Lab 1 - Normalization exercise

You need to apply the three normal forms for the following exercise:

A library database records the authors and the publisher of each book. Normalize the following relation and indicate the primary and foreign key attribute types:

R (ISBN, title, author(name, date_of_birth), publisher(name, address(streetnr, streetname, zipcode, city)), pages, price)

The assumptions are:

- each book has a unique ISBN number;
- each author has a unique name;
- each publisher has a unique name;
- a book can have multiple authors;
- an author can write more than one book;
- a publisher can publish more than one book;
- a book has only one publisher;
- a publisher has only one address.

Suppose that one book can have multiple publishers. How can you extend your model to accommodate this? Where would you put the attribute type "number_of_copies"?

Exercise source:

Lemahieu, W., vanden Broucke, S., & Baesens, B. (2018). *Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data*. Cambridge University Press

Chapter 6. Relational Databases: The Relational Model

Question 6.1 A library database records the authors and the publisher of each book. Normalize the following relation and indicate the primary and foreign key attribute types

R(ISBN, title, author(name, date_of_birth), publisher(name, address(streetnr, streetname, zipcode, city)), pages, price)

The assumptions are:

- each book has a unique ISBN number;
- each author has a unique name;
- each publisher has a unique name;
- a book can have multiple authors;
- an author can write more than one book;
- a publisher can publish more than one book;
- a book has only one publisher;
- a publisher has only one address.

Suppose that one book can have multiple publishers. How can you extend your model to accommodate this? Where would you put the attribute type "number_of_copies"?

Answer

First Normal Form

The first normal form is violated because Author is a multivalued composite attribute type and Publisher is a composite attribute type. The relation R can be brought in first normal form as follows:

BOOK(<u>ISBN</u>, title, publishername, streetnumber, streetname, zipcode, cityname, pages, price)

AUTHOR(<u>*ISBN*</u>, <u>authorname</u>, date_of_birth)

Second Normal Form

The relation AUTHOR is not in second normal form because date_of_birth is partially dependent on ISBN and authorname. In fact, it only depends on authorname. The relation AUTHOR can be brought in second normal form as follows:

BOOK(<u>ISBN</u>, title, publishername, streetnumber, streetname, zipcode, cityname, pages, price)

AUTHOR(<u>authorname</u>, date_of_birth)

WRITES(*authorname*, *ISBN*)

Third Normal Form Step 1

The relation BOOK is not in third normal form since we have a transitive functional dependence from ISBN to streetnumber via publishername, from ISBN to streetname via publishername, from ISBN to zipcode via publishername and from ISBN to cityname via publishername. The relation BOOK can be brought in third normal form as follows:

BOOK(ISBN, title, pages, price, *publishername*)

AUTHOR(authorname, date_of_birth)

WRITES(*authorname*, *ISBN)*

PUBLISHER(publishername, streetnumber, streetname, zipcode, cityname)

Third Normal Form Step 2

The relation BOOK is still not in third normal form since we have a transitive functional dependency from publishername to cityname via zipcode. The relation BOOK can be brought in third normal form as follows:

BOOK(ISBN, title, pages, price, *publishername*)

AUTHOR(authorname, date_of_birth)

WRITES(*authorname*, *ISBN)*

PUBLISHER(publishername, streetnumber, streetname, *zipcode*)

CITY(zipcode, cityname)

If you want to extend the model by allowing a book to be published by multiple publishers, then the relation BOOK is no longer in first normal form since publishername now becomes a multivalued attribute type. The new normalized set of relations then becomes:

BOOK(ISBN, title, pages, price)

AUTHOR(authorname, date_of_birth)

WRITES(*authorname*, *ISBN)*

PUBLISHER(publishername, streetnumber, streetname, *zipcode*)

CITY(zipcode, cityname)

PUBLISHES(*ISBN,publishername*)

The attribute type “number_of_copies“ can then be stored in the PUBLISHES relation.

Question 6.2 Given the following assumptions:

- a flight has a unique flight number, a passenger has a unique name, and a pilot has a unique name;
- a flight is always handled by one airline;
- a flight can have multiple passengers and a passenger can do multiple flights;
- a flight has one pilot and a pilot can do multiple flights;
- a flight is always handled by exactly one airplane.

normalize the following relation:

Flight(Flightnumber, Flighttime, airline(airlinename), passenger(passengername, gender, date of birth), pilot(pilotname, gender, date of birth), departure_city, arrival_city, airplane(planeID, type, seats))

Answer

First Normal Form

The relation Flight is not in first normal form since airline is a composite attribute types, passenger is a multivalued composite attribute type, pilot is a composite attribute type, and airplane is a composite attribute type. The relation Flight can be brought in first normal form as follows:

Flight(Flightnumber, Flighttime, airlinename, pilotname, gender, date of birth, departure_city, arrival_city, planeID, type, seats)
Flies(*Flightnumber*, passengername, gender, date of birth)

Second Normal Form

The relation Flies is not in second normal form since gender and date of birth are partially dependent on Flightnumber and passengername. In fact, both gender and date of birth only depend on passengername. The relation Flies can be brought in second normal form as follows:

Flight(Flightnumber, Flighttime, airlinename, pilotname, gender, date of birth, departure_city, arrival_city, planeID, type, seats)
Flies(*Flightnumber*, *passengername*)
Passenger(Passengername, gender, date of birth)

Introduction to MySQL Workbench – LAB 2

Scenario:

The human resources department of a company uses excel for the employee's data management, the company wants to implement a new system and hires a consultancy IT company to design the relational model of the HR database. The requirements of the department in the company are the following:

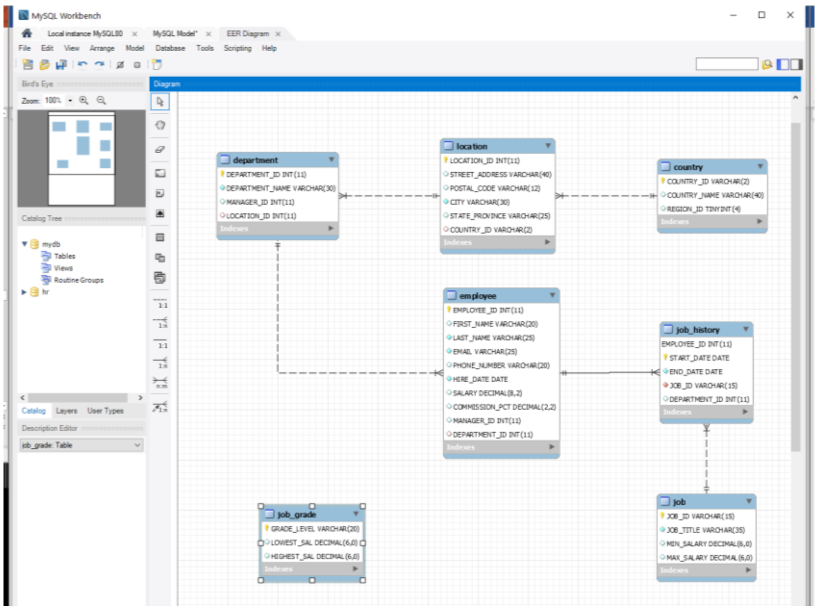
The company has locations in different countries. The company may have more than one location per country. Each location may have one or more departments of the company. Each department may have more than one employee, and each employee is assigned to one job; however, each job may be done by more than one employee. The human resources department needs to track a history of the jobs that the employees had in each of the departments. The employees may change from one job to another job. The existing jobs may have different grades/categories indicating the lowest and highest salary allowed for that job.

LAB 1:

1. Draw on paper an Entity-Relationship Model for the described scenario (try to use between 7 to 9 tables).
2. Make sure the database server is running. Then, open the MySQL workbench.
3. Create the database "hr" using the workbench.
4. Using the "hr" database, upload and run the script provided in Moodle ("CreateTables.sql"). The script will create the tables, the relationships are not yet created. Look to the code of the script. Are the primary keys defined? Are the foreign keys defined?
5. The database is now created, use the reverse engineering option to visualize the entity-relationship diagram (ERD) in workbench (see the annex 1 "LabSupplement-ReverseEngineeringHR_DB.pdf").
6. Compare the tables from the initial diagram that you drew in step 1 with the tables displayed in the ERD from class solution. How similar is your diagram?

Foreign keys and first SQL queries – LAB 3

1. Check HR database exists (if not, create the database HR, import "CreateTables.sql" from Lab 1)
2. Create the foreign keys (use the script "create_foreign_keys.sql").
3. Create ERD from the updated database (use Reverse Engineering, as learned in last class). Look at the relations between tables. They represent identifying and non-identifying relationships (more in https://www.eandbsoftware.org/difference-between-identifying-and-non-identifying-relationships-mysql-workbench/).



department
- DEPARTMENT_ID INT(11)
- DEPARTMENT_NAME VARCHAR(30)
- MANAGER_ID INT(11)
- LOCATION_ID INT(11)

location
- LOCATION_ID INT(11)
- STREET_ADDRESS VARCHAR(40)
- POSTAL_CODE VARCHAR(12)
- CITY VARCHAR(30)
- STATE_PROVINCE VARCHAR(25)
- COUNTRY_ID VARCHAR(2)

country
- COUNTRY_ID VARCHAR(2)
- COUNTRY_NAME VARCHAR(40)
- REGION_ID TINYINT(4)

employee
- EMPLOYEE_ID INT(11)
- FIRST_NAME VARCHAR(20)
- LAST_NAME VARCHAR(25)
- EMAIL VARCHAR(25)
- PHONE_NUMBER VARCHAR(20)
- HIRE_DATE DATE
- SALARY DECIMAL(8,2)
- COMMISSION_PCT DECIMAL(2,2)
- MANAGER_ID INT(11)
- DEPARTMENT_ID INT(11)

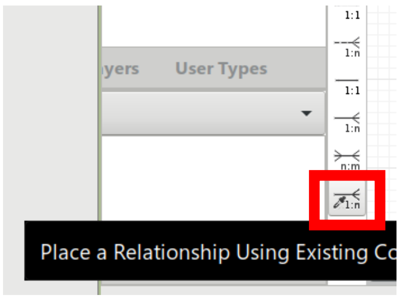
job_history
- EMPLOYEE_ID INT(11)
- START_DATE DATE
- END_DATE DATE
- JOB_ID VARCHAR(15)
- DEPARTMENT_ID INT(11)

job_grade
- GRADE_LEVEL VARCHAR(20)
- LOWEST_SAL DECIMAL(6,0)
- HIGHEST_SAL DECIMAL(6,0)

job
- JOB_ID VARCHAR(15)
- JOB_TITLE VARCHAR(35)
- MIN_SALARY DECIMAL(6,0)
- MAX_SALARY DECIMAL(6,0)

4. After looking at the ERD, the HR department decided that also wants to identify and store the regions where the countries are located (Europe, Asia, etc.). Then, you need to create a new table REGION (use the wizard). You can also do it by code.

5. Create (in workbench ERD) a foreign key to link country and region, where region is the parent table (Make sure region_id in both tables have equal type). Set up "On Update = cascade" and "On Delete = restrict"



6. Propagate the changes to the database (menu Database/Synchronize Model).

7. Insert four world regions using sql code: Europe, Americas, Asia, Middle East and Africa. (To open a sql query tab go to *File* → *New Query Tab*):

8. You receive a new update from the business user: the human resources department is not going to have job grades any longer. Then, you need to delete the table job_grade. Use sql code.

9. Insert the data into your tables (use the script “insertHRdata.sql”).
10. Write a query to display all the employee details.
11. Write a query to calculate the square root of (52.32 * 96.3)
12. Write a query to get the last name and first name of the first 10 employees.
13. Write a query to get the maximum and minimum salary from employees.
14. Write a query to get the last name (in upper case) of employees and the salary. Order by salary from the maximum to minimum.

GAD – LAB 3

Questions for team and class discussion (at the end of lab):

Considering the **foreign keys** created and the content of the tables what will happen if:

1. delete from department table the department ‘Payroll’? (why)

2. delete from department table the department ‘IT’? (why)

3. change in the department the department_id of the department ‘IT’ to be from 60 to 210? (why)

4. change in the department the department_id of the department ‘IT Helpdesk’ to be 235? (why)

5. change in the department the department_id of the department ‘IT’ to be from 60 to 65? (why)

Foreign keys and first SQL queries – LAB 3

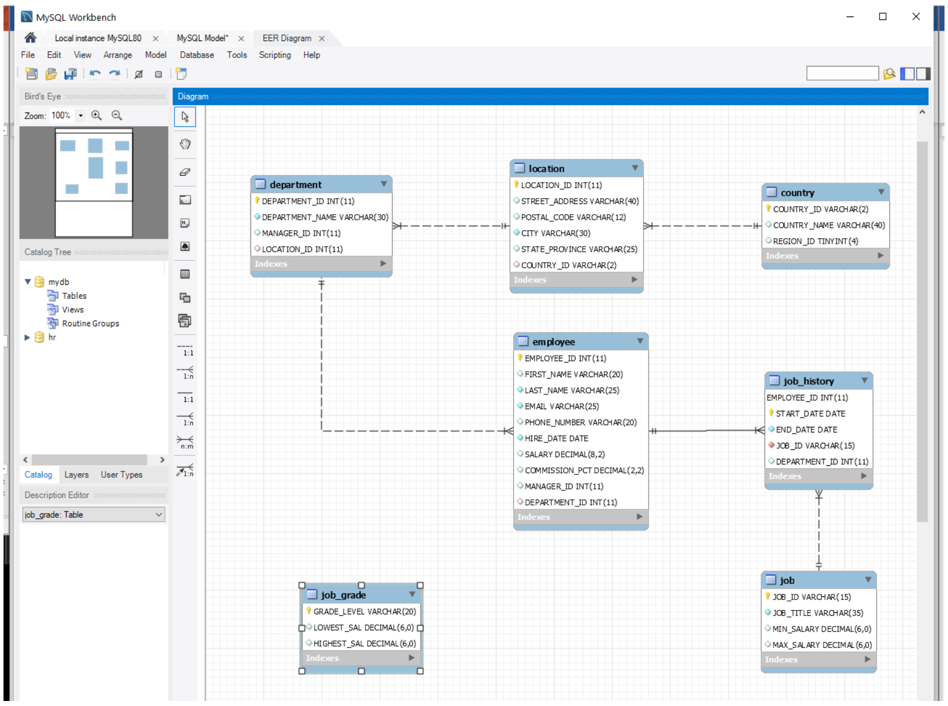
1. Check HR database exists (if not create database HR, import "CreateTables.sql")

   Solution: Right-click the "hr" database created and select "*set up as default schema*"
   Run the CreateTables.sql script (*File → Run SQL script*)

2. Create the foreign keys (use the script "create_foreign_keys.sql").

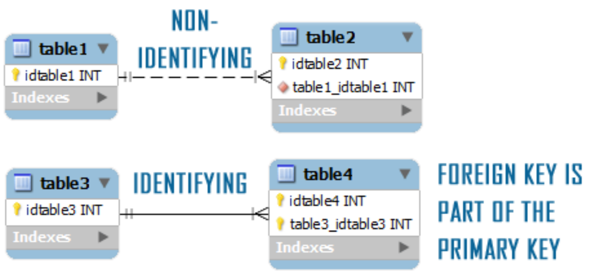
   Solution: Right-click the "hr" database created and select "*set up as default schema*"
   Run the create_foreign_keys.sql script (*File → Run SQL script*)

3. Create ERD from the updated database (use Reverse Engineering, as learned in last class). Look at the relations between tables.



Additional explanation of identifying and non-identifying relationships (source: https://www.eandbsoftware.org/difference-between-identifying-and-non-identifying-relationships-mysql-workbench/):

Non-Identifying Relationships have dotted lines, whereas Identifying Relationships have solid lines in MySQL Workbench. When you create an identifying relationship, the primary key of the child table becomes a foreign key in the parent table. In non-identifying relationships, the primary key of the child table is still included as a foreign key, but it doesn't get to participate as a primary key. Hence, it is non-identifying. Here is an example for a 1:n relationship:



Example: Taking this a bit further, you'll notice that in identifying relationships, it is really not possible to insert a record into the parent table without referencing the child table. So if you had a 1:n relationships between Brands and Products, where a Brand can have many Products, but a Product only has 1 Brand.

If that relationship is identifying, we'd have the Brand Primary Key as part of the Primary Key in Products. Therefore, each Product needs a Brand to be uniquely identified.
If that relationship is non-identifying, the Brand Primary Key is still a Foreign Key in the Product Table, but it's not part of the primary key. We can identify Products uniquely by Product ID alone.

4. After looking at the ERD, the HR department decided that also wants to identify and store the regions where the countries are located (Europe, Asia, etc.). Then, you need to create a new table REGION (using SQL code or the wizard).

Solution with wizard (Select the table icon on the left → click on the canvas → double click on the new table (table1) that appears):

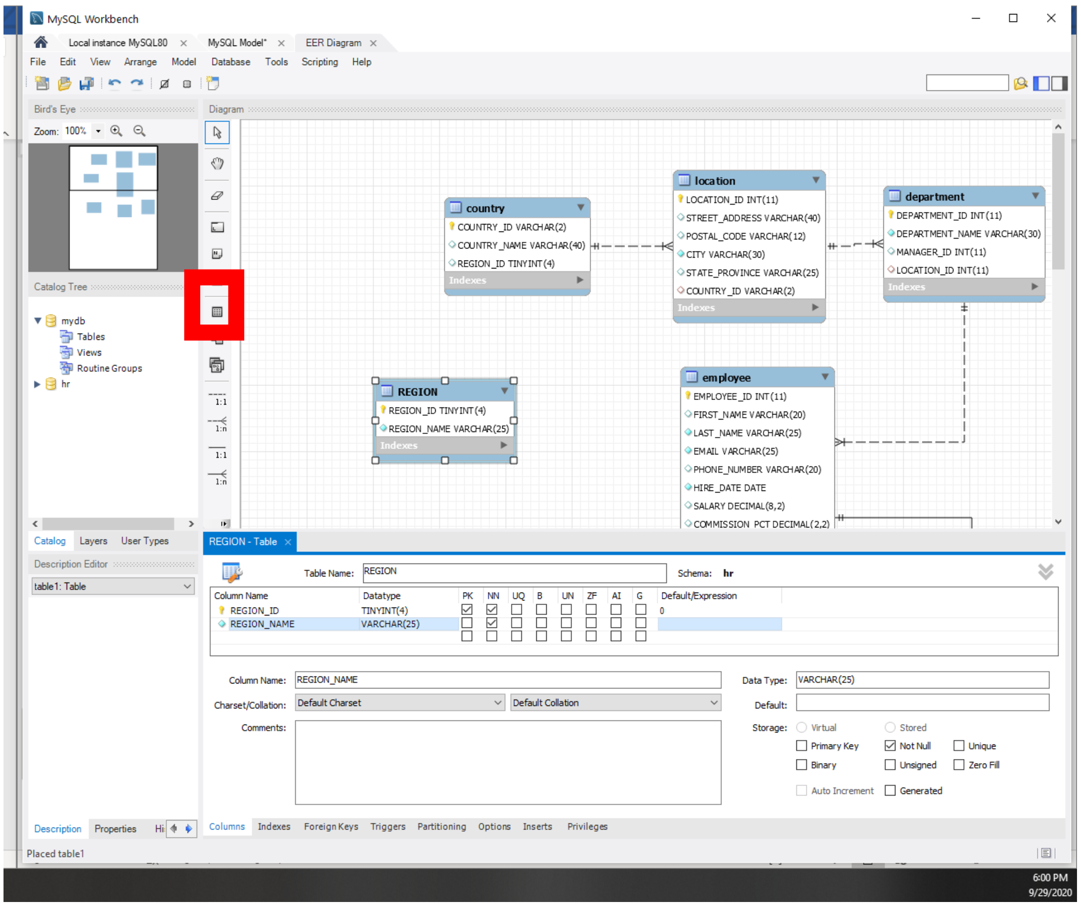


Table Name: REGION Schema: hr

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
REGION_ID	TINYINT(4)	✓	✓							0
REGION_NAME	VARCHAR(25)		✓							

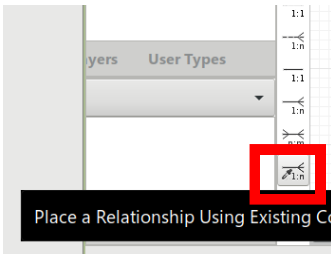
Column Name: REGION_NAME
Data Type: VARCHAR(25)
Charset/Collation: Default Charset / Default Collation
Default:
Comments:
Storage: Virtual Stored
Primary Key [✓ Not Null] Unique
Binary Unsigned Zero Fill
Auto Increment Generated

Alternative solution using SQL code (To open a sql query tab: *File → New Query Tab*):

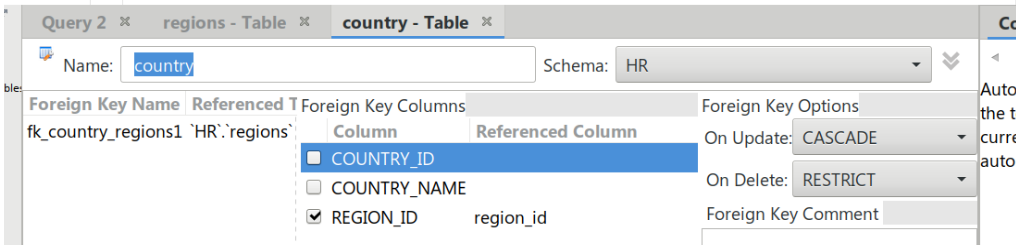
```
CREATE TABLE IF NOT EXISTS `region` (
   `REGION_ID` TINYINT NOT NULL,
   `REGION_NAME` varchar(25) DEFAULT NULL,
   PRIMARY KEY (`REGION_ID`)
) ;
```

5. Create (in workbench ERD) a foreign key to link country and region, where region is the parent table (Make sure region_id in both tables have equal type). Set up “On Update = cascade” and “On Delete = restrict”

   Solution: Click on the bottom in the red square, then click on the COUNTRY table, and then on the REGION table.



   Check if the new foreign key exists, modify “On Update” and “On Delete”



Name: country   Schema: HR

Foreign Key Name	Referenced T
fk_country_regions1	`HR`.`regions`

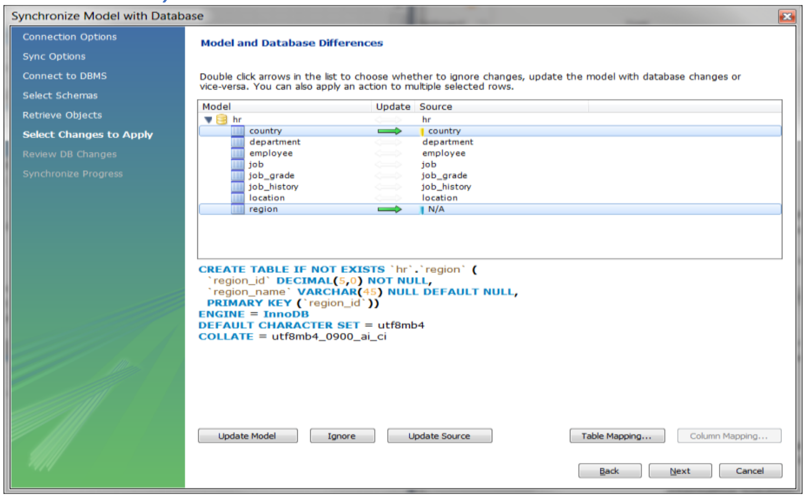
Foreign Key Columns

	Column	Referenced Column
☐	COUNTRY_ID	
☐	COUNTRY_NAME	
☑	REGION_ID	region_id

Foreign Key Options
On Update: CASCADE
On Delete: RESTRICT
Foreign Key Comment

6. Propagate the changes to the physical database.

Solution: Go to *Database → Synchronize Model.*



Model and Database Differences

Double click arrows in the list to choose whether to ignore changes, update the model with database changes or vice-versa. You can also apply an action to multiple selected rows.

```
Model            Update  Source
▼ hr                     hr
   country        ➡      country
   department            department
   employee              employee
   job                   job
   job_grade             job_grade
   job_history           job_history
   location              location
   region         ➡      N/A
```

```
CREATE TABLE IF NOT EXISTS `hr`.`region` (
  `region_id` DECIMAL(5,0) NOT NULL,
  `region_name` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`region_id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci
```

7. Insert four world regions using sql code: Europe, Americas, Asia, Middle East and Africa. (To open a sql query tab go to *File* → *New Query Tab*).

Solution using SQL code (To open a sql query tab: *File* → *New Query Tab*):

```
INSERT INTO `region` (`REGION_ID`, `REGION_NAME`) VALUES
(1, 'Europe'),
(2, 'Americas'),
(3, 'Asia'),
(4, 'Middle East and Africa');
```

8. You receive a new update from the business user; the human resources department is not going to have job grades any longer. Then, you need to delete the table job_grade.
   Solution using SQL code (To open a sql query tab: *File* → *New Query Tab*):

```
DROP TABLE IF EXISTS JOB_GRADE;
```

9. Insert the data into your tables (use the script "insertHRdata.sql").

   Solution: Run the insertHRdata.sql script (*File* → *Run SQL script*)

10. Write a query to display all the employee details.

```
select * from employee;
```

11. Write a query to calculate the square root of (52.32 * 96.3)

```
select sqrt(52.32 * 96.3) as square_root;
```

12. Write a query to get the last name and first name of the first 10 employees.

```
select e.last_name, e.first_name from employee e limit 10;
```

13. Write a query to get the difference between the maximum and minimum salary from employees.

```
select max(e.salary) - min(e.salary) from employee as e;
```

14. Write a query to get the last name (in upper case) of employees and the salary. Order by salary from the maximum to minimum.

```
select upper(e.last_name), e.salary from employee as e
order by e.salary desc;
```

GAD – LAB 3

Questions for team and class discussion (at the end of lab):

Considering the foreign keys created and the content of the tables what will happen if:

- delete from department table the department ‘Payroll’? (why)
(will be deleted. In the table “employee” is no employee with from that department)

- delete from department table the department ‘IT’? (why)
(will rise error. There are employees who belong to that department and the deletion is “Restricted”)

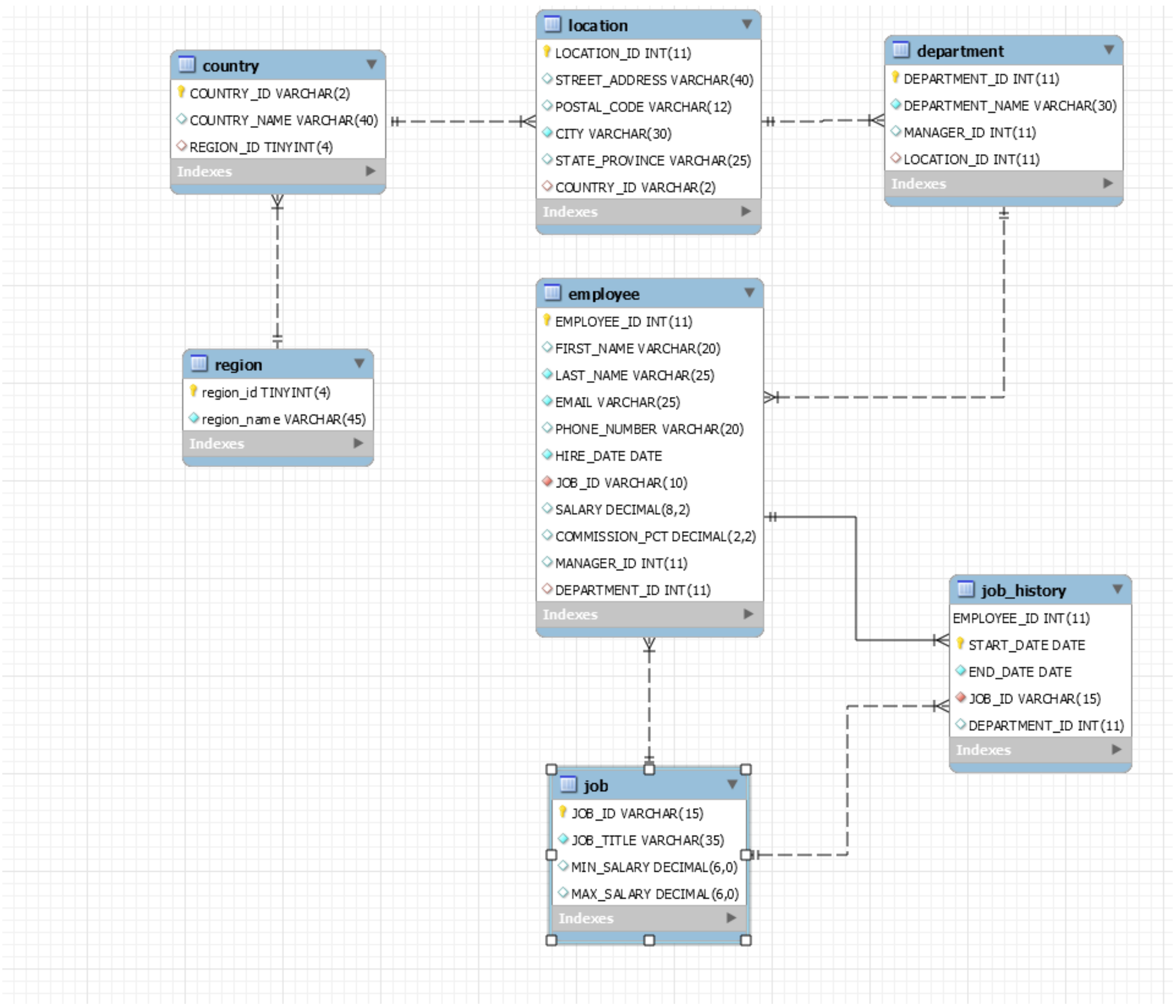
- change in the department the department_id of the department ‘IT’ to be from 60 to 210? (why)
(will rise error: the department_id 210 already exists but that column is primary key and so the values must be unique)

- change in the department the department_id of the department ‘IT Helpdesk’ to be 235? (why)
(will be changed, and nothing more. There are no employees from that department)

- change in the department the department_id of the department ‘IT’ to be from 60 to 65? (why)
(the department_id will be changed and the employees from employee table will get the new department_id because update is “cascaded”)

Use of SQL functions and Joins – LAB 4

1. Write a query to display all the last names of employees whose last name starts with ‘a’.
2. Write a query to display the last name and the email of each employee (append ‘@novaims.unl.pt’ to the email column).
3. Update the column EMAIL of the employees to append ‘@novaims.unl.pt’.
4. Write a query to display the number of employees whose first names contains an ‘a’ and that have the same job.
5. Write a query to get the average salary for all departments employing more than 15 employees. Round the average salary to two decimals.
6. List locations with theirs' addresses including country names
7. Same of above but with department name
8. Are there locations without any department? list them.
9. List the job titles with number of employees and average salary, sorted by average salary from highest to lowest.
10. List the department names with number of employees of each one including the ones without employees.
11. List employees' names with their managers' names. The manager name should concatenate the name and family name in one single field.



Use of SQL functions and Joins – LAB 4

1. Write a query to display all the last names of employees whose last name starts with 'a'.

```
select e.last_name from employee as e where e.last_name like 'a%';
```

2. Write a query to display the last name and the email of each employee (append '@novaims.unl.pt' to the email column).

```
select e.last_name, concat(e.email,'@novaims.unl.pt') from employee as e;
```

3. Update the column EMAIL of the employees to append '@novaims.unl.pt'.

```
update employee as e
set e.email = concat(e.email,'@novaims.unl.pt');
```

4. Write a query to display the number of employees whose first names contains an 'a' and that have the same job.

```
select e.job_id, count(e.employee_id) from employee as e
where e.first_name like '%a%'
group by e.job_id;
```

5. Write a query to get the average salary for all departments employing more than 15 employees. Round the average salary to two decimals.

```
select e.department_id, round(avg(e.salary),2) as avg_salary,
count(e.employee_id) as emp_x_dep
from employee as e
group by e.department_id
having emp_x_dep > 15;
```

6. List locations with theirs' addresses including country names

```
SELECT STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE, COUNTRY_NAME
FROM location l
JOIN country c ON l.COUNTRY_ID=c.COUNTRY_ID;
```

7. Same of above but with department name

```
SELECT DEPARTMENT_NAME, STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE,
COUNTRY_NAME
FROM location l
JOIN country c ON l.COUNTRY_ID=c.COUNTRY_ID
JOIN department d on d.LOCATION_ID=l.location_ID;
```

8. Are there locations without any department? list them.

```
SELECT l.STREET_ADDRESS, l.POSTAL_CODE, l.CITY, l.STATE_PROVINCE,
d.DEPARTMENT_ID
FROM location l
LEFT JOIN department d on l.LOCATION_ID=d.location_ID
WHERE d.DEPARTMENT_ID IS NULL;
```

9. List the job titles with number of employees and average salary, sorted by average salary from highest to lowest.

```
SELECT j.JOB_TITLE, ROUND(AVG(e.salary),2) as aver_sal, COUNT(1) as
numb_emp
FROM job j
```

```
JOIN employee e on e.JOB_ID=j.JOB_ID
GROUP BY j.JOB_ID
ORDER BY aver_sal DESC;
```

10. List the department names with number of employees of each one including the ones without employee s

```
SELECT d.DEPARTMENT_NAME, count(e.EMPLOYEE_ID) as numb_emp
FROM department d
LEFT JOIN employee e on e.DEPARTMENT_ID = d.DEPARTMENT_ID
GROUP BY d.DEPARTMENT_ID;
```

11. List employees' names with their managers' names (self join). The manager name should concatenate the name and family name in one single field.

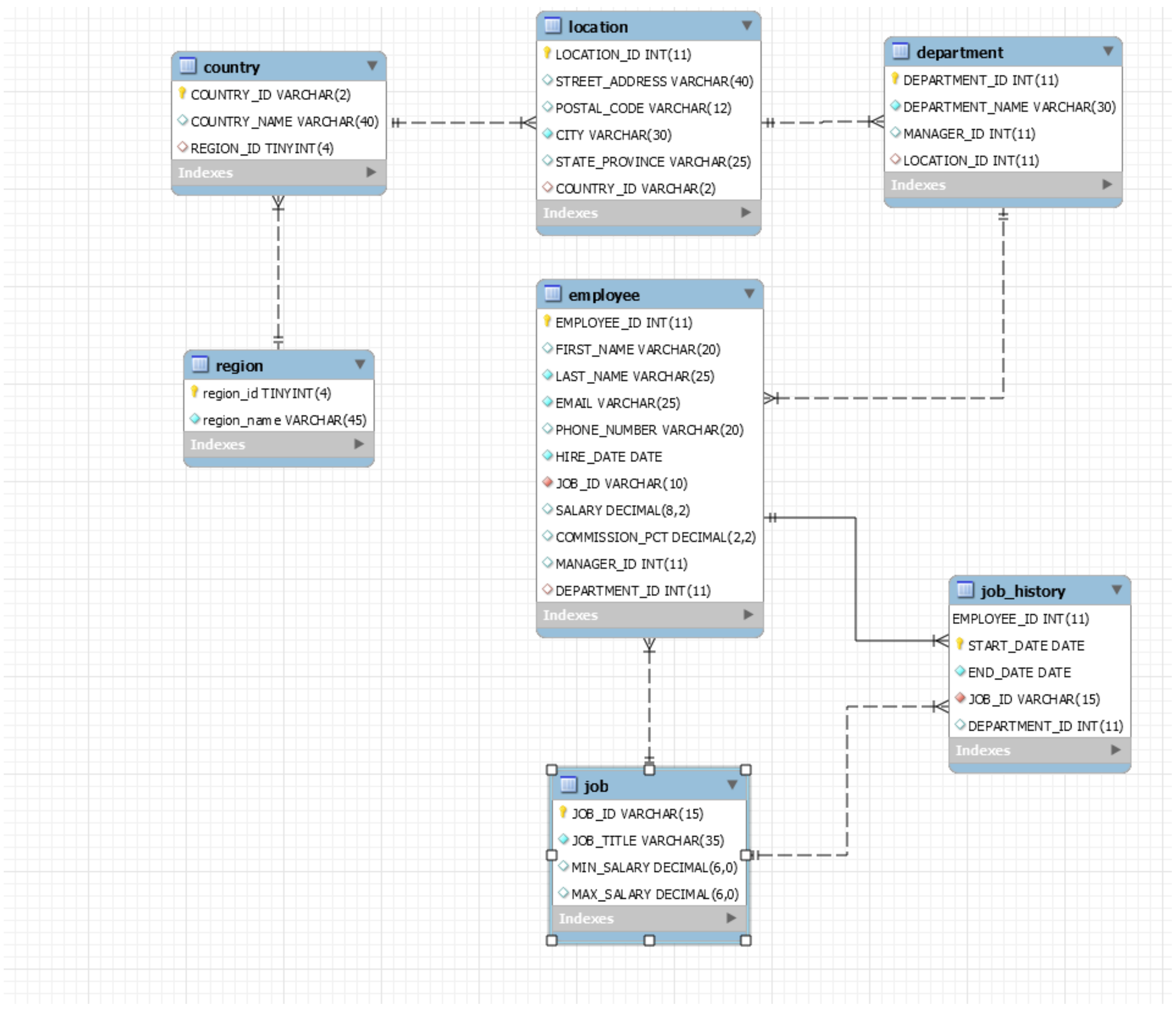
```
SELECT e.FIRST_NAME, e.LAST_NAME, concat(s.FIRST_NAME, " ", s.LAST_NAME)
as ManagerName
from employee as e
join employee as s on e.MANAGER_ID=s.EMPLOYEE_ID;
```

More Joins and Views – LAB 5

1. List all managers with number of managed employees, where the number of managed employees is bigger than 4.
2. List former job titles, start, and end dates of the employees sorted by start date.
3. Count employees by regions where there are employees.
4. Create a view jobtitle_salary with two columns:
   - salary
   - job title

   (No personal information should be included; The President's salary should not be listed)
5. Create a view jobtitle_salary_avg with averaged salaries by Job
   - average salary
   - job title

   (Again, the President's salary should not be listed)
6. Use the view jobtitle_salary to calculate the average salary by job title
7. Is the result from above equivalent of the one from jobtitle_salary_avg ignoring the order? And why?
8. Create a view employee_country that contains only the employees that belong to a department and that department has been assigned to a location and country. The view should show two columns:
   - full employee's name
   - county name where his department is located



More Joins and Views – LAB 5

1. List all managers with number of managed employees, where the number of managed employees is bigger than 4.

```
SELECT s.FIRST_NAME, s.LAST_NAME, COUNT(1) AS numb_managed
from employee as e
join employee as s on e.MANAGER_ID=s.EMPLOYEE_ID
group by s.EMPLOYEE_ID
having numb_managed > 4;
```

2. List former job titles, start and end dates of the employees sorted by start date

```
SELECT j.JOB_TITLE, e.FIRST_NAME, e.LAST_NAME, jh.START_DATE, jh.END_DATE
FROM job_history jh
JOIN employee e on jh.EMPLOYEE_ID=e.EMPLOYEE_ID
JOIN job j on jh.JOB_ID=j.JOB_ID
ORDER BY jh.START_DATE;
```

3. Count employees by regions where there are employees

```
select r.REGION_NAME, COUNT(1) AS numb_emp
from region r
join country c on c.REGION_ID=r.REGION_ID
join location l on l.COUNTRY_ID=c.COUNTRY_ID
join department d on d.LOCATION_ID=l.LOCATION_ID
join employee e on e.DEPARTMENT_ID=d.DEPARTMENT_ID
GROUP BY r.REGION_ID;
```

4. Create a view jobtitle_salary with two columns:

- salary
- job title

(No personal information should be included; The President's salary should not be listed)

```
create view jobtitle_salary as
select salary, j.JOB_TITLE from employee e
left join job j on e.JOB_ID=j.JOB_ID
where not j.JOB_TITLE = 'President';
```

5. Create a view jobtitle_salary_avg with averaged salaries by Job

- average salary
- job title

(Again, the President's salary should not be listed)

```
create view jobtitle_salary_avg as
select AVG(salary) as salary_avg, j.JOB_TITLE from employee e
left join job j on e.JOB_ID=j.JOB_ID
where not j.JOB_TITLE = 'President'
group by j.JOB_ID;
```

6. Use the view jobtitle_salary to calculate the average salary by job title

```
select avg(salary) as salary_avg, job_title from jobtitle_salary
group by JOB_TITLE;
```

7. Is the result from above equivalent of the one from jobtitle_salary_avg ignoring the order? And why?

The content of the resulting tables may be the same but is not equivalent.
The grouping in the last query is done over Job Title, which is not unique, while the view jobtitle_salary_avg has grouping over job_id which is unique (as primary key). So, it is possible to add a new job_id with already existing job title and then the result will not be the same.

8. Create a view employee_country that contains only the employees that belong to a department and that department has been assigned to a location and country. The view should show two columns:

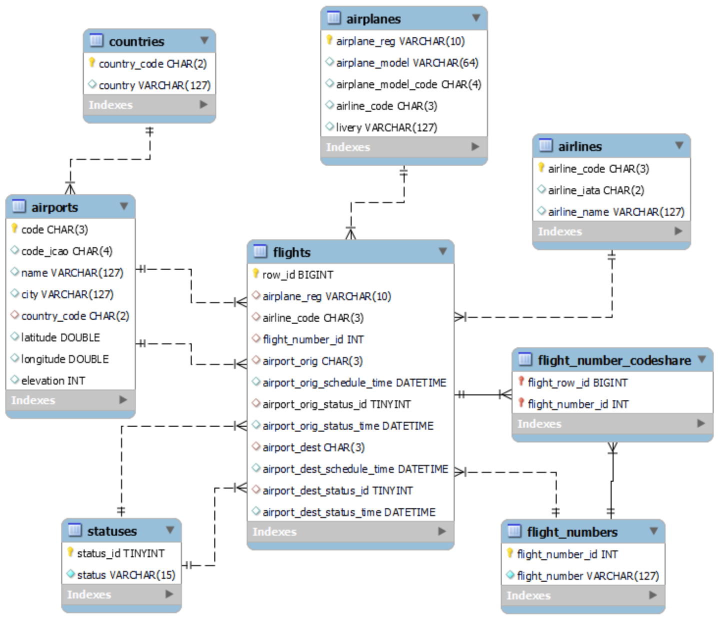
- full employee's name

- county name where his department is located

```
create view employee_country as
select concat(e.FIRST_NAME,' ',e.LAST_NAME) as name, c.COUNTRY_NAME as
country from employee as e
join department d on e.DEPARTMENT_ID =d.DEPARTMENT_ID
join location l on d.LOCATION_ID =l.LOCATION_ID
join country c on l.COUNTRY_ID =c.COUNTRY_ID;
```

Query Optimization Exercises – Part 1 – LAB 6

Load and run the “create_flropt_db” script from Moodle:



1. From the flights table retrieve the flight where the flight_number_id is “146225”. Look at the query stats using the EXPLAIN operator, also can you see the query’s Execution Plan?

2. From the flight_numbers table select the flight with flight_number_id equal to 127. Look at the query stats (you can use the EXPLAIN operator).

3. From the flight_numbers table select the flight with flight_number equal to ‘DL9411’. Look at the query stats (you can use the EXPLAIN operator).

4. Can you detect any difference in terms of query stats from the two previous questions?

5. What are the top five countries that received the most flights?

6. What are the top five airline company names with the highest number of cancelled flights?

Load and run the “flropt_fkeys” script from Moodle:

7. Re-run the queries you’ve built in questions 1, 4 and 5. Can you detect any differences?

Trigger Exercises – LAB 6

Implement logging for changes in table "employees" using triggers:

1. Create a table called "log" with columns:

   ID (primary key, unsigned, integer, autoincrement),
   TS (DATETIME),
   USR (Varchar),
   EV (Varchar),
   MSG (Varchar)

2. Create a trigger that, after a new employee is added, adds to the "log" table:
   a The current time (use function "NOW()").
   b The current user ("USER()").
   c The text "add".
   d The employee's full name.

3. Same exercise, but when the "employee" table has an updated row (change the text to "update").
4. Same exercise, but when some employee is deleted (change the text to "delete").
5. Modify, add, and delete some rows in the "employee" table and observe the content of the "log" table.

Implement a table with the current salary property over the whole company (min, max, average, count, last updated), updated by triggers every time employees are added, removed, or changed:

6. Create a table "status" with the columns:

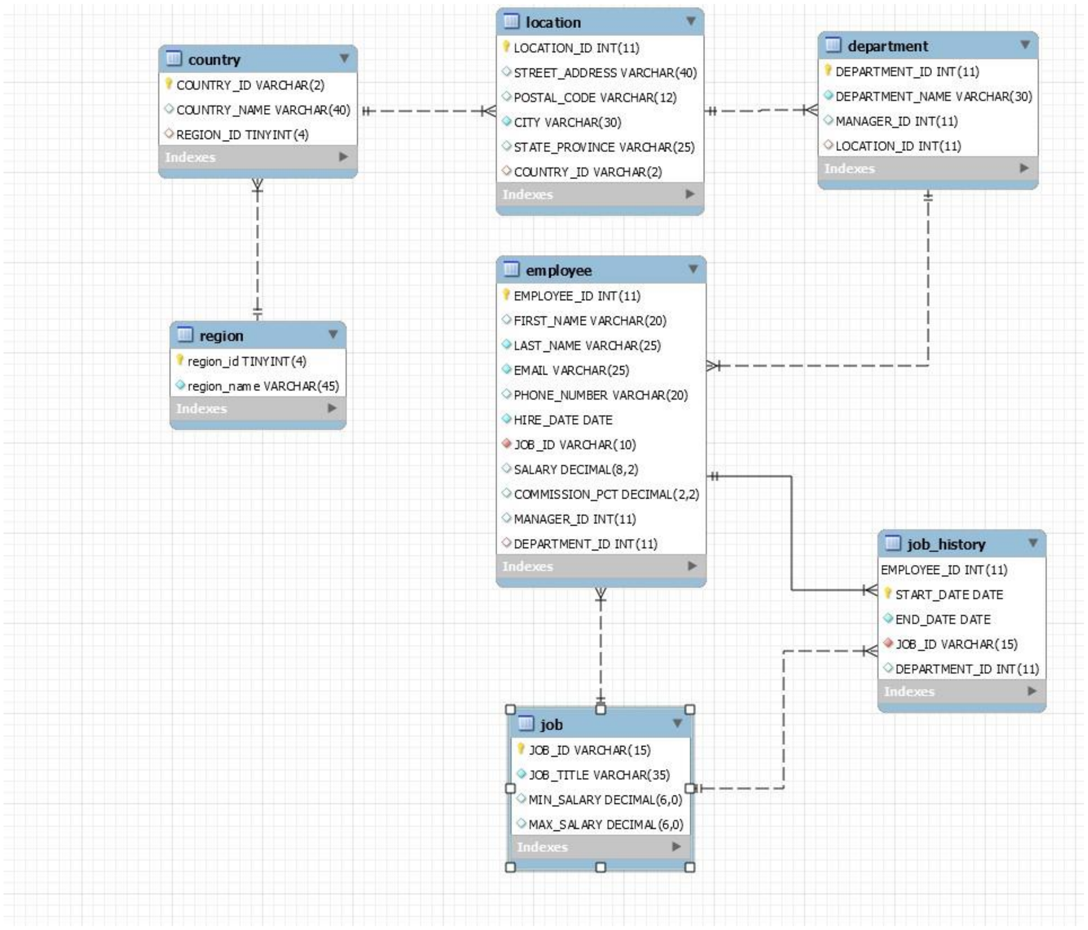
   PK (Primary key, Integer, default 0),
   SALARY_MIN (FLOAT),
   SALARY_MAX (FLOAT),
   SALARY_AVG (FLOAT),
   EMP_COUNT (INTEGER, UNSIGNED, DEFAULT 0, NOT NULL),
   LAST_UPD (DATATIME)

7. Create a query that replaces the single row in "status" with actual values. Use REPLACE ... SELECT syntax:

```
REPLACE status
  (PK, SALARY_AVG, SALARY_MIN, SALARY_MAX, EMP_COUNT, LAST_UPD)
SELECT 0, avg(SALARY), max(SALARY), min(SALARY), COUNT(1),
NOW()  FROM employee;
```

Create triggers in the employee table after delete that update the content of table "status" using the query from the previous point.

8. Enhance the triggers from questions 2 and 3 to perform on the status table.
9. Modify the "employee" table and observe the changes in the "status" table.


country
COUNTRY_ID VARCHAR(2)
COUNTRY_NAME VARCHAR(40)
REGION_ID TINYINT(4)
Indexes
location
LOCATION_ID INT(11)
STREET_ADDRESS VARCHAR(40)
POSTAL_CODE VARCHAR(12)
CITY VARCHAR(30)
STATE_PROVINCE VARCHAR(25)
COUNTRY_ID VARCHAR(2)
Indexes
department
DEPARTMENT_ID INT(11)
DEPARTMENT_NAME VARCHAR(30)
MANAGER_ID INT(11)
LOCATION_ID INT(11)
Indexes
region
region_id TINYINT(4)
region_name VARCHAR(45)
Indexes
employee
EMPLOYEE_ID INT(11)
FIRST_NAME VARCHAR(20)
LAST_NAME VARCHAR(25)
EMAIL VARCHAR(25)
PHONE_NUMBER VARCHAR(20)
HIRE_DATE DATE
JOB_ID VARCHAR(10)
SALARY DECIMAL(8,2)
COMMISSION_PCT DECIMAL(2,2)
MANAGER_ID INT(11)
DEPARTMENT_ID INT(11)
Indexes
job_history
EMPLOYEE_ID INT(11)
START_DATE DATE
END_DATE DATE
JOB_ID VARCHAR(15)
DEPARTMENT_ID INT(11)
Indexes
job
JOB_ID VARCHAR(15)
JOB_TITLE VARCHAR(35)
MIN_SALARY DECIMAL(6,0)
MAX_SALARY DECIMAL(6,0)
Indexes

A. Implement logging for changes in table “employees” using triggers:
1. Create table “log” with columns: ID (primay key, integer, autoincrement),TS (DATETIME), USR(VARCHAR(63)), EV(VARCHAR(15)),MSG(VARCHAR(255))

```
CREATE TABLE log (
LOG_ID INTEGER UNSIGNED AUTO_INCREMENT PRIMARY KEY,
DT DATETIME NOT NULL,
USR VARCHAR(63),
EV VARCHAR(15),
MSG VARCHAR(255)
);
```

2. Create trigger that add to the “log” table current time (use function NOW()), the current user (USER()), the text “add” and the employee full name when new employee is added

```
DELIMITER $$
CREATE TRIGGER emp_ai_log
AFTER INSERT
ON employee
FOR EACH ROW
BEGIN
      INSERT log (TS,USR,EV,MSG)
      VALUES (NOW(),USER(),"add",CONCAT(NEW.FIRST_NAME,' ',NEW.LAST_NAME));
END $$
DELIMITER ;
```

3. Same but when “employee” table has updated row

```
DELIMITER $$
CREATE TRIGGER emp_au_log
AFTER UPDATE
ON employee
FOR EACH ROW
BEGIN
      INSERT log (TS,USR,EV,MSG) VALUES
      (NOW(),USER(),"update",CONCAT(NEW.FIRST_NAME,' ',NEW.LAST_NAME));
END $$
DELIMITER ;
```

4. Same but when some employee is deleted

```
DELIMITER $$
CREATE TRIGGER emp_bd_log
BEFORE DELETE
ON employee
FOR EACH ROW
BEGIN
      INSERT log (TS,USR,EV,MSG) VALUES
      (NOW(),USER(),"delete",CONCAT(OLD.FIRST_NAME,' ',OLD.LAST_NAME));
END $$
DELIMITER ;
```

5. Modify, add and delete some rows in “employee” table and observe the content of “log” table

B. Implement table with current salary property over the whole company (min, max, average, count, last updated), updated by triggers every time when employees are added, removed or changed;

1. Create table status with columns: PK (Primary key, Integer, default 0), SALARY_MIN, SALARY_MAX, SALARY_AVG (FLOAT), EMP_COUNT (INTEGER UNSIGNED, DEFAULT 0, NOT NULL), LAST_UPD (DATATIME)

```
CREATE TABLE status (
  PK INTEGER UNSIGNED DEFAULT 0 PRIMARY KEY,
  SALARY_AVG FLOAT,
  SALARY_MIN FLOAT,
  SALARY_MAX FLOAT,
  EMP_COUNT INTEGER UNSIGNED DEFAULT 0 NOT NULL,
  LAST_UPD DATETIME
);
```

2. Create query that replaces the single row in “status” with actual values. Use REPLACE … SELECT syntax:

```
REPLACE status (PK,SALARY_AVG,SALARY_MIN,SALARY_MAX,EMP_COUNT,LAST_UPD)
SELECT 0,avg(SALARY),max(SALARY),min(SALARY),COUNT(1),NOW() FROM employee;
```

3. Create triggers in employee table **after** delete that update the content of table “status” using the query from the previous point

```
DELIMITER $$
CREATE TRIGGER emp_ad_status
AFTER DELETE
ON employee
FOR EACH ROW
BEGIN
  REPLACE status (PK,SALARY_AVG,SALARY_MIN,SALARY_MAX,EMP_COUNT,LAST_UPD)
  SELECT 0,avg(SALARY),max(SALARY),min(SALARY),COUNT(1),NOW() FROM employee;
END $$
DELIMITER ;
```

4. Enhance the triggers from points A.2. and A.3 to perform the status table

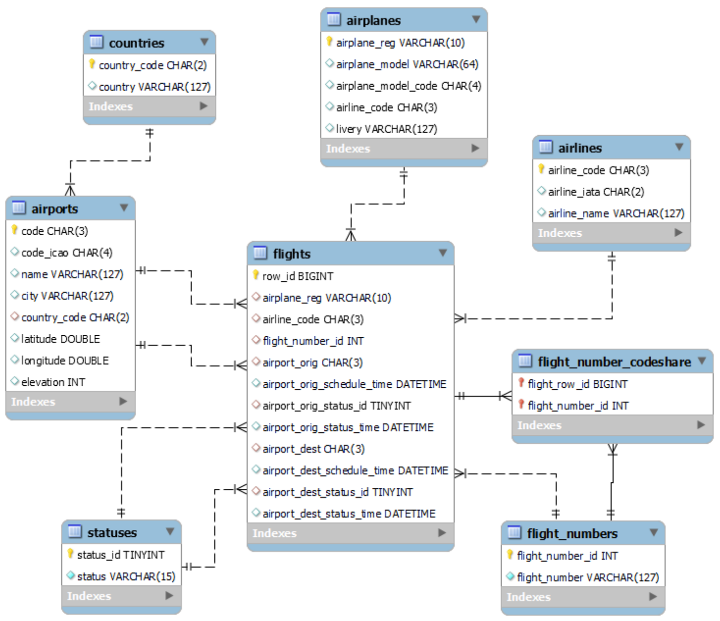
```
DROP TRIGGER emp_ai_log;
DELIMITER $$
CREATE TRIGGER emp_ai_log
AFTER INSERT
ON employee
FOR EACH ROW
BEGIN
      INSERT log (TS,USR,EV,MSG)
      VALUES (NOW(),USER(),"add",CONCAT(NEW.FIRST_NAME,' ',NEW.LAST_NAME));
      REPLACE status (PK,SALARY_AVG,SALARY_MIN,SALARY_MAX,EMP_COUNT,LAST_UPD)
      SELECT 0,avg(SALARY),max(SALARY),min(SALARY),COUNT(1),NOW() FROM employee;
END $$
DELIMITER ;

DROP TRIGGER emp_au_log;
DELIMITER $$
CREATE TRIGGER emp_au_log
AFTER UPDATE
ON employee
FOR EACH ROW
BEGIN
      INSERT log (TS,USR,EV,MSG) VALUES
      (NOW(),USER(),"update",CONCAT(NEW.FIRST_NAME,' ',NEW.LAST_NAME));
      REPLACE status (PK,SALARY_AVG,SALARY_MIN,SALARY_MAX,EMP_COUNT,LAST_UPD)
      SELECT 0,avg(SALARY),max(SALARY),min(SALARY),COUNT(1),NOW() FROM employee;
END $$
DELIMITER ;
```

5. Modify the “employee” table and observe that changes in “status” table

Query Optimization Exercises – Part 2 – LAB 7

Be sure that the “create_flropt_db” and “flropt_fkeys” scripts from Moodle have been loaded and ran:



You are hired to optimize the “flropt” database so that it can consistently and in a brief manner output a set of information’s for the normal functioning of any given airport. For each required information **write the required queries, benchmark them, and create relevant indexes to speed up the process.**

1. At each moment, any passenger in the airport needs to know all flights that are leaving the airport in the next 3 hours. Produce a query to feed this invoice with the given requisites:
    a. You are doing it for the Lisbon airport.
    b. For testing’s sake, assume that the current timestamp is 01-01-2021 at 11:00 am.

(Day, Hour, Flight Number, Destination Country, Air Company Name, Status)

2. At any moment, airlines need to know which of their planes are on the air or are grounded due to cancellations or other factors. Produce a query to feed this invoice with the given requisites:
    a. You are doing it for the airline TAP.
    b. For testing’s sake, assume that the current timestamp is 12-01-2021 at 10:00 am.

(Day, Hour, Flight Number, Airplane Model, Origin Airport, Origin Time, Destination Airport, Destination Time, Status)

3. Even though flight cancelations depend on a wide variety of variables, knowing the consistency of airports can have its relevancy. Produce a query that outputs the probability of cancelation (number of cancelled flights/number of flights) for each airport for the previous week.
    a. For testing’s sake, assume that the current timestamp is 20-01-2021 at 10:00 am.

(Last Week Date, Current Date, Airport Number, Number of Flights, Number of Cancelations, Cancelation Probability)

```sql
-- refresh statistics
analyze table flights, airlines,
airplanes,airports,countries,flight_number_codeshare,flight_numbers,statuses;
-- show occupied space
SELECT table_schema "DB Name", '*' as "table_name",
  sum(data_length) AS "data", sum(index_length) AS "index",
  ROUND((sum(data_length) + sum(index_length)) / 1024 / 1024, 1) "Size in MB"
FROM information_schema.tables
WHERE table_schema='flropt'
UNION
SELECT table_schema, table_name, data_length, index_length,
       ROUND((data_length + index_length) / 1024 / 1024, 1)
FROM information_schema.tables
WHERE table_schema='flropt';
```

```sql
-- Part1,1.
explain
-- analyze
-- 0.325 sec -- 0.051 sec
select * from flights where flight_number_id=146225;
```

```sql
-- Part1,2.
explain
select * from flight_numbers
WHERE flight_number_id=127;
```

```sql
-- Part1,3.
select * from flight_numbers
WHERE flight_number='DL9411';
```

```sql
-- Part1,5.
-- 2.363 -- 0.786
explain
-- analyze
select count(1) c, countries.country from flights
join airports on flights.airport_dest=airports.code
join countries on airports.country_code=countries.country_code
group by airports.country_code
-- group by countries.country_code
-- group by countries.country
order by c desc
limit 5;
```

```sql
-- Part1,5a.
-- 2.640 sec, 0.942 sec
explain
select count(1) c, country, country_code from (
  select countries.country, countries.country_code from flights
  join airports on flights.airport_dest=airports.code
  join countries on airports.country_code=countries.country_code
) t
group by country
order by c desc
limit 5;
```

```
-- Part1,6.
-- 0.403 sec, 0.123
explain
-- analyze
SELECT COUNT(1) C,airlines.airline_name FROM flights
JOIN airlines on flights.airline_code=airlines.airline_code
JOIN statuses on flights.airport_orig_status_id = statuses.status_id
-- where flights.airport_orig_status_id=10
 where status='canceled'
group by flights.airline_code
order by c desc limit 5;
```

```
-- =====
```

```
-- Part2,1.
-- 0.259, 0.025
-- explain
SELECT airport_orig_schedule_time, flight_number, countries.country as
dest_country, airlines.airline_name, statuses.status FROM flights
JOIN flight_numbers ON flights.flight_number_id=flight_numbers.flight_number_id
JOIN airports on flights.airport_dest=airports.code
JOIN countries on airports.country_code=countries.country_code
JOIN airlines on flights.airline_code=airlines.airline_code
JOIN statuses on flights.airport_orig_status_id=statuses.status_id
JOIN airports airports_orig on flights.airport_orig=airports_orig.code
WHERE airports_orig.city='lisbon'
and airport_orig_schedule_time between '2022-01-01 11:00' and '2022-01-01
13:59:59.9'
ORDER BY airport_orig_schedule_time;
```

```
-- Part2,2.
-- 0.286(0.484) -- (0.114)
-- explain
select flight_number, airplane_model,
airports_orig.name, airport_orig_schedule_time,
airports_dest.name, airport_dest_schedule_time
from flights
JOIN flight_numbers ON flights.flight_number_id=flight_numbers.flight_number_id
join airplanes on flights.airplane_reg=airplanes.airplane_reg
join airlines on airplanes.airline_code=airlines.airline_code
JOIN airports airports_orig on flights.airport_orig=airports_orig.code
JOIN airports airports_dest on flights.airport_dest=airports_dest.code
WHERE airport_orig_status_time<='2022-01-12 10:00' and
airport_dest_status_time>='2022-01-12 10:00'
-- and airlines.airline_name like 'TAP%'
;
```

```
-- Part2,3.
-- explain
select min(airport_orig_schedule_time) min_time,
max(airport_orig_schedule_time) max_time,
airports.name,
count(1) as num_flights,
 round((count(case when statuses.status='departed' then null else 1
end)/count(1))*100) as per_canc
-- round((sum(not statuses.status='departed')/count(1))*100) as per_canc
from flights
join statuses on flights.airport_orig_status_id=statuses.status_id
join airports on flights.airport_orig=airports.code
where airport_orig_schedule_time between
  DATE_SUB('2022-01-20 10:00', interval 1 week) and '2022-01-20 10:00'
group by flights.airport_orig;
```