madrid')

**Find all information about trips whose price is higher than the average**

SELECT \*

FROM Trip

WHERE price > (SELECT avg(price)

FROM Trip)

**Find all information about all trips in cities whose area is greater than 100**

SELECT \*

FROM trip

WHERE city\_id in (SELECT id

FROM city

WHERE area > 100)

**Find all information about the cities which are less populated than all countries in the database**

* > ALL means "greater than every other value from the parentheses".

SELECT \*

FROM city

WHERE population < ALL (

SELECT population

FROM country)

**Find all information about all the city trips which have the same price as any hiking trip**

SELECT \*

FROM trip

WHERE price = Any (

SELECT price

FROM hiking\_trip)

**Find all information about cities with a rating higher than the average rating for all cities in that specific country.**

**SELECT \***

FROM city as c1

WHERE rating > (

SELECT avg(rating)

FROM city as c2

where c1.country\_id = c2.country\_id)

**Show all information about all trips to cities which have a rating lower than 4.**

SELECT \*

FROM TRIP

WHERE city\_id in (

SELECT id

FROM city

WHERE rating < 4)

* **The operator EXISTS /NOT EXISTS is a new operator that checks if there are any rows that meet the condition:**

**Select all countries where there is at least one mountain**

SELECT \*

FROM country

WHERE EXISTS (

SELECT \*

FROM MOUNTAIN

WHERE mountain.country\_id = country.id)

**Select all mountains with no hiking trips to them**.

SELECT \*

FROM mountain

WHERE NOT EXISTS (

SELECT \*

FROM hiking\_trip

where mountain.id = hiking\_trip.mountain\_id)

**Select the hiking trip with the longest distance (column length) for every mountain**.

SELECT \*

FROM hiking\_trip hk

WHERE length >= ALL (

SELECT length

FROM hiking\_trip ht

WHERE hk.mountain\_id = ht.mountain\_id)

**Select those trips which last shorter than any hiking\_trip with the same price**

SELECT \*

FROM TRIP

WHERE days < ANY (

SELECT days

FROM hiking\_trip

WHERE trip.price = hiking\_trip.price)

**Show mountains together with their countries. The countries must have at least 50,000 residents.**

SELECT m.name, c.name, c.population

FROM mountain m join country c

WHERE m.country\_id = c.id and c.population >= 50000

OR:

SELECT \*

FROM mountain, (SELECT \*

FROM country

WHERE population >= 50000) as country

WHERE mountain.country\_id = country.id

**Show hiking trips together with their mountains. The mountains must be at least 3,000 high. Select only the columns length and height**

SELECT length, height

FROM hiking\_trip, (SELECT \*

FROM mountain

WHERE height >= 3000) as mountain

WHERE hiking\_trip.mountain\_id = mountain.id

* **Subquery can also be used withing the col list in a SELECT clause, it is important that the subquery returns exactly one row and column**:

**Show each mountain name together with the number of hiking trips to that mountain (name the column count)**

SELECT name, (SELECT count(\*)

FROM hiking\_trip

WHERE mountain.id = hiking\_trip.mountain\_id) as count

FROM mountain

**Show all the medals for the period between 2010 and 2014 for skating and cycling. Use the UNION keyword**

SELECT \*

FROM skating

WHERE year between 2010 and 2014

UNION

SELECT \*

FROM cycling

WHERE year between 2010 and 2014

* **UNION returns result from 1st and 2nd query and removes duplicate while UNION ALL does the same but keeps the duplicate rows**
* **INTERSECT shows the rows that belong to BOTH tables**

**Find names of each person who has medals both in cycling and in skating**

SELECT person

FROM cycling

INTERSECT

SELECT person

FROM skating

* EXCEPT : shows all the results from first (left) table with the exception of those that also appeared in the second(right) table(it returns the rows from the first table MINUS the rows from the second table).

**Find all the countries which have a medal in cycling but not in skating :**

The below query will see all country who have medal in cycling **MINUS** all country who medal in skating

SELECT country

FROM cycling

EXCEPT

SELECT country

FROM skating

**Find all the years when there was at least one medal in skating but no medals in cycling. Use the keyword MINUS**

SELECT year

FROM skating

MINUS

SELECT year

FROM cycling

**Select all columns from horoscopes for Pisces and Aquarius from the years 2010 to** 2014.

SELECT \*

FROM horoscope

WHERE sign in ('Aquarius' , 'Pisces') and year between 2010 and 2014

**Show all pets (show the columns name, type, year\_born) whose name begins with an 'M' together with their owners (the columns name, year\_born).**

**Rename the column year\_born from the table pet as pet\_year\_born and the column year\_born from the table owner as owner\_year\_born**

SELECT p.name, p.type, p.year\_born as pet\_year\_born, O.name, O.year\_born as owner\_year\_born

FROM pet as p join OWNER as O

ON p.owner\_id = O.id

where p.name LIKE 'M%'

**Show students' names (column person) together with**

**the number of essays they handed in (name the column number\_of\_essays).**

**their average number of points (name the column avg\_points).**

**Show only those students whose average number of points is more than 80**

SELECT person, count(\*) as number\_of\_essays, avg(points) as avg\_points

FROM essay

group by person

having avg(points)> 80

**Show all coaches together with the players they train, show all columns for coaches and players. Show unemployed coaches with NULLs instead of player data**.

SELECT \*

FROM coach left join player

on player.id = coach.player\_id

**Show all columns for the prisons where there is at least one prisoner above 50 years of age.**

SELECT \*

FROM prison

WHERE id in (SELECT prison\_id

FROM prisoner

WHERE age >50)

**Show all columns for the products which are gluten free and vegetarian at the same time**

SELECT \*

FROM gluten\_free\_product

INTERSECT

SELECT \*

FROM VEGETARIAN\_PRODUCT

**The owner of the shop would like to see each customer's**

**id (name the column cus\_id).**

**name (name the column cus\_name).**

**id of their latest purchase (name the column latest\_purchase\_id).**

**the total quantity of all flowers purchased by the customer, in all purchases, not just the last purchase (name the column all\_items\_purchased).**

**Remember, you need not use all columns from all the tables here – choose them carefully.**

SELECT c.id as cus\_id, c.name as cus\_name, max(p.id) as latest\_purchase\_id,sum(pu.quantity) as all\_items\_purchased

FROM customer as c join purchase as p

ON c.id = p.customer\_id

join purchase\_item as pu

ON p.id = pu.purchase\_id

group by c.id

------------------------------PRACTICE:

**For each painting created in 1800 or later, show its title and the year it was painted (column painted). Include paintings for which the creation date is unknown**

SELECT title, painted

FROM painting

WHERE painted >=1800 or (painted is null)

**For each painting, show its title plus the first and last name of the painter. Only show the results for Dutch and Flemish painters.**

SELECT p.title, a.first\_name, a.last\_name

FROM artist as a join painting as p

ON a.id = p.artist\_id

WHERE a.nationality in('Dutch' , 'Flemish')

**Show the number of paintings whose date of creation is known and which were created in 1800 or later. Name the column paintings\_no**

SELECT count(id) as paintings\_no

FROM painting

WHERE painted >=1800 and (painted is not null)

**For each painter, show their first and last name together with the number of paintings they have painted. The names of the columns should be: first\_name, last\_name, and paintings\_no**

SELECT a.first\_name, a.last\_name, count(p.id) as paintings\_no

FROM artist as a join painting as p

ON a.id = p.artist\_id

group by a.first\_name, a.last\_name

**For each artist, show their first and last name, the average, minimum and maximum ratings from all their paintings. Only show those artists who have more than 2 paintings**

SELECT a.first\_name, a.last\_name, avg(p.rating) as avg\_rating, min(p.rating) as min\_rating, max(p.rating) as max\_rating

FROM artist as a join painting as p

on a.id = p.artist\_id

group by a.first\_name, a.last\_name

having count(p.id)>2

**For each item, show the following sentence: 'ID X is Y.', where X is the id of the item and Y is its name. Name the column sentence**

SELECT 'ID ' || id || ' is ' || name || '.' as sentence

FROM item

OR

SELECT concat('ID ', id ,' is ', name, '.') as sentence

FROM item

**Show the IDs of all the items with a name longer than 8 characters. Show the length as the second column, name it name\_length**

SELECT ID, length(name) as name\_length

FROM item

where length(name) >8

**For each slogan, show the item name and the slogan with all the periods (.) replaced by exclamation marks (!). Name the second column changed\_text**.

SELECT i.name, replace(s.text, '.','!') as changed\_text

FROM slogan as s join item as i

ON i.id = s.item\_id

**For each slogan with a text longer than 20 characters, show the text fragment from character 5 until character 20. Name the column text\_substring**

SELECT substring(text, 5,16) as text\_substring

FROM slogan

WHERE length(text) > 20

**For each 'tv commercial' slogan, show the item name, the item type, and the text with each period (.) turned into three exclamation marks (!!!) – name this column changed\_text.**

SELECT i.name, i.type, replace(s.text, '.','!!!') as changed\_text

FROM item as i join slogan as s

ON i.id = s.item\_id

WHERE s.type = 'tv commercial'