Unit 3: Introduction to SQL.

2020/2021

Contents

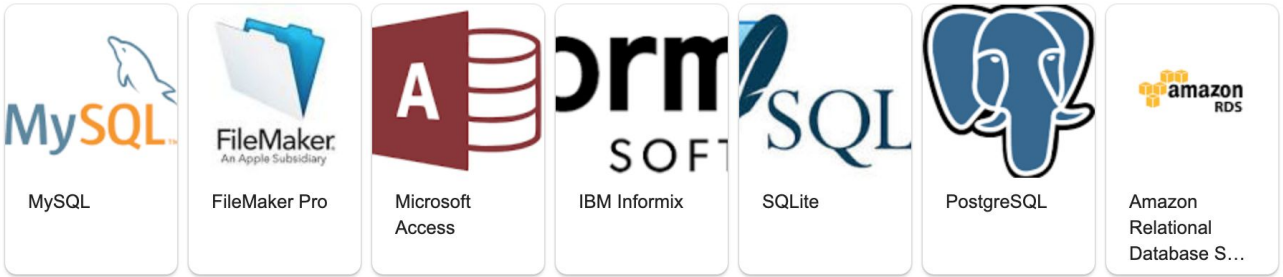
2.1. History.

- **SEQUEL** (Structured English Query Language) developed by IBM, designed to manipulate and retrieve data stored in IBM's original quasi-relational database management system.
- Renamed **Structured Query Language** (**SQL**), because "SEQUEL" was a registered trademark.
- By 1986, **ANSI and ISO standard groups** officially adopted the standard "**Database Language SQL**" language definition. New versions of the standard were published in 1989, 1992, 1996, 1999, 2003, 2006, 2008, 2011, and 2016.

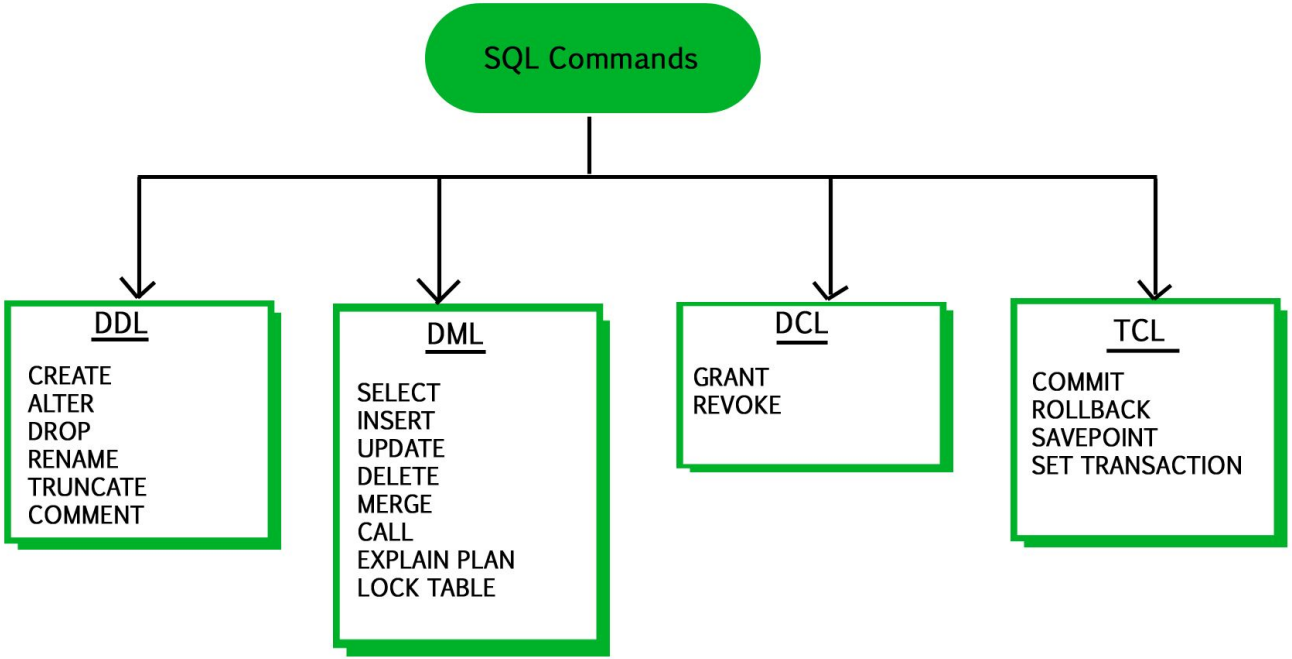
2.1. History (II).

- **Commercial systems** offer most, if not all, **SQL-92** features, plus varying propietary feature sets: Not all examples of the slides may work on your particular system.

And many more!



2.1. History (III).



2.2. Data Definition Language - DDL.

DDL (Data Definition Language): Part of SQL to define the **database schema**.

Statement	Function
CREATE DATABASE	Creates a new database and the file used to store the database.
DROP DATABASE	Removes an existing database.
CREATE TABLE	Creates a new table.
DROP TABLE	Removes a table definition and all data, indexes, and constraints for that table.
ALTER TABLE	Modifies a table definition by altering, adding, or dropping columns and constraints.
CREATE INDEX	Creates an index on a given table.
DROP INDEX	Removes one or more indexes from the current database.

2.3. Creating a database.

CREATE DATABASE example:

```
CREATE DATABASE shop;
```

Creating a database does not select it for use:

```
SELECT * FROM shop.customers;
```

OR:

```
USE shop;

SELECT * FROM customers;
```

2.4. Domain Types in SQL.

- **char(n)**: Fixed length character string (n: length).
- **varchar(n)**: Variable length character strings (n: length).
- **int**: Integer. Others: smallint, longint, ...
- **numeric(p,d)**: Fixed point number (p: precision of p digits, d: digits to the decimal point).
- **real, double precision**: Floating point and double-precision floating point numbers.
- **float(n)**: Floating point number (with precision of at least n digits).
- And many more that we will see in the following units...

2.5. Create Table.

- An SQL relation is defined using the create table command:
  - **create table** r (A_1 D_1, A_2 D_2, ..., A_n D_n,
    (integrity-constraint$_1$),
    ...,
    (integrity-constraint$_k$))
  - r is the name of the relation
  - each Ai is an attribute name in the schema of relation r
  - Di is the data type of values in the domain of attribute Ai

- Example:

```
create table DEPARTMENTS (
  num       number not null,
  name          varchar2(30) not null,
  constraint pk_departments primary key (num)
);
```

2.6. Integrity constraints in create table.

- **not null**
- **primary key** (A1, ..., An)
- **foreign key** (Am, ..., An) **references** r
- Example:

```
CREATE TABLE EMPLOYEES (
        num    INTEGER,
        surname VARCHAR(50) NOT NULL,
        name VARCHAR(50) NOT NULL,
        manager INTEGER,
        start_date DATE,
        salary INTEGER,
        commission INTEGER,
        dept_num INTEGER DEFAULT 10,
        PRIMARY KEY (num),
        FOREIGN KEY (dept_num)
                REFERENCES DEPARTMENTS (num),
        FOREIGN KEY (manager)
                REFERENCES EMPLOYEES (num)
        );
```

- **primary key** declaration on an attribute automatically ensures **not null**

2.6. Integrity constraints in create table (II).

```
CREATE TABLE PEOPLE (
        nif VARCHAR(9),
        name VARCHAR(40),
        surname VARCHAR(40),
        PRIMARY KEY (nif)
);
```

```
CREATE TABLE MODELS (
        code VARCHAR(5),
        name VARCHAR(40),
        brand VARCHAR(5),
        PRIMARY KEY (code),
        FOREIGN KEY (brand) references BRANDS (code)
) ;
```

```
CREATE TABLE BRANDS (
        code VARCHAR(5),
        name VARCHAR(40),
        PRIMARY KEY (code)
);
```

```
CREATE TABLE PEOPLE_VEHICLES (
        plate_number VARCHAR(7),
        nif VARCHAR(9),
        PRIMARY KEY (plate_number, nif),
        FOREIGN KEY (plate_number) REFERENCES VEHICLES (plate_number),
        FOREIGN KEY (nif) REFERENCES PEOPLE (nif)
);
```

```
CREATE TABLE VEHICLES (
        plate_number VARCHAR(7),
        model VARCHAR(5),
        PRIMARY KEY (plate_number),
        FOREIGN KEY (model) references MODELS (code)
) ;
```

```
CONSTRAINT `PEOPLE_VEHICLE_FK2` FOREIGN KEY (nif) REFERENCES PEOPLE (nif)
```

2.7. Updates to tables (I).

- **Insert**
  - **insert into** OCCUPATIONS (code, name) **values** ('MAN', 'MANAGER');
  - **insert into** OCCUPATIONS **values** ('MAN', 'MANAGER');
  - **insert into** OCCUPATIONS (code, name) **values**
    ('MAN', 'MANAGER'),
    ('EMP', 'EMPLOYEE'),
    ('SAL', 'SALESMAN'),
    ('ANA', 'ANALYST'),
    ('OWN', 'OWNER');
- **Delete**
  - Remove tuples from the OCCUPATIONS relation
    - **delete from** OCCUPATIONS **where** code = ’OWN’;
    - **delete from** OCCUPATIONS; <- deletes everything inside the table!

2.7. Updates to tables (II).

- **Update**
  - **UPDATE** EMPLOYEES **SET** occ_code = 'EMP' **WHERE** occupation = 'EMPLOYEE';
  - **UPDATE** EMPLOYEES **SET** name = 'Sergio', surname = 'González' **WHERE** num = 8001;
  - **UPDATE** EMPLOYEES **SET** salary = salary*1.05 **WHERE** salary < 9000;
- **In a nutshell, DML (Data Manipulation Language) is:**

Statement	Function
INSERT	Adds a new row to a table.
UPDATE	Changes existing data in a table.
DELETE	Removes rows from a table.

Let’s work!

Do the exercise: create_table.

In this exercise you will use/see:

- show databases
- use database_name
- describe table_name
- create database
- create table (not null, default, primary key, foreign key)
- alter table (if you make a mistake creating the table)
- insert
- update
- delete
- MyISAM Vs. InnoDB

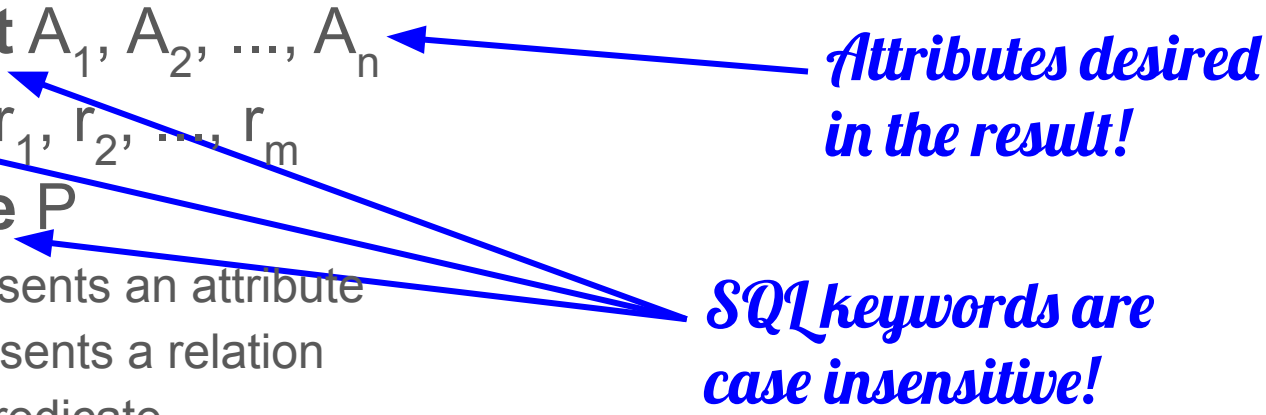
2.8. The select clause (I).

- Structure of a SQL query sentence:

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P

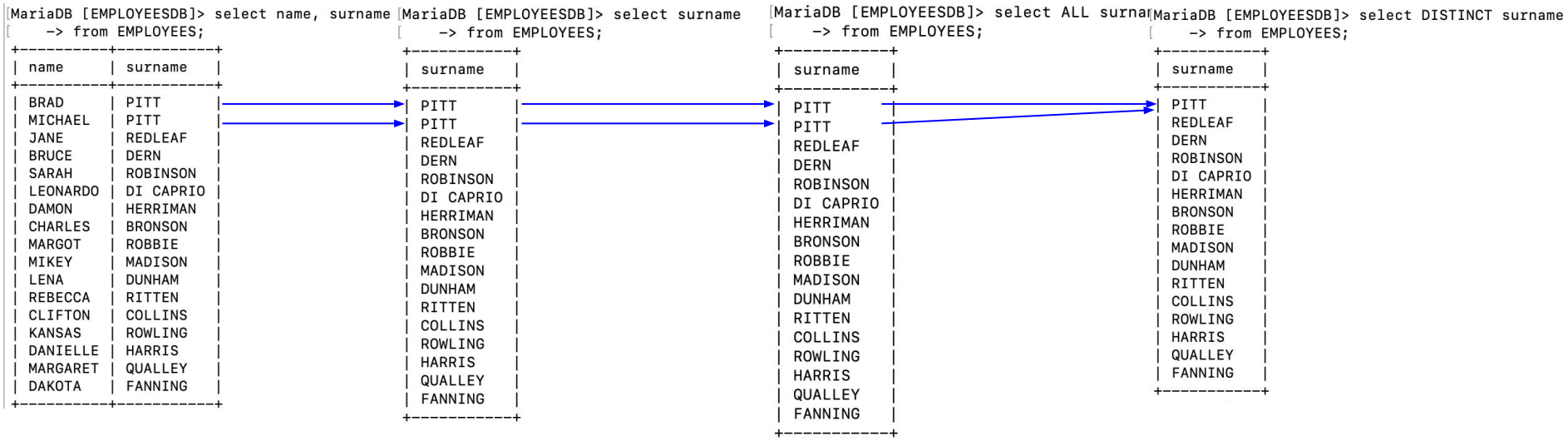
  - A_i represents an attribute
  - R_i represents a relation
  - P is a predicate.

- The **result** of an SQL query is always **relation**.



2.8. The select clause (II).

- Duplicates are allowed in query results, you can avoid them with the keyword **DISTINCT**:



1.8. The select clause (III).

- **Asterisk (*)** to denote “all attributes”

```
MariaDB [EMPLOYEESDB]> select *
    -> from DEPARTMENTS;
+-----+-------------+
| num | name        |
+-----+-------------+
|  10 | ACCOUNTING  |
|  20 | RESEARCH    |
|  30 | SALES       |
|  40 | PRODUCTION  |
+-----+-------------+
```

```
MariaDB [EMPLOYEESDB]> select surname, commission, commission + 100
    -> from EMPLOYEES;
+-----------+------------+------------------+
| surname   | commission | commission + 100 |
+-----------+------------+------------------+
| PITT      |       NULL |             NULL |
| PITT      |       NULL |             NULL |
| REDLEAF   |       NULL |             NULL |
| DERN      |        390 |              490 |
| ROBINSON  |        650 |              750 |
| DI CAPRIO |       NULL |             NULL |
| HERRIMAN  |       1020 |             1120 |
| BRONSON   |       NULL |             NULL |
| ROBBIE    |       NULL |             NULL |
| MADISON   |       NULL |             NULL |
| DUNHAM    |          0 |              100 |
| RITTEN    |       NULL |             NULL |
| COLLINS   |       NULL |             NULL |
| ROWLING   |       NULL |             NULL |
| HARRIS    |       NULL |             NULL |
| QUALLEY   |       NULL |             NULL |
| FANNING   |       NULL |             NULL |
+-----------+------------+------------------+
```

```
MariaDB [EMPLOYEESDB]> select 2 + 3, 5*8;
+-------+-----+
| 2 + 3 | 5*8 |
+-------+-----+
|     5 |  40 |
+-------+-----+
```

- You can use **literals** like attributes with **arithmetic operators** (+, –, *, and /):

```
MariaDB [EMPLOYEESDB]> select 'DEP.'
    -> from DEPARTMENTS;
+------+
| DEP. |
+------+
| DEP. |
| DEP. |
| DEP. |
| DEP. |
+------+
```

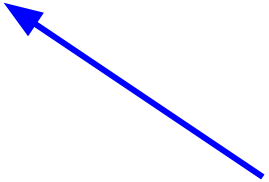
```
MariaDB [EMPLOYEESDB]> select 'DEP.', name
    -> from DEPARTMENTS;
+------+-------------+
| DEP. | name        |
+------+-------------+
| DEP. | ACCOUNTING  |
| DEP. | RESEARCH    |
| DEP. | SALES       |
| DEP. | PRODUCTION  |
+------+-------------+
```

```
MariaDB [EMPLOYEESDB]> select 'DEP.' as f1, name
    -> from DEPARTMENTS;
+------+-------------+
| f1   | name        |
+------+-------------+
| DEP. | ACCOUNTING  |
| DEP. | RESEARCH    |
| DEP. | SALES       |
| DEP. | PRODUCTION  |
+------+-------------+
```

2.8. The select clause (IV).

- Another example: salary VS Monthly Salary

```
MariaDB [EMPLOYEESDB]> select name, surname, salary, salary/12 as `Monthly salary`
    -> from EMPLOYEES;
+----------+-----------+--------+----------------+
| name     | surname   | salary | Monthly salary |
+----------+-----------+--------+----------------+
| BRAD     | PITT      | 104000 |      8666.6667 |
| JANE     | REDLEAF   | 104000 |      8666.6667 |
| BRUCE    | DERN      |  15000 |      1250.0000 |
| SARAH    | ROBINSON  |  16250 |      1354.1667 |
| LEONARDO | DI CAPRIO |  29000 |      2416.6667 |
| DAMON    | HERRIMAN  |  16000 |      1333.3333 |
| CHARLES  | BRONSON   |  30050 |      2504.1667 |
| MARGOT   | ROBBIE    |  28850 |      2404.1667 |
| MIKEY    | MADISON   |  30000 |      2500.0000 |
| LENA     | DUNHAM    |  13500 |      1125.0000 |
| REBECCA  | RITTEN    |  14300 |      1191.6667 |
| CLIFTON  | COLLINS   |  13350 |      1112.5000 |
| KANSAS   | ROWLING   |  30000 |      2500.0000 |
| DANIELLE | HARRIS    |  16900 |      1408.3333 |
| MARGARET | QUALLEY   |  28850 |      2404.1667 |
| DAKOTA   | FANNING   |  28850 |      2404.1667 |
+----------+-----------+--------+----------------+
```



Use the char ` for aliases and field names with white spaces (only MariaDB/MySQL).

2.9. The where clause (I).

- The **where** clause specifies conditions that the result will satisfy.
- Find all employees with profession “manager”:

```
MariaDB [EMPLOYEESDB]> select name, surname
    -> from EMPLOYEES
    -> where occupation = 'MANAGER';
+----------+-----------+
| name     | surname   |
+----------+-----------+
| LEONARDO | DI CAPRIO |
| CHARLES  | BRONSON   |
| MARGOT   | ROBBIE    |
| MARGARET | QUALLEY   |
| DAKOTA   | FANNING   |
+----------+-----------+
```

2.9. The where clause (II).

- Comparison results can be combined using the logical connectives **and**, **or**, and **not**
  - Find all employees with occupation manager and with salary > 20000

```
select name, surname, salary
from EMPLOYEES
where occupation = 'MANAGER'
and salary > 20000;
```

```
MariaDB [EMPLOYEESDB]> select name, surname, salary
    -> from EMPLOYEES
    -> where occupation = 'MANAGER' and salary > 10000*2;
+----------+-----------+--------+
| name     | surname   | salary |
+----------+-----------+--------+
| LEONARDO | DI CAPRIO |  29000 |
| CHARLES  | BRONSON   |  30050 |
| MARGOT   | ROBBIE    |  28850 |
| MARGARET | QUALLEY   |  28850 |
| DAKOTA   | FANNING   |  28850 |
+----------+-----------+--------+
```

- Comparisons can be applied to results of arithmetic expressions.

Let’s work!

Do the exercises:

- where_logical_expressions.

In this exercise you will use/see:

- logical expressions for the WHERE clause

2.10. The from clause (I).

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product instructor X teaches

  select *
  from EMPLOYEES, DEPARTMENTS
  - generates every possible employee – department pair, with all attributes from both relations.
- **Cartesian product** not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

2.10. The from clause (II).

```
MariaDB [EMPLOYEESDB]> select * from EMPLOYEES;
+------+-----------+----------+------------+---------+------------+--------+------------+----------+
| num  | surname   | name     | occupation | manager | begin_date | salary | commission | dept_num |
+------+-----------+----------+------------+---------+------------+--------+------------+----------+
| 1000 | PITT      | BRAD     | OWNER      |    NULL | 1984-01-01 | 104000 |       NULL |       20 |
| 7369 | REDLEAF   | JANE     | EMPLOYEE   |    8001 | 1990-12-17 | 104000 |       NULL |       20 |
| 7499 | DERN      | BRUCE    | SALESMAN   |    7698 | 1990-02-20 |  15000 |        390 |       30 |
| 7521 | ROBINSON  | SARAH    | SALESMAN   |    7782 | 1991-02-22 |  16250 |        650 |       30 |
| 7566 | DI CAPRIO | LEONARDO | MANAGER    |    1000 | 1991-04-02 |  29000 |       NULL |       20 |
| 7654 | HERRIMAN  | DAMON    | SALESMAN   |    7698 | 1991-09-29 |  16000 |       1020 |       30 |
| 7698 | BRONSON   | CHARLES  | MANAGER    |    1000 | 1991-05-01 |  30050 |       NULL |       30 |
| 7782 | ROBBIE    | MARGOT   | MANAGER    |    1000 | 1991-06-09 |  28850 |       NULL |       10 |
| 7788 | MADISON   | MIKEY    | ANALYST    |    8000 | 1991-11-09 |  30000 |       NULL |       20 |
| 7844 | DUNHAM    | LENA     | SALESMAN   |    7698 | 1991-09-08 |  13500 |          0 |       30 |
| 7876 | RITTEN    | REBECCA  | EMPLOYEE   |    7788 | 1991-09-23 |  14300 |       NULL |       20 |
| 7900 | COLLINS   | CLIFTON  | EMPLOYEE   |    8001 | 1991-12-03 |  13350 |       NULL |       30 |
| 7902 | ROWLING   | KANSAS   | ANALYST    |    8000 | 1991-12-03 |  30000 |       NULL |       20 |
| 7934 | HARRIS    | DANIELLE | EMPLOYEE   |    8001 | 1992-01-23 |  16900 |       NULL |       10 |
| 8000 | QUALLEY   | MARGARET | MANAGER    |    1000 | 1991-01-09 |  28850 |       NULL |       20 |
| 8001 | FANNING   | DAKOTA   | MANAGER    |    1000 | 1992-06-10 |  28850 |       NULL |       20 |
+------+-----------+----------+------------+---------+------------+--------+------------+----------+
```

X

```
MariaDB [EMPLOYEESDB]> select * from DEPARTMENTS;
+-----+-------------+
| num | name        |
+-----+-------------+
|  10 | ACCOUNTING  |
|  20 | RESEARCH    |
|  30 | SALES       |
|  40 | PRODUCTION  |
+-----+-------------+
```

```
MariaDB [EMPLOYEESDB]> select *
    -> from EMPLOYEES, DEPARTMENTS;
+------+-----------+----------+------------+---------+------------+--------+------------+----------+-----+-------------+
| num  | surname   | name     | occupation | manager | begin_date | salary | commission | dept_num | num | name        |
+------+-----------+----------+------------+---------+------------+--------+------------+----------+-----+-------------+
| 1000 | PITT      | BRAD     | OWNER      |    NULL | 1984-01-01 | 104000 |       NULL |       20 |  10 | ACCOUNTING  |
| 1000 | PITT      | BRAD     | OWNER      |    NULL | 1984-01-01 | 104000 |       NULL |       20 |  20 | RESEARCH    |
| 1000 | PITT      | BRAD     | OWNER      |    NULL | 1984-01-01 | 104000 |       NULL |       20 |  30 | SALES       |
| 1000 | PITT      | BRAD     | OWNER      |    NULL | 1984-01-01 | 104000 |       NULL |       20 |  40 | PRODUCTION  |
| 7369 | REDLEAF   | JANE     | EMPLOYEE   |    8001 | 1990-12-17 | 104000 |       NULL |       20 |  10 | ACCOUNTING  |
| 7369 | REDLEAF   | JANE     | EMPLOYEE   |    8001 | 1990-12-17 | 104000 |       NULL |       20 |  20 | RESEARCH    |
| 7369 | REDLEAF   | JANE     | EMPLOYEE   |    8001 | 1990-12-17 | 104000 |       NULL |       20 |  30 | SALES       |
| 7369 | REDLEAF   | JANE     | EMPLOYEE   |    8001 | 1990-12-17 | 104000 |       NULL |       20 |  40 | PRODUCTION  |
| 7499 | DERN      | BRUCE    | SALESMAN   |    7698 | 1990-02-20 |  15000 |        390 |       30 |  10 | ACCOUNTING  |
| 7499 | DERN      | BRUCE    | SALESMAN   |    7698 | 1990-02-20 |  15000 |        390 |       30 |  20 | RESEARCH    |
| 7499 | DERN      | BRUCE    | SALESMAN   |    7698 | 1990-02-20 |  15000 |        390 |       30 |  30 | SALES       |
| 7499 | DERN      | BRUCE    | SALESMAN   |    7698 | 1990-02-20 |  15000 |        390 |       30 |  40 | PRODUCTION  |
| 7521 | ROBINSON  | SARAH    | SALESMAN   |    7782 | 1991-02-22 |  16250 |        650 |       30 |  10 | ACCOUNTING  |
| 7521 | ROBINSON  | SARAH    | SALESMAN   |    7782 | 1991-02-22 |  16250 |        650 |       30 |  20 | RESEARCH    |
| 7521 | ROBINSON  | SARAH    | SALESMAN   |    7782 | 1991-02-22 |  16250 |        650 |       30 |  30 | SALES       |
| 7521 | ROBINSON  | SARAH    | SALESMAN   |    7782 | 1991-02-22 |  16250 |        650 |       30 |  40 | PRODUCTION  |
| 7566 | DI CAPRIO | LEONARDO | MANAGER    |    1000 | 1991-04-02 |  29000 |       NULL |       20 |  10 | ACCOUNTING  |
| 7566 | DI CAPRIO | LEONARDO | MANAGER    |    1000 | 1991-04-02 |  29000 |       NULL |       20 |  20 | RESEARCH    |
| 7566 | DI CAPRIO | LEONARDO | MANAGER    |    1000 | 1991-04-02 |  29000 |       NULL |       20 |  30 | SALES       |
| 7566 | DI CAPRIO | LEONARDO | MANAGER    |    1000 | 1991-04-02 |  29000 |       NULL |       20 |  40 | PRODUCTION  |
| 7654 | HERRIMAN  | DAMON    | SALESMAN   |    7698 | 1991-09-29 |  16000 |       1020 |       30 |  10 | ACCOUNTING  |
| 7654 | HERRIMAN  | DAMON    | SALESMAN   |    7698 | 1991-09-29 |  16000 |       1020 |       30 |  20 | RESEARCH    |
| 7654 | HERRIMAN  | DAMON    | SALESMAN   |    7698 | 1991-09-29 |  16000 |       1020 |       30 |  30 | SALES       |
| 7654 | HERRIMAN  | DAMON    | SALESMAN   |    7698 | 1991-09-29 |  16000 |       1020 |       30 |  40 | PRODUCTION  |
| 7698 | BRONSON   | CHARLES  | MANAGER    |    1000 | 1991-05-01 |  30050 |       NULL |       30 |  10 | ACCOUNTING  |
| 7698 | BRONSON   | CHARLES  | MANAGER    |    1000 | 1991-05-01 |  30050 |       NULL |       30 |  20 | RESEARCH    |
| 7698 | BRONSON   | CHARLES  | MANAGER    |    1000 | 1991-05-01 |  30050 |       NULL |       30 |  30 | SALES       |
| 7698 | BRONSON   | CHARLES  | MANAGER    |    1000 | 1991-05-01 |  30050 |       NULL |       30 |  40 | PRODUCTION  |
| 7782 | ROBBIE    | MARGOT   | MANAGER    |    1000 | 1991-06-09 |  28850 |       NULL |       10 |  10 | ACCOUNTING  |
| 7782 | ROBBIE    | MARGOT   | MANAGER    |    1000 | 1991-06-09 |  28850 |       NULL |       10 |  20 | RESEARCH    |
| 7782 | ROBBIE    | MARGOT   | MANAGER    |    1000 | 1991-06-09 |  28850 |       NULL |       10 |  30 | SALES       |
| 7782 | ROBBIE    | MARGOT   | MANAGER    |    1000 | 1991-06-09 |  28850 |       NULL |       10 |  40 | PRODUCTION  |
| 7788 | MADISON   | MIKEY    | ANALYST    |    8000 | 1991-11-09 |  30000 |       NULL |       20 |  10 | ACCOUNTING  |
| 7788 | MADISON   | MIKEY    | ANALYST    |    8000 | 1991-11-09 |  30000 |       NULL |       20 |  20 | RESEARCH    |
| 7788 | MADISON   | MIKEY    | ANALYST    |    8000 | 1991-11-09 |  30000 |       NULL |       20 |  30 | SALES       |
| 7788 | MADISON   | MIKEY    | ANALYST    |    8000 | 1991-11-09 |  30000 |       NULL |       20 |  40 | PRODUCTION  |
| 7844 | DUNHAM    | LENA     | SALESMAN   |    7698 | 1991-09-08 |  13500 |          0 |       30 |  10 | ACCOUNTING  |
| 7844 | DUNHAM    | LENA     | SALESMAN   |    7698 | 1991-09-08 |  13500 |          0 |       30 |  20 | RESEARCH    |
| 7844 | DUNHAM    | LENA     | SALESMAN   |    7698 | 1991-09-08 |  13500 |          0 |       30 |  30 | SALES       |
| 7844 | DUNHAM    | LENA     | SALESMAN   |    7698 | 1991-09-08 |  13500 |          0 |       30 |  40 | PRODUCTION  |
| 7876 | RITTEN    | REBECCA  | EMPLOYEE   |    7788 | 1991-09-23 |  14300 |       NULL |       20 |  10 | ACCOUNTING  |
| 7876 | RITTEN    | REBECCA  | EMPLOYEE   |    7788 | 1991-09-23 |  14300 |       NULL |       20 |  20 | RESEARCH    |
| 7876 | RITTEN    | REBECCA  | EMPLOYEE   |    7788 | 1991-09-23 |  14300 |       NULL |       20 |  30 | SALES       |
| 7876 | RITTEN    | REBECCA  | EMPLOYEE   |    7788 | 1991-09-23 |  14300 |       NULL |       20 |  40 | PRODUCTION  |
| 7900 | COLLINS   | CLIFTON  | EMPLOYEE   |    8001 | 1991-12-03 |  13350 |       NULL |       30 |  10 | ACCOUNTING  |
| 7900 | COLLINS   | CLIFTON  | EMPLOYEE   |    8001 | 1991-12-03 |  13350 |       NULL |       30 |  20 | RESEARCH    |
| 7900 | COLLINS   | CLIFTON  | EMPLOYEE   |    8001 | 1991-12-03 |  13350 |       NULL |       30 |  30 | SALES       |
| 7900 | COLLINS   | CLIFTON  | EMPLOYEE   |    8001 | 1991-12-03 |  13350 |       NULL |       30 |  40 | PRODUCTION  |
| 7902 | ROWLING   | KANSAS   | ANALYST    |    8000 | 1991-12-03 |  30000 |       NULL |       20 |  10 | ACCOUNTING  |
| 7902 | ROWLING   | KANSAS   | ANALYST    |    8000 | 1991-12-03 |  30000 |       NULL |       20 |  20 | RESEARCH    |
| 7902 | ROWLING   | KANSAS   | ANALYST    |    8000 | 1991-12-03 |  30000 |       NULL |       20 |  30 | SALES       |
| 7902 | ROWLING   | KANSAS   | ANALYST    |    8000 | 1991-12-03 |  30000 |       NULL |       20 |  40 | PRODUCTION  |
| 7934 | HARRIS    | DANIELLE | EMPLOYEE   |    8001 | 1992-01-23 |  16900 |       NULL |       10 |  10 | ACCOUNTING  |
| 7934 | HARRIS    | DANIELLE | EMPLOYEE   |    8001 | 1992-01-23 |  16900 |       NULL |       10 |  20 | RESEARCH    |
| 7934 | HARRIS    | DANIELLE | EMPLOYEE   |    8001 | 1992-01-23 |  16900 |       NULL |       10 |  30 | SALES       |
| 7934 | HARRIS    | DANIELLE | EMPLOYEE   |    8001 | 1992-01-23 |  16900 |       NULL |       10 |  40 | PRODUCTION  |
| 8000 | QUALLEY   | MARGARET | MANAGER    |    1000 | 1991-01-09 |  28850 |       NULL |       20 |  10 | ACCOUNTING  |
| 8000 | QUALLEY   | MARGARET | MANAGER    |    1000 | 1991-01-09 |  28850 |       NULL |       20 |  20 | RESEARCH    |
| 8000 | QUALLEY   | MARGARET | MANAGER    |    1000 | 1991-01-09 |  28850 |       NULL |       20 |  30 | SALES       |
| 8000 | QUALLEY   | MARGARET | MANAGER    |    1000 | 1991-01-09 |  28850 |       NULL |       20 |  40 | PRODUCTION  |
| 8001 | FANNING   | DAKOTA   | MANAGER    |    1000 | 1992-06-10 |  28850 |       NULL |       20 |  10 | ACCOUNTING  |
| 8001 | FANNING   | DAKOTA   | MANAGER    |    1000 | 1992-06-10 |  28850 |       NULL |       20 |  20 | RESEARCH    |
| 8001 | FANNING   | DAKOTA   | MANAGER    |    1000 | 1992-06-10 |  28850 |       NULL |       20 |  30 | SALES       |
| 8001 | FANNING   | DAKOTA   | MANAGER    |    1000 | 1992-06-10 |  28850 |       NULL |       20 |  40 | PRODUCTION  |
+------+-----------+----------+------------+---------+------------+--------+------------+----------+-----+-------------+
```

2.11. Select clause examples.

- Find the names, surname and department name of all the employees:
  - **select** EMPLOYEES.name, EMPLOYEES.surname, DEPARTMENTS.name
    from EMPLOYEES, DEPARTMENTS
    where EMPLOYEES.dept_num = DEPARTMENTS.num

- Find the names, surname and department name of all the employees who earn more than $25000:
  - **select** EMPLOYEES.name, EMPLOYEES.surname, DEPARTMENTS.name
    from EMPLOYEES, DEPARTMENTS
    where EMPLOYEES.dept_num = DEPARTMENTS.num **and**
    EMPLOYEES.salary > 25000;

2.12. The rename operation.

- The SQL allows renaming relations and attributes using the as clause:

  old-name **as** new-name

- Find the name, surname and department name of all the employees:
  - **select** E.name, E.surname, D.name AS dept_name
    from EMPLOYEES AS E, DEPARTMENTS AS D
    where E.dept_num = D.num
- Keyword **as** is optional and may be omitted

  EMPLOYEES **as** E ≡ EMPLOYEES E

2.13. String operations (I).

- SQL includes a string-matching operator for comparisons on character strings. The operator **like** uses patterns that are described using two special characters:
  - **percent (%)**. The **%** character matches any substring.
  - **underscore (_).** The _ character matches any character.
- Find the names of all employees whose name includes the substring “ad”.
  - **select** name, surname
    from EMPLOYEES
    where name **like** '%ad%'
- Match the string “100%”
  - **like** ‘100 \%' **escape** '\'

in that above we use backslash (\) as the escape character.

2.13. String operations (II).

- Patterns are case sensitive.
- Pattern matching examples:
  - ‘RO%’ matches any string beginning with “RO”.
  - ‘%ACO%’ matches any string containing “ACO” as a substring.
  - ‘_ _ _’ matches any string of exactly three characters.
  - ‘_ _ _ %’ matches any string of at least three characters.
- SQL supports a variety of string operations such as
  - **concatenation**: using CONCAT(string1, string2, ...)
  - converting from **upper to lower case** (and vice versa). LOWER(string1) or UPPER(string1).
  - finding string length, extracting substrings, etc.
- SQL supports a variety of date functions to transform them to strings, such as
  - YEAR
  - MONTHNAME
  - etc.

2.14. Ordering the display of tuples.

- List in alphabetic order the names of all instructors
  - **select distinct** name
    from EMPLOYEES
    order by name
- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** name **desc**
- Can sort on multiple attributes
  - Example: **order by** dept_name **desc**, name **asc**

2.15. Where clause predicates.

- SQL includes a **between** comparison operator
- Example: Find the names and surnames of all employees with salary between $10,000 and $20,000 (that is, $10,000 and $20,000 included)
    - **select** name, surname
      from EMPLOYEES
      where salary **between** 10000 **and** 20000
- JOIN between two tables (foreign keys are the glue!):
    - **select** E.name, E.surname, D.name
      from EMPLOYEES E, DEPARTMENTS D
      where EMPLOYEES.dept_num = DEPARTMENTS.num

2.16. Joined Relations (I).

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match under some conditions.
- The join operations are typically used as subquery expressions in the **from** clause.
- ANSI-standard SQL specifies **five types of JOIN**: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER and CROSS. As a special case, a table (base table, view, or joined table) can JOIN to itself in a **self-join**.

2.16. Joined Relations (II).

- **JOIN example:**
  - **select** E.name, E.surname, D.name as deptname
    from EMPLOYEES as E, DEPARTMENTS as D
    where E.dept_num = D.num;

```
CREATE TABLE `EMPLOYEES` (
  `num` int(11) NOT NULL,
  `surname` varchar(50) NOT NULL,
  `name` varchar(50) NOT NULL,
  `occupation` varchar(30) DEFAULT NULL,
  `manager` int(11) DEFAULT NULL,
  `begin_date` date DEFAULT NULL,
  `salary` int(11) DEFAULT NULL,
  `commission` int(11) DEFAULT NULL,
  `dept_num` int(11) DEFAULT NULL,
  CONSTRAINT `EMPLOYEES_pk` PRIMARY KEY (`num`),
  CONSTRAINT `EMPLOYEES_ibfk_1` FOREIGN KEY (`dept_num`) REFERENCES `DEPARTMENTS` (`num`),
  CONSTRAINT `EMPLOYEES_ibfk_2` FOREIGN KEY (`manager`) REFERENCES `EMPLOYEES` (`num`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
CREATE TABLE `DEPARTMENTS` (
  `num` int(11) NOT NULL,
  `name` varchar(30) NOT NULL,
   CONSTRAINT `DEPARTMENTS_pk` PRIMARY KEY
(`num`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

2.16. Joined Relations (III).

- Reflexive relationship:

EMPLOYEES (num, name, surname, profession, manager,
begin_date, salary, commission, dept_num)

```
CREATE TABLE `EMPLOYEES` (
  `num` int(11) NOT NULL,
  `surname` varchar(50) NOT NULL,
  `name` varchar(50) NOT NULL,
  `occupation` varchar(30) DEFAULT NULL,
  `manager` int(11) DEFAULT NULL,
  `begin_date` date DEFAULT NULL,
  `salary` int(11) DEFAULT NULL,
  `commission` int(11) DEFAULT NULL,
  `dept_num` int(11) DEFAULT NULL,
  CONSTRAINT `EMPLOYEES_pk` PRIMARY KEY (`num`),
  CONSTRAINT `EMPLOYEES_ibfk_1` FOREIGN KEY (`dept_num`)
REFERENCES `DEPARTMENTS` (`num`),
  CONSTRAINT `EMPLOYEES_ibfk_2` FOREIGN KEY (`manager`)
REFERENCES `EMPLOYEES` (`num`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Find the manager of “ROBBIE”.

```
select E.name as employee_name, E.surname as employee_surname,
        E.occupation as employee_occupation,
        M.name as employee_name, M.surname as employee_surname,
        M.occupation as manager_occupation
from EMPLOYEES as E, EMPLOYEES as M
where E.manager = M.num and
        E.surname = 'ROBBIE';
```

```
+---------------+-----------------+---------------------+---------------+-----------------+--------------------+
| employee_name | employee_surname | employee_occupation | employee_name | employee_surname | manager_occupation |
+---------------+-----------------+---------------------+---------------+-----------------+--------------------+
| MARGOT        | ROBBIE          | MANAGER             | BRAD          | PITT            | OWNER              |
+---------------+-----------------+---------------------+---------------+-----------------+--------------------+
```

Let’s work!

Do the exercises:

- C02_queries01.

In these exercises you will use/see:

- select (also select distinct)
- order by (asc/desc)
- where (select conditions)
- string patterns (% and _) and like
- IN and NOT IN
- functions LOWER, UPPER, CONCAT, YEAR and MONTHNAME

2.17. Duplicates.

```
select distinct YEAR(begin_date)
from EMPLOYEES
order by begin_date;
```

YEAR(begin_date)
1984
1990
1991
1992

```
select YEAR(begin_date)
from EMPLOYEES
order by begin_date;
```

YEAR(begin_date)
1984
1990
1990
1991
1991
1991
1991
1991
1991
1991
1991
1991
1991
1991
1991
1991
1992
1992

2.18. Set operations (I).

- Employees who start working in 1984 or 1990:
  (select name, surname, begin_date from EMPLOYEES where YEAR(begin_date) = 1984)
  UNION
  (select name, surname, begin_date from EMPLOYEES where YEAR(begin_date) = 1990);
- Employees who start working in 1991 and who are managers of other employees:
  (select num from EMPLOYEES where YEAR(begin_date) = 1991)
  INTERSECT
  (select distinct manager from EMPLOYEES);
- Employees who start working in 1991 and **who are not managers** of other employees:
  (select num from EMPLOYEES where YEAR(begin_date) = 1991)
  EXCEPT
  (select distinct manager from EMPLOYEES);

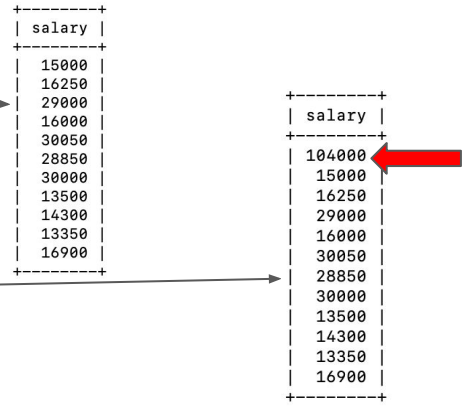
name	surname	begin_date
BRAD	PITT	1984-01-01
JANE	REDLEAF	1990-12-17
BRUCE	DERN	1990-02-20

num
7698
7782
7788
8000

num
7521
7566
7654
7844
7876
7900
7902

2.18. Set operations (II).

- Find the salaries of all employees that are less than the largest salary:
  - **SELECT DISTINCT** E.salary
    FROM EMPLOYEES E, EMPLOYEES F
    WHERE E.salary < F.salary;
- Find all the salaries of all employees:
  - **SELECT DISTINCT** salary
    FROM EMPLOYEES;
- Find the largest salary of all EMPLOYEES:
  - (select “second query”)
    EXCEPT
    (select “first query”)



salary
15000
16250
29000
16000
30050
28850
30000
13500
14300
13350
16900

salary
104000
15000
16250
29000
16000
30050
28850
30000
13500
14300
13350
16900

```
MariaDB [EMPLOYEESDB]> (SELECT DISTINCT salary
    -> FROM EMPLOYEES)
    -> EXCEPT
    -> (SELECT DISTINCT E.salary
    -> FROM EMPLOYEES E, EMPLOYEES F
    -> WHERE E.salary < F.salary);
+--------+
| salary |
+--------+
| 104000 |
+--------+
1 row in set (0.001 sec)
```

SELECT DISTINCT salary
FROM EMPLOYEES
ORDER BY salary DESC
LIMIT 1;

2.18. Set operations (III).

- Set operations **union**, **intersect**, and **except**:
  - Each of the above operations automatically eliminates duplicates
- To retain all duplicates use the corresponding multiset versions **union all**, **intersect all** and **except all**.
- Suppose a tuple occurs m times in the relation r and n times in the relation s, then, it occurs:
  - m + n times in r **union all** s
  - min(m,n) times in r **intersect all** s
  - max(0, m – n) times in r **except all** s

2.19. NULL values (I).

- It is possible for tuples to have a null value for some of their attributes.
- null signifies an unknown value, that a **value does not exist** or **value not applicable**.
- **The result of any arithmetic expression involving null is null**
  - Example: 5 + null returns null
- The predicate **is null** can be used to check for null values.
- Example: Find all employees whose salary is null.
  - **select** name, surname
    from EMPLOYEES
    where salary is null

2.19. NULL values (II).

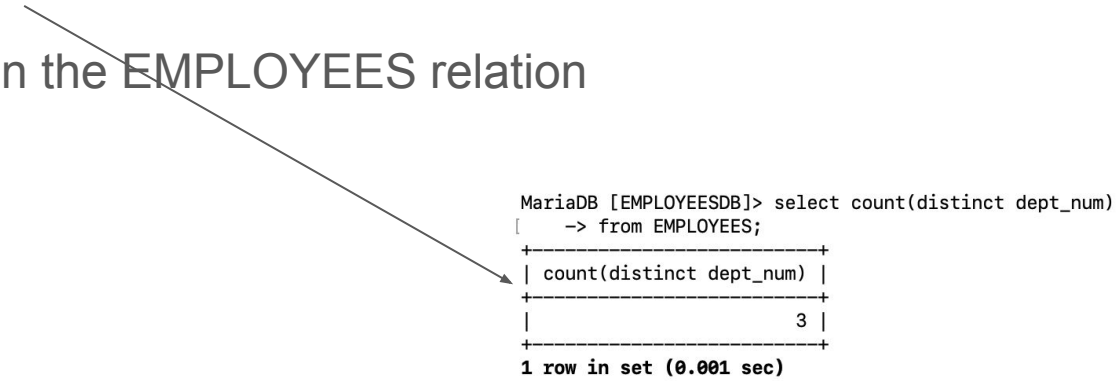
- Three values – true, false, unknown
- Any comparison with null returns unknown
  - Example: 5 < null or null <> null or null = null
- Three-valued logic using the value unknown:
  - OR: (unknown or true) = true,
    (unknown or false) = unknown
    (unknown or unknown) = unknown
  - AND: (true and unknown) = unknown,
    (false and unknown) = false,
    (unknown and unknown) = unknown
  - NOT: (not unknown) = unknown
  - “P is unknown” evaluates to true if predicate P evaluates to unknown
- Result of where clause predicate is treated as false if it evaluates to unknown.

2.20. Aggregate Functions (I).

- These functions operate on the multiset of values of a column of a relation, and return a value.
  - **avg**: average value
  - **min**: minimum value
  - **max**: maximum value
  - **sum**: sum of values
  - **count**: number of values

2.20. Aggregate Functions (II).

- Find the average salary of employees in department with num 10:
  - **select avg** (salary)
    from EMPLOYEES
    where dept_num = 10;
- Find the total number of departments with employees working in it:
  - **select count**(**distinct** dept_num)
    from EMPLOYEES;
- Find the number of tuples in the EMPLOYEES relation
  - **select count** (*)
    from EMPLOYEES;



2.20. Aggregate Functions (III).

- Find the minimum and maximum salary of employees in department with num 10:
  - **select min**(salary), **max**(salary)
    from EMPLOYEES
    where dept_num = 10;

```
+-------------+-------------+
| min(salary) | max(salary) |
+-------------+-------------+
|       16900 |       28850 |
+-------------+-------------+
1 row in set (0.000 sec)
```

  - **select min**(salary), **max**(salary)
    from EMPLOYEES;

```
+-------------+-------------+
| min(salary) | max(salary) |
+-------------+-------------+
|       13350 |      104000 |
+-------------+-------------+
1 row in set (0.001 sec)
```

2.20. Aggregate Functions: Group By (I).

- Find the average salary of employees in each department:
  - **select** D.name, avg(E.salary)
    from EMPLOYEES E, DEPARTMENTS D
    where E.dept_num = D.num
    group by D.name;

```
MariaDB [EMPLOYEESDB]> select *
    -> from DEPARTMENTS;
+-----+------------+
| num | name       |
+-----+------------+
|  10 | ACCOUNTING |
|  20 | RESEARCH   |
|  30 | SALES      |
|  40 | PRODUCTION |
+-----+------------+
```

```
MariaDB [EMPLOYEESDB]> select
    -> E.num, E.name, E.surname,
    -> E.salary, D.name
    -> from EMPLOYEES E, DEPARTMENTS D
    -> where E.dept_num = D.num
    -> order by D.name;
+------+----------+-----------+--------+------------+
| num  | name     | surname   | salary | name       |
+------+----------+-----------+--------+------------+
| 7782 | MARGOT   | ROBBIE    |  28850 | ACCOUNTING |
| 7934 | DANIELLE | HARRIS    |  16900 | ACCOUNTING |
| 7566 | LEONARDO | DI CAPRIO |  29000 | RESEARCH   |
| 8001 | DAKOTA   | FANNING   |  28850 | RESEARCH   |
| 7788 | MIKEY    | MADISON   |  30000 | RESEARCH   |
| 7876 | REBECCA  | RITTEN    |  14300 | RESEARCH   |
| 1000 | BRAD     | PITT      | 104000 | RESEARCH   |
| 7369 | JANE     | REDLEAF   | 104000 | RESEARCH   |
| 7902 | KANSAS   | ROWLING   |  30000 | RESEARCH   |
| 8000 | MARGARET | QUALLEY   |  28850 | RESEARCH   |
| 7654 | DAMON    | HERRIMAN  |  16000 | SALES      |
| 7698 | CHARLES  | BRONSON   |  30050 | SALES      |
| 7844 | LENA     | DUNHAM    |  13500 | SALES      |
| 7900 | CLIFTON  | COLLINS   |  13350 | SALES      |
| 7499 | BRUCE    | DERN      |  15000 | SALES      |
| 7521 | SARAH    | ROBINSON  |  16250 | SALES      |
+------+----------+-----------+--------+------------+
```

```
+------------+----------------+
| name       | avg(E.salary)  |
+------------+----------------+
| ACCOUNTING |     22875.0000 |
| RESEARCH   |     46125.0000 |
| SALES      |     17358.3333 |
+------------+----------------+
```

2.20. Aggregate Functions: Group By (II).

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list
  - /* erroneous query */

```
select D.name, avg(E.salary)
from EMPLOYEES E, DEPARTMENTS D
where E.dept_num = D.num;
```

```
+----------+---------------+
| name     | avg(E.salary) |
+----------+---------------+
| RESEARCH |    32431.2500 |
+----------+---------------+
1 row in set (0.001 sec)
```

The query was runned in MariaDB 10.3. In other DBMS you may get an error if you don't use GROUP BY.

  - /* right query */

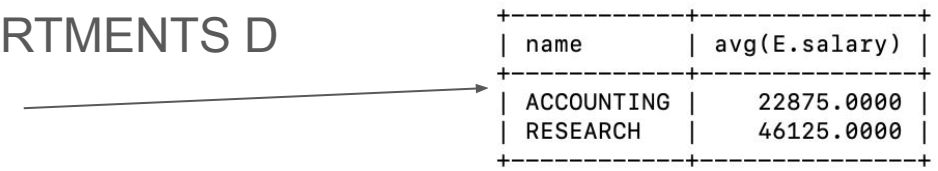
```
select D.name, avg(E.salary)
from EMPLOYEES E, DEPARTMENTS D
where E.dept_num = D.num
group by D.name;
```

```
+------------+---------------+
| name       | avg(E.salary) |
+------------+---------------+
| ACCOUNTING |    22875.0000 |
| RESEARCH   |    46125.0000 |
| SALES      |    17358.3333 |
+------------+---------------+
3 rows in set (0.000 sec)
```

2.20. Aggregate Functions: Having Clause (I).

- Find the names and average salaries of all departments whose average salary is greater than 21000:

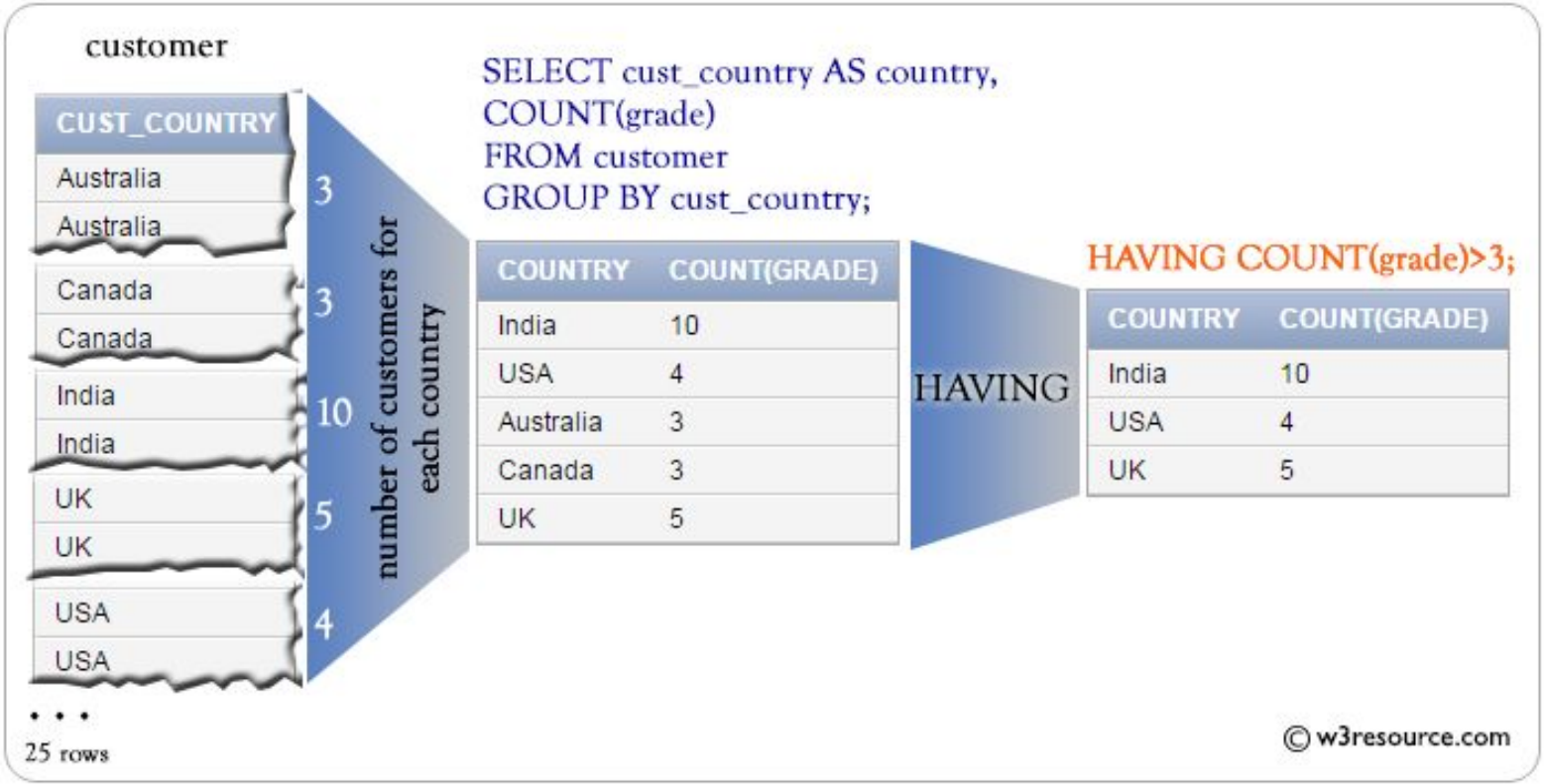
```
select D.name, avg(E.salary)
from EMPLOYEES E, DEPARTMENTS D
where E.dept_num = D.num
group by D.name
having avg(salary) > 21000;
```



```
+------------+---------------+
| name       | avg(E.salary) |
+------------+---------------+
| ACCOUNTING |    22875.0000 |
| RESEARCH   |    46125.0000 |
+------------+---------------+
```

- Note: Predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups.

2.20. Aggregate Functions: Having Clause (II).



2.20. Aggregate Functions: Null Values.

- Total all salaries
  - **select sum**(salary)
    from EMPLOYEES
- Above statement ignores null amounts
- Result is null if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

```
MariaDB [EMPLOYEESDB]> select count(commission) from EMPLOYEES;
+-------------------+
| count(commission) |
+-------------------+
|                 4 |
+-------------------+
1 row in set (0.000 sec)
```

2.21. Nested Subqueries.

- SQL provides a mechanism for the nesting of subqueries. A **subquery** is a select-from-where expression that is nested within another query.
- The nesting can be done in the following SQL query

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P
  as follows:
  - CASE 1: A_i can be replaced be a subquery that generates a single value.
  - CASE 2: r_i can be replaced by any valid subquery
  - CASE 3: P can be replaced with an expression of the form:

    B <operation> (subquery)

    Where B is an attribute and <operation> to be defined later.

2.21. Nested Subqueries (II).

Subquery example CASE 1:

- Show name and surname of employees with their department name:
  - **select** E.name, E.surname,
    (**select** D.name
    from DEPARTMENTS D
    where E.dept_num = D.num) **as** dept_name
    from EMPLOYEES E;

name	surname	dept_name
BRAD	PITT	RESEARCH
JANE	REDLEAF	RESEARCH
BRUCE	DERN	SALES
SARAH	ROBINSON	SALES
LEONARDO	DI CAPRIO	RESEARCH
DAMON	HERRIMAN	SALES
CHARLES	BRONSON	SALES
MARGOT	ROBBIE	ACCOUNTING
MIKEY	MADISON	RESEARCH
LENA	DUNHAM	SALES
REBECCA	RITTEN	RESEARCH
CLIFTON	COLLINS	SALES
KANSAS	ROWLING	RESEARCH
DANIELLE	HARRIS	ACCOUNTING
MARGARET	QUALLEY	RESEARCH
DAKOTA	FANNING	RESEARCH

2.21. Nested Subqueries (III).

Subquery example CASE 2:

- Show name, surname, salary and department number from employees but also the average salary in their department:
  - **select** E.name, E.surname, E.salary,
    E.dept_num, S.avgsalary
    from EMPLOYEES **as** E,
    (select dept_num,
    AVG(salary) **as** avgsalary
    from EMPLOYEES
    group by dept_num) **as** S
    where E.dept_num = S.dept_num;

name	surname	salary	dept_num	avgsalary
BRAD	PITT	104000	20	46125.0000
JANE	REDLEAF	104000	20	46125.0000
BRUCE	DERN	15000	30	17358.3333
SARAH	ROBINSON	16250	30	17358.3333
LEONARDO	DI CAPRIO	29000	20	46125.0000
DAMON	HERRIMAN	16000	30	17358.3333
CHARLES	BRONSON	30050	30	17358.3333
MARGOT	ROBBIE	28850	10	22875.0000
MIKEY	MADISON	30000	20	46125.0000
LENA	DUNHAM	13500	30	17358.3333
REBECCA	RITTEN	14300	20	46125.0000
CLIFTON	COLLINS	13350	30	17358.3333
KANSAS	ROWLING	30000	20	46125.0000
DANIELLE	HARRIS	16900	10	22875.0000
MARGARET	QUALLEY	28850	20	46125.0000
DAKOTA	FANNING	28850	20	46125.0000

2.21. Nested Subqueries (IV).

Subquery example CASE 2:

- Find the the maximum number of employees of all the departments:
  - **select** MAX(T.num_employees)

    from (

    select E.dept_num, count(*) as num_employees

    from EMPLOYEES E

    group by E.dept_num

) as T;

```
+----------------------+
| MAX(T.num_employees) |
+----------------------+
|                    8 |
+----------------------+
1 row in set (0.001 sec)
```

2.21. Nested Subqueries (V).

Subquery example CASE 3:

- Show the departments whose average salary is greater than the average of salaries of all employees:
  - **select** dept_num, avg(salary)
    from EMPLOYEES
    group by dept_num
    having avg(salary) > (**select** avg(salary) **from** EMPLOYEES);

```
+----------+-------------+
| dept_num | avg(salary) |
+----------+-------------+
|       20 |  46125.0000 |
+----------+-------------+
```

2.21. Nested Subqueries (VI).

Subquery example CASE 3:

- List all employees who are managers:
  - select name, surname

    from EMPLOYEES

    where num IN

        (select distinct manager

        from EMPLOYEES);

name	surname
BRAD	PITT
CHARLES	BRONSON
MARGOT	ROBBIE
MIKEY	MADISON
MARGARET	QUALLEY
DAKOTA	FANNING

6 rows in set (0.004 sec)

2.21. Nested Subqueries (VII): More examples.

```
/*Type 1: Subquery in SELECT clause*
I get an error if I use avgsal in the `diff` field,
so I copy again the query...
*/
select E.num, E.name, E.surname, E.salary,
       (select avg(E2.salary)
        from EMPLOYEES E2
        where E2.dept_num=E.dept_num
        group by E2.dept_num) as avgsal,
       E.salary - (select avg(E2.salary)
                   from EMPLOYEES E2
                   where E2.dept_num=E.dept_num
                   group by E2.dept_num) as diff
from EMPLOYEES as E;
```

```
/*Type 2: Subquery in FROM clause*/
select E.num, E.name, E.surname, E.salary,
       T.avgsal,E.salary - T.avgsal as diff
from EMPLOYEES as E,
     (select dept_num, avg(salary) as avgsal
     from EMPLOYEES
     group by dept_num) as T
where E.dept_num=T.dept_num;
```

```
+------+----------+-----------+--------+------------+-------------+
| num  | name     | surname   | salary | avgsal     | diff        |
+------+----------+-----------+--------+------------+-------------+
| 1000 | BRAD     | PITT      | 104000 | 46125.0000 |  57875.0000 |
| 7369 | JANE     | REDLEAF   | 104000 | 46125.0000 |  57875.0000 |
| 7499 | BRUCE    | DERN      |  15000 | 17358.3333 |  -2358.3333 |
| 7521 | SARAH    | ROBINSON  |  16250 | 17358.3333 |  -1108.3333 |
| 7566 | LEONARDO | DI CAPRIO |  29000 | 46125.0000 | -17125.0000 |
| 7654 | DAMON    | HERRIMAN  |  16000 | 17358.3333 |  -1358.3333 |
| 7698 | CHARLES  | BRONSON   |  30050 | 17358.3333 |  12691.6667 |
| 7782 | MARGOT   | ROBBIE    |  28850 | 22875.0000 |   5975.0000 |
| 7788 | MIKEY    | MADISON   |  30000 | 46125.0000 | -16125.0000 |
| 7844 | LENA     | DUNHAM    |  13500 | 17358.3333 |  -3858.3333 |
| 7876 | REBECCA  | RITTEN    |  14300 | 46125.0000 | -31825.0000 |
| 7900 | CLIFTON  | COLLINS   |  13350 | 17358.3333 |  -4008.3333 |
| 7902 | KANSAS   | ROWLING   |  30000 | 46125.0000 | -16125.0000 |
| 7934 | DANIELLE | HARRIS    |  16900 | 22875.0000 |  -5975.0000 |
| 8000 | MARGARET | QUALLEY   |  28850 | 46125.0000 | -17275.0000 |
| 8001 | DAKOTA   | FANNING   |  28850 | 46125.0000 | -17275.0000 |
+------+----------+-----------+--------+------------+-------------+
16 rows in set (0.001 sec)
```

2.22. Set comparison: “some” clause (I).

- Find names of employees with salary greater than that of some (at least one) employees in the department number 10:

```
select distinct F.name, F.surname, F.salary
from EMPLOYEES as F, EMPLOYEES as E
where F.salary > E.salary and E.dept_num = 10;
```

- Same query using > **some** clause:

```
select name, surname, salary
from EMPLOYEES
where salary > some (select salary
                     from EMPLOYEES
                     where dept_num = 10);
```

name	surname	salary
BRAD	PITT	104000
JANE	REDLEAF	104000
LEONARDO	DI CAPRIO	29000
CHARLES	BRONSON	30050
MARGOT	ROBBIE	28850
MIKEY	MADISON	30000
KANSAS	ROWLING	30000
MARGARET	QUALLEY	28850
DAKOTA	FANNING	28850

2.22. Set comparison: “some” clause (II).

(5 < **some** | 0 / 5 / 6 |) = true (read: 5 < some tuple in the relation)

(5 < **some** | 0 / 5 |) = false

(5 = **some** | 0 / 5 |) = true

(5 ≠ **some** | 0 / 5 |) = true (since 0 ≠ 5)

(= **some**) ≡ **in**

However, (≠ **some**) ≢ **not in**

Source of the picture: Abraham Silberschatz, Henry F. Korth and S. Sudarshan. Database System Concepts

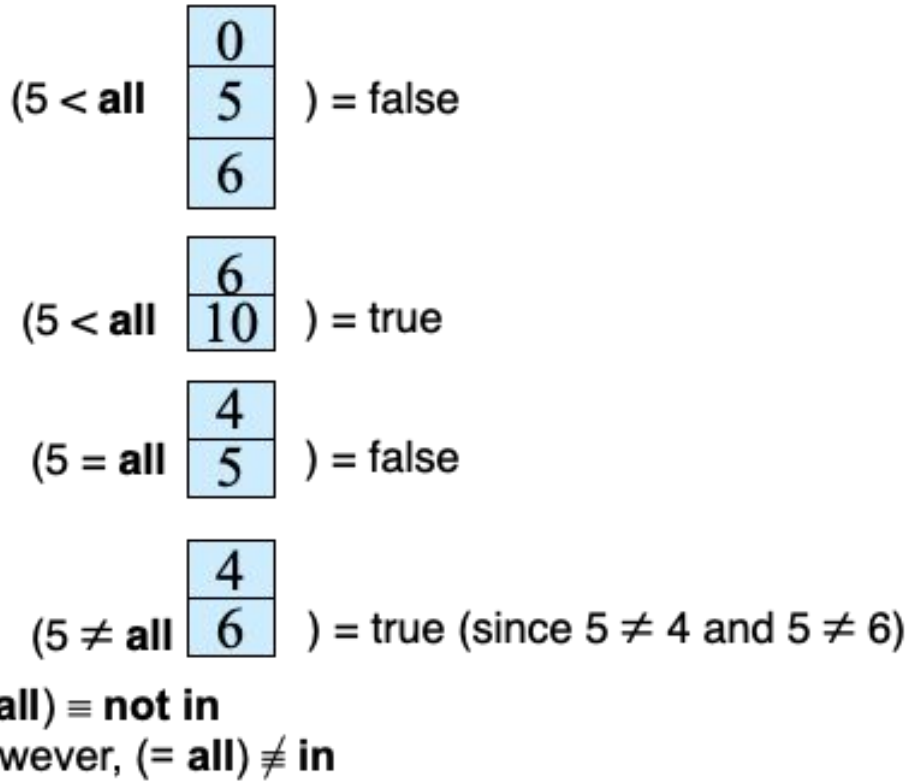
2.23. Set comparison: “all” clause (I).

- Find names, salary and department number of all employees whose salary is greater than that of all employees in the department number 10:

```
select name, surname, salary, dept_num
from EMPLOYEES
where salary > all (select salary
                    from EMPLOYEES
                    where dept_num = 10);
```

name	surname	salary	dept_num
BRAD	PITT	104000	20
JANE	REDLEAF	104000	20
LEONARDO	DI CAPRIO	29000	20
CHARLES	BRONSON	30050	30
MIKEY	MADISON	30000	20
KANSAS	ROWLING	30000	20

2.23. Set comparison: “all” clause (II).



Source of the picture: Abraham Silberschatz, Henry F. Korth and S. Sudarshan. Database System Concepts

2.24. Set comparison: exists clause.

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

2.24. Set comparison: exists clause (II).

- Yet another way of specifying the query “Find the salaries of all employees that are less than the largest salary”

```
SELECT DISTINCT E.salary
FROM EMPLOYEES E
WHERE
EXISTS
        (SELECT *
        FROM EMPLOYEES F
        WHERE
        E.salary < F.salary)
```

- **Correlation name** – variable E in the outer query.
- **Correlated subquery** – the inner query.

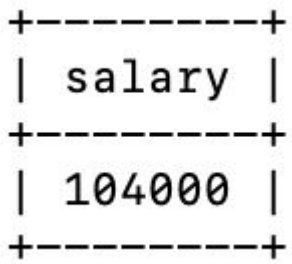
```
+--------+
| salary |
+--------+
|  15000 |
|  16250 |
|  29000 |
|  16000 |
|  30050 |
|  28850 |
|  30000 |
|  13500 |
|  14300 |
|  13350 |
|  16900 |
+--------+
```

```
SELECT DISTINCT
salary
FROM EMPLOYEES;
```

```
+--------+
| salary |
+--------+
| 104000 |
|  15000 |
|  16250 |
|  29000 |
|  16000 |
|  30050 |
|  28850 |
|  30000 |
|  13500 |
|  14300 |
|  13350 |
|  16900 |
+--------+
```

2.24. Set comparison: exists clause (III).

- **SELECT DISTINCT** E.salary
  FROM EMPLOYEES E
  WHERE
  NOT EXISTS
  (**SELECT** *
  FROM EMPLOYEES F
  WHERE
  E.salary < F.salary);



2.25. Set comparison: unique clause.

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
- The **unique** construct evaluates to “true” if a given subquery contains no duplicates.
- Maximum salary:

```
SELECT E.salary
FROM EMPLOYEES E
WHERE
UNIQUE
        (SELECT *
        FROM EMPLOYEES F
        WHERE
        E.salary < F.salary);
```

Does NOT work in MySQL/MariaDB. More info about UNIQUE here.

Let’s work!

Do the exercises:

- imdb_small.

2.26. Modification of the database with subqueries.

- **Deletion** of tuples from a given relation.
- **Insertion** of new tuples into a given relation
- **Updating** of values in some tuples in a given relation

2.27. Deletion with subqueries (I).

- Delete all employees

```
delete from EMPLOYEES;
```

- Delete all employees from the Sales department:

```
delete from EMPLOYEES
where dept_num = 30;
```

```
delete from EMPLOYEES
where dept_num = (select num
                  from DEPARTMENTS
                  where name='SALES');
```

- Delete all tuples in the *EMPLOYEES* relation for those instructors associated with a department located in MADRID.

```
delete from EMPLOYEES
where dept_num IN (select num
                   from DEPARTMENTS D, TOWNS T
                   where T.code = D.town_code and
                   T.name='MADRID');
```

2.27. Deletion with subqueries (II).

- Delete all employees whose salary is less than the average salary:

  delete from *EMPLOYEES*
  where *salary* < (**select avg** (*salary*)
  from *EMPLOYEES*);

  - ❏ Problem: as we delete tuples from deposit, the average salary changes
  - ❏ Solution used in SQL:

    1. First, compute **avg** (salary) and find all tuples to delete
    2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

2.28. Insertion with subqueries (I).

- Add a new tuple to *course*

  insert into *DEPARTMENTS*
  values (50, 'FINANCES', 'VFQ');
- or equivalently

  insert into *DEPARTMENTS* (*num*, *name*, *town_code*)
  values (50, 'FINANCES', 'VFQ');
- Add a new tuple to *EMPLOYEES* with *town_code* set to null

  insert into *DEPARTMENTS* (*num*, *name*, *town_code*)
  values (50, 'FINANCES', NULL);

2.28. Insertion with subqueries (II).

- Add all employees with professions manager to the table MANAGERS:

```
insert into MANAGERS
    select num, name, surname, start_date, salary, commission, dept_num, occu_code
    from   EMPLOYEES
   where occu_code = 'MAN'
```

- The **select from where** statement is evaluated fully before any of its results are inserted into the relation.

  Otherwise queries like

  insert into *table*1 **select** * **from** *table*1

  would cause problem

2.29. Updates with subqueries (I).

- Increase salaries and comissions of employees whose salary is over $3,000 by 3%, and all others by a 5%:
  - Write two update statements:

```
update EMPLOYEES
set salary = salary * 1.03, commission = commission * 1.03
where salary > 3000;
update EMPLOYEES
set salary = salary * 1.05, commission = commission * 1.05
where salary <= 3000;
```

  - The order is important
  - Can be done better using the case statement (next slide)

2.29. Updates with subqueries (II).

- Same query as before but with case statement

```
update EMPLOYEES
        set commission = case
                        when salary <= 3000 then commission * 1.05
                          else commission * 1.03
                    end,
        salary = case
                        when salary <= 3000 then salary * 1.05
                          else salary * 1.03
                    end;
```

https://mariadb.com/kb/en/library/case-operator/
https://www.techonthenet.com/mariadb/functions/case.php

2.29. Updates with su

```
SELECT F.dept_num, AVG(F.salary)
FROM EMPLOYEES F
WHERE
F.salary <= 1800
GROUP BY dept_num;
```

dept_num	AVG(F.salary)
10	1690.0000
20	1170.0000
30	1482.0000

- Update salaries for the employees with salary smaller than 1200 to the average salary of the employees with salary smaller or equal to 1800:

```
UPDATE EMPLOYEES E
SET E.salary = (SELECT AVG(salary)
                FROM EMPLOYEES F
                WHERE E.dept_num = F.dept_num AND
                F.salary <= 1800)
WHERE E.salary < 1200;
```

```
SELECT num, salary
FROM EMPLOYEES;
```

num	salary
1000	1040
7369	1040
7499	1500
7521	1625
7566	2900
7654	1600
7698	3005
7782	2885
7788	3000
7844	1350
7876	1430
7900	1335
7902	3000
7934	1690
8000	2885
8001	2885

This kind of UPDATE/DELETE only works from MariaDB 10.3.1 and newer versions...

Let’s work!

Do the exercises:

- ApplyingToCollege

In this exercise you will use/see:

- Everything in this UNIT!

Sources.

- **M. J. Ramos, A. Ramos and F. Montero.** Sistemas gestores de bases de datos (Chapter 1, pag. 7-15). McGrawHill: 1th Edition, 2006.
- **Abraham Silberschatz, Henry F. Korth and S. Sudarshan**. *Database System Concepts (Chapter 3)*. McGrawHill: 6th Edition, 2010.<http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html>
- Apunts de la UIB del professor Miquel Manresa (1996).
- https://www.studytonight.com/dbms/