Unit 4: Intermediate SQL.

2020/2021

Contents

4.1. 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 condition). It also specifies the attributes that are present in the result of the join
- 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**.

4.1. Joined Relations (II).

```
create database UNIT41;
use UNIT41;
CREATE TABLE `DEPARTMENTS` (
  `num` tinyint NOT NULL,
  `name` varchar(30) NOT NULL,
  primary key (`num`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
INSERT INTO `DEPARTMENTS` (`num`, `name`) VALUES
(10, 'ACCOUNTING'),
(20, 'RESEARCH'),
(30, 'SALES'),
(40, 'PRODUCTION');
CREATE TABLE `EMPLOYEES` (
  `num` int(11) NOT NULL,
  `surname` varchar(50) NOT NULL,
  `name` varchar(50) NOT NULL,
  `manager` int(11) DEFAULT NULL,
  `start_date` date DEFAULT NULL,
  `salary` int(11) DEFAULT NULL,
  `commission` int(11) DEFAULT NULL,
  `dept_num` tinyint DEFAULT NULL,
  primary key (`num`),
  foreign key (`dept_num`) references `DEPARTMENTS` (`num`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

num	name
10	ACCOUNTING
20	RESEARCH
30	SALES
40	PRODUCTION

There aren't employees in that department...

num	surname	name	manager	start_date	salary	commission	dept_num
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20

```
INSERT INTO `EMPLOYEES` (`num`, `surname`, `name`, `manager`,
`start_date`, `salary`, `commission`, `dept_num`) VALUES
(1000, 'PITT', 'BRAD', NULL, '1984-01-01', 1040, NULL, 20),
(7369, 'SÁNCHEZ', 'SERGIO', 8001, '1990-12-17', 1040, NULL, 20),
(7499, 'ARROYO', 'MARTA', 7698, '1990-02-20', 1500, 390, 30),
(7521, 'SEGUI', 'RAUL', 7782, '1991-02-22', 1625, 650, 30),
(7566, 'GUAL', 'JUDIT', 1000, '1991-04-02', 2900, NULL, 20),
(7654, 'MARTÍN', 'MONICA', 7698, '1991-09-29', 1600, 1020, NULL),
(7698, 'NEGRO', 'BARTOLOME', 1000, '1991-05-01', 3005, NULL, 30),
(7782, 'CRESPI', 'ENRIQUE', 1000, '1991-06-09', 2885, NULL, 10),
(7788, 'GIL', 'JAIME', 8000, '1991-11-09', 3000, NULL, 20),
(7844, 'TOVAR', 'LUIS', 7698, '1991-09-08', 1350, 0, 30),
(7876, 'ALONSO', 'FERNANDO', 7788, '1991-09-23', 1430, NULL, 20),
(7900, 'JUAREZ', 'XAVIER', 8001, '1991-12-03', 1335, NULL, 30),
(7902, 'FERNÁNDEZ', 'ANA', 8000, '1991-12-03', 3000, NULL, 20),
(7934, 'MUÑOZ', 'ANTONIA', 8001, '1992-01-23', 1690, NULL, 10),
(8000, 'BANDERAS', 'ANTONIO', 1000, '1991-01-09', 2885, NULL,
20),
(8001, 'RUIZ', 'FERNANDA', 1000, '1992-06-10', 2885, NULL, 20);
```

4.1. Joined Relations (III).

- Relation *COURSES:*

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-135	Robotics	Comp. Sci.	3
CS-190	Game Design	Comp. Sci.	4

- Relation *PREREQUISITES:*

course_id	prereq_id
BIO-201	BIO-101
CS-190	CS-101
CS-347	CS-101

Can you do anything better in that database?

```
create database UNIT42;
use UNIT42;
create table COURSES (
        course_id varchar(7),
        title varchar(30),
        dept_name varchar(20),
        credits tinyint,
        primary key (course_id)) engine=InnoDB;
insert into COURSES values ('BIO-301', 'Genetics',
'Biology', 4);
insert into COURSES values ('CS-190', 'Game Design',
'Comp. Sci.', 4);
insert into COURSES values ('CS-135', 'Robotics',
'Comp. Sci.', 3);
create table PREREQUISITES (
        course_id varchar(7),
        prereq_id varchar(7)) engine=InnoDB;
insert into PREREQUISITES values ('BIO-201',
'BIO-101');
insert into PREREQUISITES values ('CS-190', 'CS-101');
insert into PREREQUISITES values ('CS-347', 'CS-101');
```

- Observe that:
  - PREREQUISITES information is missing for CS-315 and
  - COURSES information is missing for CS-347

4.1. Joined Relations (IV).

```
create database UNIT43;
use UNIT43;
create table PEOPLE (
      id_card varchar(9),
      name varchar(30) NOT NULL,
      surname1 varchar(30) NOT NULL,
      surname2 varchar(30) NOT NULL,
      father varchar(9),
      mother varchar(9),
      primary key (id_card),
      foreign key (father) references PEOPLE (id_card),
      foreign key (mother) references PEOPLE (id_card)) engine=InnoDB;
```

Babies must have an id card...

id_card	name	surname1	surname2	father	mother
11111111A	Antonio	Gual	Mateu	NULL	NULL
11113333F	Manuel	Gomis	Munar	NULL	NULL
11132222G	Antonia	Mut	Xamena	NULL	NULL
22222222B	Francisca	Mir	Amer	NULL	NULL
33333333C	Antonio	Gual	Mir	11111111A	22222222B
33333334C	Pixedis	Gual	Mir	11111111A	22222222B
33333335C	Francisca	Gual	Mir	11111111A	22222222B
44444444C	Antonio	Gomis	Mut	11113333F	11132222G
44444445D	Francisco	Gomis	Mut	11113333F	11132222G
44444446D	Josep	Gomis	Mut	11113333F	11132222G
55555555X	Francisca	Gual	Mir	NULL	33333335C

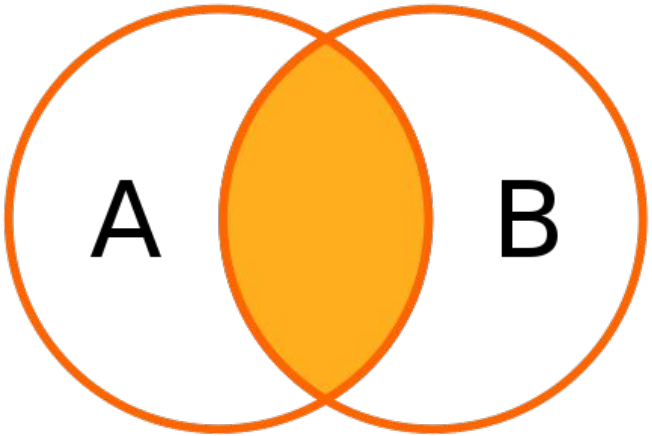
```
insert into PEOPLE values ('11111111A', 'Antonio', 'Gual', 'Mateu', NULL, NULL);
insert into PEOPLE values ('22222222B', 'Francisca', 'Mir', 'Amer', NULL, NULL);
insert into PEOPLE values ('33333333C', 'Antonio', 'Gual', 'Mir', '11111111A', '22222222B');
insert into PEOPLE values ('33333334C', 'Pixedis', 'Gual', 'Mir', '11111111A', '22222222B');
insert into PEOPLE values ('33333335C', 'Francisca', 'Gual', 'Mir', '11111111A', '22222222B');
insert into PEOPLE values ('55555555X', 'Francisca', 'Gual', 'Mir', NULL, '33333335C');
insert into PEOPLE values ('11113333F', 'Manuel', 'Gomis', 'Munar', NULL, NULL);
insert into PEOPLE values ('11132222G', 'Antonia', 'Mut', 'Xamena', NULL, NULL);
insert into PEOPLE values ('44444444C', 'Antonio', 'Gomis', 'Mut', '11113333F', '11132222G');
insert into PEOPLE values ('44444445D', 'Francisco', 'Gomis', 'Mut', '11113333F', '11132222G');
insert into PEOPLE values ('44444446D', 'Josep', 'Gomis', 'Mut', '11113333F', '11132222G');
```

4.2. Inner join (I).

- An **inner join** requires each row in the two joined tables to have matching column values.
- Inner join creates a new result table by combining column values of two tables (A and B) based upon the **join-predicate**.
- The query compares each row of A with each row of B to find all pairs of rows which satisfy the **join-predicate**.
- All the JOINS that we did in Unit 3 are INNER JOINS.

4.2. Inner join (II).

- When the **join-predicate** is satisfied by matching **non-NULL values**, column values for each matched pair of rows of A and B are combined into a result row.
- The result of the join can be defined as the outcome of first taking the Cartesian product (or Cross join) of all rows in the tables (combining every row in table A with every row in table B) and then returning all rows which satisfy the join predicate.



4.2. Inner join (III).

```
MariaDB [UNIT41]> SELECT * FROM EMPLOYEES E, DEPARTMENTS D;
```

num	surname	name	manager	start_date	salary	commission	dept_num	num	name
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	10	ACCOUNTING
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	20	RESEARCH
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	30	SALES
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	40	PRODUCTION
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	10	ACCOUNTING
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	20	RESEARCH
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	30	SALES
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	40	PRODUCTION
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	10	ACCOUNTING
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	20	RESEARCH
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	30	SALES
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	40	PRODUCTION
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	10	ACCOUNTING
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	20	RESEARCH
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	30	SALES
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	40	PRODUCTION
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	10	ACCOUNTING
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	20	RESEARCH
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	30	SALES
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	40	PRODUCTION
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	10	ACCOUNTING
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	20	RESEARCH
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	30	SALES
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	40	PRODUCTION
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	10	ACCOUNTING
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	20	RESEARCH
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	30	SALES
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	40	PRODUCTION
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	10	ACCOUNTING
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	20	RESEARCH
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	30	SALES
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	40	PRODUCTION
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	10	ACCOUNTING
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	20	RESEARCH
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	30	SALES
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	40	PRODUCTION
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	10	ACCOUNTING
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	20	RESEARCH
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	30	SALES
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	40	PRODUCTION
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	10	ACCOUNTING
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	20	RESEARCH
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	30	SALES
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	40	PRODUCTION
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	10	ACCOUNTING
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	20	RESEARCH
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	30	SALES
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	40	PRODUCTION
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	10	ACCOUNTING
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	20	RESEARCH
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	30	SALES
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	40	PRODUCTION
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	10	ACCOUNTING
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	20	RESEARCH
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	30	SALES
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	40	PRODUCTION
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	10	ACCOUNTING
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	20	RESEARCH
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	30	SALES
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	40	PRODUCTION
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	10	ACCOUNTING
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	20	RESEARCH
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	30	SALES
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	40	PRODUCTION

Red color: The query compares each row of A with each row of B to find all pairs of **rows which satisfy the join-predicate**.

Blue color: **NULLs never satisfy the join-predicate**. Mónica Martín is not here because it is not satisfied E.dept_num=D.num

If I want all the Employees (satisfied join-predicate and unsatisfied join-predicate) SOLUTION: OUTER JOIN.

```
MariaDB [UNIT41]> SELECT E.surname, E.dept_num, D.name
    -> FROM EMPLOYEES AS E LEFT OUTER JOIN DEPARTMENTS AS D
    -> ON E.dept_num=D.num;
```

surname	dept_num	name
PITT	20	RESEARCH
SÁNCHEZ	20	RESEARCH
ARROYO	30	SALES
SEGUI	30	SALES
GUAL	20	RESEARCH
MARTÍN	NULL	NULL
NEGRO	30	SALES
CRESPI	10	ACCOUNTING
GIL	20	RESEARCH
TOVAR	30	SALES
ALONSO	20	RESEARCH
JUAREZ	30	SALES
FERNÁNDEZ	20	RESEARCH
MUÑOZ	10	ACCOUNTING
BANDERAS	20	RESEARCH
RUIZ	20	RESEARCH

16 rows in set (0.001 sec)

```
MariaDB [UNIT41]> SELECT * FROM EMPLOYEES E, DEPARTMENTS D
    -> WHERE E.dept_num=D.num;
```

num	surname	name	manager	start_date	salary	commission	dept_num	num	name
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	10	ACCOUNTING
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	10	ACCOUNTING
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	20	RESEARCH
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	20	RESEARCH
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	20	RESEARCH
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	20	RESEARCH
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	20	RESEARCH
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	20	RESEARCH
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	20	RESEARCH
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	20	RESEARCH
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	30	SALES
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	30	SALES
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	30	SALES
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	30	SALES
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	30	SALES

15 rows in set (0.001 sec)

4.2. Inner join (IV).

- **Explicit inner join**:
  SELECT EMPLOYEES.surname, EMPLOYEES.dept_num,
  DEPARTMENTS.name
  FROM EMPLOYEES
  INNER JOIN DEPARTMENTS ON
  EMPLOYEES.dept_num = DEPARTMENTS.num;
- **Implicit inner join**:
  SELECT EMPLOYEES.surname, EMPLOYEES.dept_num,
  DEPARTMENTS.name
  FROM EMPLOYEES, DEPARTMENTS
  WHERE
  EMPLOYEES.dept_num = DEPARTMENTS.num;
- One can further classify inner joins as **equi-joins**, as **natural joins**, or as **cross-joins**.

It is just the same!!

4.3. Equi-join.

- An **equi-join** is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join.

- Explicit equi-join:

```
SELECT *
FROM EMPLOYEES JOIN DEPARTMENTS
ON
EMPLOYEES.dept_num = DEPARTMENTS.num;
```

- Implicit equi-join:

```
SELECT *
FROM EMPLOYEES, DEPARTMENTS
WHERE
EMPLOYEES.dept_num = DEPARTMENTS.num;
```

4.4. Natural join (I).

- A **natural join** is a type of equi-join where **the join predicate have the same column-names** in the joined tables.
- The resulting joined table contains only one column for each pair of equally named columns.
- In the case that no columns with the same names are found, the result is a cross join.

4.4. Natural join (II).

- For instance:
  - SELECT * FROM EMPLOYEES NATURAL JOIN DEPARTMENTS;
  - Returns 0 rows (=Empty set):

```
MariaDB [UNIT41]> SELECT * FROM EMPLOYEES NATURAL JOIN DEPARTMENTS;
Empty set (0.001 sec)
```

4.4. Natural join (III).

- For instance:
    - SELECT * FROM COURSES
      NATURAL JOIN PREREQUISITES;
    - Returns 1 row:

```
+-----------+-------------+------------+---------+-----------+
| course_id | title       | dept_name  | credits | prereq_id |
+-----------+-------------+------------+---------+-----------+
| CS-190    | Game Design | Comp. Sci. |       4 | CS-101    |
+-----------+-------------+------------+---------+-----------+
1 row in set (0.000 sec)
```

4.4. Natural join (IV).

- PostgreSQL, MySQL and Oracle support natural joins; Microsoft T-SQL and IBM DB2 do not.
- A natural join assumes **stability and consistency in column names** which can change during vendor mandated version upgrades.

4.5. Cross join.

- **CROSS JOIN** returns the **Cartesian product** of rows from tables in the join. In other words, it will produce rows which combine each row from the first table with each row from the second table.
- Explicit cross join:

  SELECT *
  FROM EMPLOYEES CROSS JOIN DEPARTMENTS;
- Implicit cross join:

  SELECT *
  FROM EMPLOYEES, DEPARTMENTS;
- The cross join does not itself apply any predicate to filter rows from the joined table. The results of a cross join can be filtered by using a **WHERE** clause which may then produce the equivalent of an inner join.
- Normal uses are for **checking the server's performance**.

```
MariaDB [UNIT41]> SELECT * FROM EMPLOYEES E, DEPARTMENTS D;
```

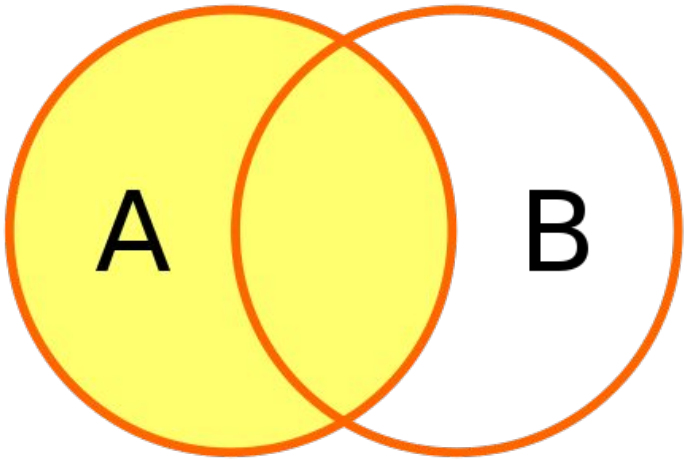
num	surname	name	manager	start_date	salary	commission	dept_num	num	name
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	10	ACCOUNTING
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	20	RESEARCH
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	30	SALES
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	40	PRODUCTION
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	10	ACCOUNTING
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	20	RESEARCH
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	30	SALES
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	40	PRODUCTION
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	10	ACCOUNTING
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	20	RESEARCH
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	30	SALES
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	40	PRODUCTION
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	10	ACCOUNTING
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	20	RESEARCH
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	30	SALES
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	40	PRODUCTION
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	10	ACCOUNTING
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	20	RESEARCH
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	30	SALES
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	40	PRODUCTION
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	10	ACCOUNTING
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	20	RESEARCH
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	30	SALES
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	40	PRODUCTION
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	10	ACCOUNTING
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	20	RESEARCH
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	30	SALES
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	40	PRODUCTION
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	10	ACCOUNTING
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	20	RESEARCH
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	30	SALES
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	40	PRODUCTION
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	10	ACCOUNTING
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	20	RESEARCH
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	30	SALES
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	40	PRODUCTION
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	10	ACCOUNTING
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	20	RESEARCH
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	30	SALES
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	40	PRODUCTION
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	10	ACCOUNTING
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	20	RESEARCH
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	30	SALES
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	40	PRODUCTION
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	10	ACCOUNTING
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	20	RESEARCH
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	30	SALES
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	40	PRODUCTION
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	10	ACCOUNTING
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	20	RESEARCH
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	30	SALES
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	40	PRODUCTION
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	10	ACCOUNTING
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	20	RESEARCH
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	30	SALES
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	40	PRODUCTION
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	10	ACCOUNTING
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	20	RESEARCH
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	30	SALES
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	40	PRODUCTION
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	10	ACCOUNTING
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	20	RESEARCH
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	30	SALES
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	40	PRODUCTION

4.6. Outer join.

- **Outer join** is an extension of the join operation that avoids **loss of information**.
- The joined table retains each row, even if no other matching row exists.
- Uses **null values**.
- Outer joins subdivide further into **left outer joins**, **right outer joins**, and **full outer joins**, depending on which table's rows are retained (left, right, or both).
- ‘left’ and ‘right’ refer to the two sides of the JOIN keyword.

4.4. Left outer join (l).

- The result of a **left outer join** (or simply **left join**) for tables A and B always contains all rows of the "left" table (A), even if the join-condition does not find any matching row in the "right" table (B).
- A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table, including rows with **NULL** values in the link column.



4.4. Left outer join (II).

```
SELECT *
FROM EMPLOYEES INNER JOIN DEPARTMENTS
ON
EMPLOYEES.dept_num = DEPARTMENTS.num
ORDER BY EMPLOYEES.NUM;
```

num	surname	name	manager	start_date	salary	commission	dept_num	num	name
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	20	RESEARCH
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	20	RESEARCH
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	30	SALES
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	30	SALES
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	20	RESEARCH
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	30	SALES
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	10	ACCOUNTING
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	20	RESEARCH
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	30	SALES
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	20	RESEARCH
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	30	SALES
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	20	RESEARCH
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	10	ACCOUNTING
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	20	RESEARCH
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	20	RESEARCH

15 rows in set (0.056 sec)

```
SELECT *
FROM EMPLOYEES LEFT OUTER JOIN
DEPARTMENTS
ON
EMPLOYEES.dept_num = DEPARTMENTS.num
ORDER BY EMPLOYEES.NUM;
```

num	surname	name	manager	start_date	salary	commission	dept_num	num	name
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	20	RESEARCH
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	20	RESEARCH
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	30	SALES
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	30	SALES
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	20	RESEARCH
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	NULL	NULL
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	30	SALES
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	10	ACCOUNTING
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	20	RESEARCH
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	30	SALES
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	20	RESEARCH
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	30	SALES
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	20	RESEARCH
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	10	ACCOUNTING
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	20	RESEARCH
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	20	RESEARCH

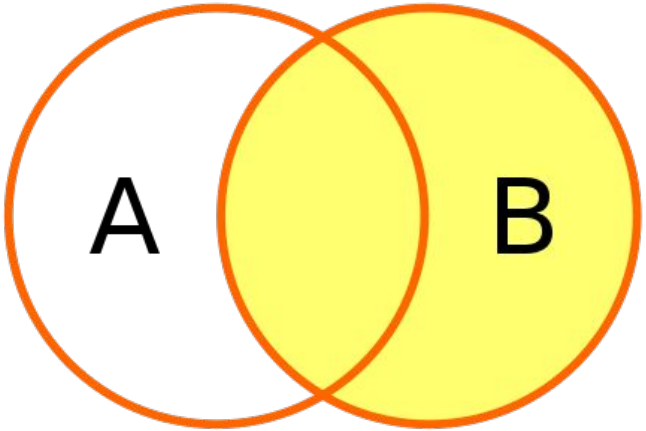
16 rows in set (0.008 sec)

4.4. Left outer join (III).

- **Alternative syntaxes**:
  - **Oracle** supports the deprecated syntax:
    ```
    SELECT *
    FROM EMPLOYEES, DEPARTMENTS
    WHERE
    EMPLOYEES.dept_num = DEPARTMENTS.num(+)
    ORDER BY EMPLOYEES.NUM;
    ```
  - **Sybase** supports the syntax (**Microsoft SQL Server** deprecated this syntax since version 2000):
    ```
    SELECT *
    FROM EMPLOYEES, DEPARTMENTS
    WHERE
    EMPLOYEES.dept_num *= DEPARTMENTS.num
    ORDER BY EMPLOYEES.NUM;
    ```
  - **IBM Informix** supports the syntax:
    ```
    SELECT *
    FROM EMPLOYEES, OUTER DEPARTMENTS
    WHERE
    EMPLOYEES.dept_num = DEPARTMENTS.num(+)
    ORDER BY EMPLOYEES.NUM;
    ```

4.8. Right outer join (I).

- A **right outer join** (or **right join**) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those rows that have no match in B.
- A right outer join returns all the values from the right table and matched values from the left table (NULL in the case of no matching join predicate). For example, this allows us to find each employee and his or her department, but still show departments that have no employees.



4.8. Right outer join (II).

```
SELECT *
FROM EMPLOYEES RIGHT OUTER JOIN DEPARTMENTS
ON
EMPLOYEES.dept_num = DEPARTMENTS.num
ORDER BY EMPLOYEES.NUM;
```

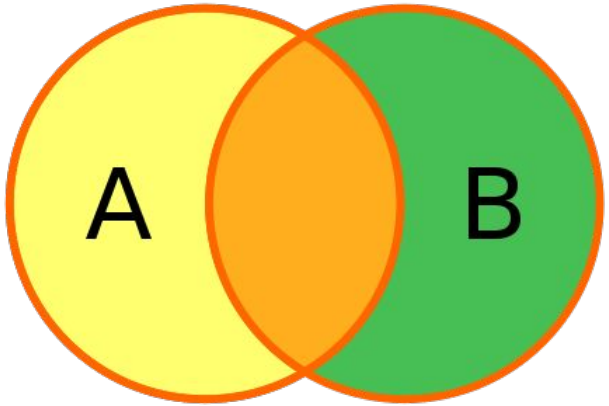
num	surname	name	manager	start_date	salary	commission	dept_num	num	name
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	40	PRODUCTION
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	20	RESEARCH
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	20	RESEARCH
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	30	SALES
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	30	SALES
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	20	RESEARCH
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	30	SALES
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	10	ACCOUNTING
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	20	RESEARCH
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	30	SALES
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	20	RESEARCH
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	30	SALES
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	20	RESEARCH
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	10	ACCOUNTING
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	20	RESEARCH
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	20	RESEARCH

16 rows in set (0.011 sec)

Right and left outer joins are functionally equivalent. Neither provides any functionality that the other does not, so right and left outer joins may replace each other as long as the table order is switched.

4.9. Full outer join (I).

- Conceptually, a **full outer join** combines the effect of applying both left and right outer joins. Where rows in the FULL OUTER JOINed tables do not match, the result set will have NULL values for every column of the table that lacks a matching row.
- For those rows that do match, a single row will be produced in the result set (containing columns populated from both tables).



4.9. Full outer join (II).

You don't have FULL JOINS in MySQL/MariaDB! Check this link.

```
SELECT *
FROM EMPLOYEES FULL OUTER JOIN DEPARTMENTS
ON
EMPLOYEES.dept_num = DEPARTMENTS.num
ORDER BY EMPLOYEES.num;
```

num	surname	name	manager	start_date	salary	commission	dept_num	num	name
1000	PITT	BRAD	NULL	1984-01-01	1040	NULL	20	20	RESEARCH
7369	SÁNCHEZ	SERGIO	8001	1990-12-17	1040	NULL	20	20	RESEARCH
7499	ARROYO	MARTA	7698	1990-02-20	1500	390	30	30	SALES
7521	SEGUI	RAUL	7782	1991-02-22	1625	650	30	30	SALES
7566	GUAL	JUDIT	1000	1991-04-02	2900	NULL	20	20	RESEARCH
7654	MARTÍN	MONICA	7698	1991-09-29	1600	1020	NULL	NULL	NULL
7698	NEGRO	BARTOLOME	1000	1991-05-01	3005	NULL	30	30	SALES
7782	CRESPI	ENRIQUE	1000	1991-06-09	2885	NULL	10	10	ACCOUNTING
7788	GIL	JAIME	8000	1991-11-09	3000	NULL	20	20	RESEARCH
7844	TOVAR	LUIS	7698	1991-09-08	1350	0	30	30	SALES
7876	ALONSO	FERNANDO	7788	1991-09-23	1430	NULL	20	20	RESEARCH
7900	JUAREZ	XAVIER	8001	1991-12-03	1335	NULL	30	30	SALES
7902	FERNÁNDEZ	ANA	8000	1991-12-03	3000	NULL	20	20	RESEARCH
7934	MUÑOZ	ANTONIA	8001	1992-01-23	1690	NULL	10	10	ACCOUNTING
8000	BANDERAS	ANTONIO	1000	1991-01-09	2885	NULL	20	20	RESEARCH
8001	RUIZ	FERNANDA	1000	1992-06-10	2885	NULL	20	20	RESEARCH
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	40	PRODUCTION

17 rows in set (0.009 sec)

This allows us to see each employee who is in a department and each department that has an employee, but also see each employee who is not part of a department and each department which doesn't have an employee.

4.9. Full outer join (II).

- Way to do it in MySQL/MariaDB:

```
(SELECT *
FROM EMPLOYEES LEFT OUTER JOIN DEPARTMENTS
ON
EMPLOYEES.dept_num = DEPARTMENTS.num)
UNION
(SELECT *
FROM EMPLOYEES RIGHT OUTER JOIN DEPARTMENTS
ON
EMPLOYEES.dept_num = DEPARTMENTS.num);
```

```
CREATE TABLE `EMPLOYEES` (
  `num` int(11) NOT NULL,
  `surname` varchar(50) NOT NULL,
  `name` varchar(50) NOT NULL,
  `manager` int(11) DEFAULT NULL,
  `start_date` date DEFAULT NULL,
  `salary` int(11) DEFAULT NULL,
  `commission` int(11) DEFAULT NULL,
  `dept_num` tinyint DEFAULT NULL,
  primary key (`num`),
  foreign key (`dept_num`) references `DEPARTMENTS` (`num`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `DEPARTMENTS` (
  `num` tinyint NOT NULL,
  `name` varchar(30) NOT NULL,
  primary key (`num`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

ALWAYS OUTER JOIN

```
SELECT *
FROM EMPLOYEES LEFT OUTER JOIN DEPARTMENTS
ON EMPLOYEES.dept_num = DEPARTMENTS.num
ORDER BY EMPLOYEES.NUM;
```

ALLOWS NULLs

4.10. Condition in WHERE clause vs. ON clause.

```
select concat(E.name, ' ', E.surname) as fullname, E.salary, O.name
from EMPLOYEES E
LEFT OUTER JOIN OCCUPATIONS O ON E.occu_code = O.code
ORDER BY fullname;
```

fullname	salary	name
ANA FERNÁNDEZ	3000	ANALYST
ANTONIA MUÑOZ	1690	EMPLOYEE
ANTONIO BANDERAS	2885	MANAGER
BARTOLOME AMER	3005	MANAGER
ENRIQUE COLOM	2885	MANAGER
FERNANDA RUIZ	2885	MANAGER
FERNANDO ALONSO	1430	EMPLOYEE
JAVIER GIL	3000	ANALYST
JOSEP AGUILO	1625	SALESMAN
JUDIT AROCA	2900	MANAGER
LUIS TOVAR	1350	SALESMAN
MARTA ARROYO	1500	SALESMAN
MONICA MARTÍN	1600	SALESMAN
SERGIO SÁNCHEZ	1040	EMPLOYEE
XAVIER JIMENO	1335	EMPLOYEE

15 rows in set (0.001 sec)

LEFT OUTER JOIN without conditions

```
select concat(E.name, ' ', E.surname) as fullname, E.salary, O.name
from EMPLOYEES E
LEFT OUTER JOIN OCCUPATIONS O ON E.occu_code = O.code
AND
E.salary > 2000
ORDER BY fullname;
```

fullname	salary	name
ANA FERNÁNDEZ	3000	ANALYST
ANTONIA MUÑOZ	1690	NULL
ANTONIO BANDERAS	2885	MANAGER
BARTOLOME AMER	3005	MANAGER
ENRIQUE COLOM	2885	MANAGER
FERNANDA RUIZ	2885	MANAGER
FERNANDO ALONSO	1430	NULL
JAVIER GIL	3000	ANALYST
JOSEP AGUILO	1625	NULL
JUDIT AROCA	2900	MANAGER
LUIS TOVAR	1350	NULL
MARTA ARROYO	1500	NULL
MONICA MARTÍN	1600	NULL
SERGIO SÁNCHEZ	1040	NULL
XAVIER JIMENO	1335	NULL

15 rows in set (0.001 sec)

LEFT OUTER JOIN only for EMPLOYEES with salary > 2000

```
select concat(E.name, ' ', E.surname) as fullname, E.salary, O.name
from EMPLOYEES E
LEFT OUTER JOIN OCCUPATIONS O ON E.occu_code = O.code
WHERE
E.salary > 2000
ORDER BY fullname;
```

fullname	salary	name
ANA FERNÁNDEZ	3000	ANALYST
ANTONIO BANDERAS	2885	MANAGER
BARTOLOME AMER	3005	MANAGER
ENRIQUE COLOM	2885	MANAGER
FERNANDA RUIZ	2885	MANAGER
JAVIER GIL	3000	ANALYST
JUDIT AROCA	2900	MANAGER

7 rows in set (0.001 sec)

LEFT OUTER JOIN and after making the join condition salary > 2000

You can download this new database EMPLOYEESDBNORMAL from here.

4.10. Condition in WHERE clause vs. ON clause.

- With INNER JOIN there aren't differences:

```
select concat(E.name, ' ', E.surname) as fullname,
E.salary, O.name
from EMPLOYEES E
INNER JOIN OCCUPATIONS O ON E.occu_code =
O.code
AND
E.salary > 2000
ORDER BY fullname;
```

```
+------------------+--------+---------+
| fullname         | salary | name    |
+------------------+--------+---------+
| ANA FERNÁNDEZ    |   3000 | ANALYST |
| ANTONIO BANDERAS |   2885 | MANAGER |
| BARTOLOME AMER   |   3005 | MANAGER |
| ENRIQUE COLOM    |   2885 | MANAGER |
| FERNANDA RUIZ    |   2885 | MANAGER |
| JAVIER GIL       |   3000 | ANALYST |
| JUDIT AROCA      |   2900 | MANAGER |
+------------------+--------+---------+
7 rows in set (0.004 sec)
```

```
select concat(E.name, ' ', E.surname) as fullname,
E.salary, O.name
from EMPLOYEES E
INNER JOIN OCCUPATIONS O ON E.occu_code =
O.code
WHERE
E.salary > 2000
ORDER BY fullname;
```

```
+------------------+--------+---------+
| fullname         | salary | name    |
+------------------+--------+---------+
| ANA FERNÁNDEZ    |   3000 | ANALYST |
| ANTONIO BANDERAS |   2885 | MANAGER |
| BARTOLOME AMER   |   3005 | MANAGER |
| ENRIQUE COLOM    |   2885 | MANAGER |
| FERNANDA RUIZ    |   2885 | MANAGER |
| JAVIER GIL       |   3000 | ANALYST |
| JUDIT AROCA      |   2900 | MANAGER |
+------------------+--------+---------+
7 rows in set (0.001 sec)
```

4.10. Condition in WHERE clause vs. ON clause.

- Using MySQL/MariaDB LEFT JOIN clause to find unmatched rows:

```
select concat(E.name, ' ', E.surname) as fullname, E.salary, D.name
from EMPLOYEES E
LEFT OUTER JOIN DEPARTMENTS D ON E.dept_num = D.num
ORDER BY fullname;
```

```
+------------------+--------+------------+
| fullname         | salary | name       |
+------------------+--------+------------+
| ANA FERNÁNDEZ    |   3000 | RESEARCH   |
| ANTONIA MUÑOZ    |   1690 | ACCOUNTING |
| ANTONIO BANDERAS |   2885 | RESEARCH   |
| BARTOLOME NEGRO  |   3005 | SALES      |
| BRAD PITT        |   1040 | RESEARCH   |
| ENRIQUE CRESPI   |   2885 | ACCOUNTING |
| FERNANDA RUIZ    |   2885 | RESEARCH   |
| FERNANDO ALONSO  |   1430 | RESEARCH   |
| JAIME GIL        |   3000 | RESEARCH   |
| JUDIT GUAL       |   2900 | RESEARCH   |
| LUIS TOVAR       |   1350 | SALES      |
| MARTA ARROYO     |   1500 | SALES      |
| MONICA MARTÍN    |   1600 | NULL       |
| RAUL SEGUI       |   1625 | SALES      |
| SERGIO SÁNCHEZ   |   1040 | RESEARCH   |
| XAVIER JUAREZ    |   1335 | SALES      |
+------------------+--------+------------+
16 rows in set (0.001 sec)
```

```
select concat(E.name, ' ', E.surname) as fullname, E.salary,
      E.dept_num as name
from EMPLOYEES E
WHERE
E.dept_num is NULL
ORDER BY fullname;
```

```
+----------------+--------+------+
| fullname       | salary | name |
+----------------+--------+------+
| MONICA MARTÍN  |   1600 | NULL |
+----------------+--------+------+
1 row in set (0.001 sec)
```

```
select concat(E.name, ' ', E.surname) as fullname, E.salary, D.name
from EMPLOYEES E, DEPARTMENTS D
WHERE
E.dept_num = D.num
ORDER BY fullname;
```

USE UNIT41;
(this is not EMPLOYEESDBNORMAL)

```
+------------------+--------+------------+
| fullname         | salary | name       |
+------------------+--------+------------+
| ANA FERNÁNDEZ    |   3000 | RESEARCH   |
| ANTONIA MUÑOZ    |   1690 | ACCOUNTING |
| ANTONIO BANDERAS |   2885 | RESEARCH   |
| BARTOLOME NEGRO  |   3005 | SALES      |
| BRAD PITT        |   1040 | RESEARCH   |
| ENRIQUE CRESPI   |   2885 | ACCOUNTING |
| FERNANDA RUIZ    |   2885 | RESEARCH   |
| FERNANDO ALONSO  |   1430 | RESEARCH   |
| JAIME GIL        |   3000 | RESEARCH   |
| JUDIT GUAL       |   2900 | RESEARCH   |
| LUIS TOVAR       |   1350 | SALES      |
| MARTA ARROYO     |   1500 | SALES      |
| RAUL SEGUI       |   1625 | SALES      |
| SERGIO SÁNCHEZ   |   1040 | RESEARCH   |
| XAVIER JUAREZ    |   1335 | SALES      |
+------------------+--------+------------+
15 rows in set (0.007 sec)
```

4.10. Condition in WHERE clause vs. ON clause.

- When the primary key and the foreign key have the same name you can use:
  - USING (CustomerID), instead of
  - ON C.CustomerID = O.CustomerID
  - Example:

```
SELECT
C.CustomerID,
C.CompanyName,
O.OrderID,
O.OrderDate
FROM
Customers AS C LEFT JOIN Orders AS O
USING (CustomerID);
```

```
SELECT
C.CustomerID,
C.CompanyName,
O.OrderID,
O.OrderDate
FROM
Customers AS C LEFT JOIN Orders AS O
ON C.CustomerID=O.CustomerID;
```

4.10. Condition in WHERE clause vs. ON clause.

- Equivalent syntaxes:
  - A LEFT JOIN B ⟷ A LEFT OUTER JOIN B
  - A RIGHT JOIN B ⟷ A RIGHT OUTER JOIN B
  - A FULL JOIN B ⟷ A FULL OUTER JOIN B
  - A JOIN B ⟷ A INNER JOIN B

4.11. Self-join (I).

- A self-join is joining a table to itself.
- For instance, show me the fathers (table PEOPLE) using explicit INNER JOIN:
  SELECT Person.name, Person.surname1, Person.surname2, Father.name, Father.surname1, Father.surname2
  FROM PEOPLE as Person INNER JOIN PEOPLE as Father ON Person.father = Father.id_card
  ORDER BY Person.surname1, Person.surname2, Person.name;

```
+-----------+----------+----------+---------+----------+----------+
| name      | surname1 | surname2 | name    | surname1 | surname2 |
+-----------+----------+----------+---------+----------+----------+
| Antonio   | Gomis    | Mut      | Manuel  | Gomis    | Munar    |
| Francisco | Gomis    | Mut      | Manuel  | Gomis    | Munar    |
| Josep     | Gomis    | Mut      | Manuel  | Gomis    | Munar    |
| Antonio   | Gual     | Mir      | Antonio | Gual     | Mateu    |
| Francisca | Gual     | Mir      | Antonio | Gual     | Mateu    |
| Pixedis   | Gual     | Mir      | Antonio | Gual     | Mateu    |
+-----------+----------+----------+---------+----------+----------+
6 rows in set (0.043 sec)
```

4.11. Self-join (II).

- Using the implicit inner join is easier to understand how to show both parents:

```
SELECT Person.name, Person.surname1, Person.surname2,
Father.name, Father.surname1, Father.surname2,
Mother.name, Mother.surname1, Mother.surname2
FROM PEOPLE as Person, PEOPLE as Father, PEOPLE as Mother
where Person.father = Father.id_card and
Person.mother = Mother.id_card
ORDER BY Person.surname1, Person.surname2, Person.name;
```

```
+-----------+----------+----------+---------+----------+----------+-----------+----------+----------+
| name      | surname1 | surname2 | name    | surname1 | surname2 | name      | surname1 | surname2 |
+-----------+----------+----------+---------+----------+----------+-----------+----------+----------+
| Antonio   | Gomis    | Mut      | Manuel  | Gomis    | Munar    | Antonia   | Mut      | Xamena   |
| Francisco | Gomis    | Mut      | Manuel  | Gomis    | Munar    | Antonia   | Mut      | Xamena   |
| Josep     | Gomis    | Mut      | Manuel  | Gomis    | Munar    | Antonia   | Mut      | Xamena   |
| Antonio   | Gual     | Mir      | Antonio | Gual     | Mateu    | Francisca | Mir      | Amer     |
| Francisca | Gual     | Mir      | Antonio | Gual     | Mateu    | Francisca | Mir      | Amer     |
| Pixedis   | Gual     | Mir      | Antonio | Gual     | Mateu    | Francisca | Mir      | Amer     |
+-----------+----------+----------+---------+----------+----------+-----------+----------+----------+
6 rows in set (0.018 sec)
```

4.11. Self-join (II).

- Let's consider NULLs now:

```
SELECT Person.name, Person.surname1, Person.surname2,
       Father.name, Father.surname1, Father.surname2,
       Mother.name, Mother.surname1, Mother.surname2
FROM PEOPLE as Person
       LEFT OUTER JOIN PEOPLE as Father ON Person.father = Father.id_card
       LEFT OUTER JOIN PEOPLE as Mother ON Person.mother = Mother.id_card
ORDER BY Person.surname1, Person.surname2, Person.name;
```

Two OUTER JOINs together...

name	surname1	surname2	name	surname1	surname2	name	surname1	surname2
Manuel	Gomis	Munar	NULL	NULL	NULL	NULL	NULL	NULL
Antonio	Gomis	Mut	Manuel	Gomis	Munar	Antonia	Mut	Xamena
Francisco	Gomis	Mut	Manuel	Gomis	Munar	Antonia	Mut	Xamena
Josep	Gomis	Mut	Manuel	Gomis	Munar	Antonia	Mut	Xamena
Antonio	Gual	Mateu	NULL	NULL	NULL	NULL	NULL	NULL
Antonio	Gual	Mir	Antonio	Gual	Mateu	Francisca	Mir	Amer
Francisca	Gual	Mir	Antonio	Gual	Mateu	Francisca	Mir	Amer
Francisca	Gual	Mir	NULL	NULL	NULL	Francisca	Gual	Mir
Pixedis	Gual	Mir	Antonio	Gual	Mateu	Francisca	Mir	Amer
Francisca	Mir	Amer	NULL	NULL	NULL	NULL	NULL	NULL
Antonia	Mut	Xamena	NULL	NULL	NULL	NULL	NULL	NULL

11 rows in set (0.022 sec)

Let’s work!

Do the exercises:

- P04_queries01

4.12. Views (I).

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database).
- Consider a person who needs to know an employee's name and department, but not the salary. This person should see a relation described, in SQL, by:

```
select E.num, E.name, E.surname, D.name
from EMPLOYEES E, DEPARTMENTS D
where E.dept_num = D.num;
```

- A **view** provides a mechanism to **hide certain data** from the view of **certain users**.
- Any relation that is not part of the conceptual model but is made visible to a user as a "virtual relation" is called a view.

4.12. Views (II).

- A view is defined using the create view statement which has the form:

  create view v **as** <query expression>
  - where <query expression> is any legal SQL expression. The **view name** is represented by v.
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression.
- **YOU CAN DEFINE VIEWS USING OTHER VIEWS (INCLUDING ITSELF IN SOME DBMS)!**

4.12. Views (III).

- Example:

~~create view DEPARTMENTSFULL as~~
~~ select D.num, D.name, COUNT(E.num) AS num_employees~~
~~ from EMPLOYEES E, DEPARTMENTS D~~
~~ where E.dept_num = D.num~~
~~ group by D.num, D.name;~~

~~select * from DEPARTMENTSFULL;~~

```
+-----+------------+---------------+
| num | name       | num_employees |
+-----+------------+---------------+
|  10 | ACCOUNTING |             2 |
|  20 | RESEARCH   |             8 |
|  30 | SALES      |             5 |
+-----+------------+---------------+
3 rows in set (0.027 sec)
```

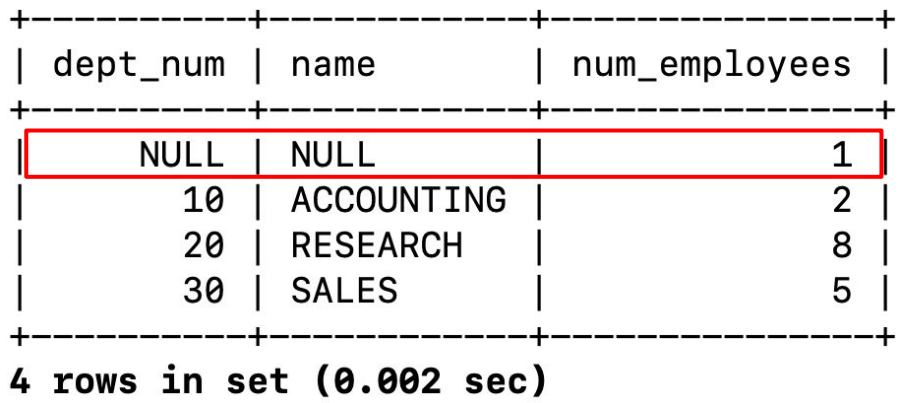
```
create view DEPARTMENTSFULL2 as
  select D.num, D.name, COUNT(E.num) AS num_employees
    from EMPLOYEES E right outer join DEPARTMENTS D
  on E.dept_num = D.num
  group by D.num, D.name;

select * from DEPARTMENTSFULL2;
```

```
+-----+------------+---------------+
| num | name       | num_employees |
+-----+------------+---------------+
|  10 | ACCOUNTING |             2 |
|  20 | RESEARCH   |             8 |
|  30 | SALES      |             5 |
|  40 | PRODUCTION |             0 |
+-----+------------+---------------+
4 rows in set (0.001 sec)
```

4.12. Views (IV).

- Example:

```
create view DEPARTMENTSFULL3 as
   select E.dept_num, D.name, COUNT(E.num) AS num_employees
         from EMPLOYEES E left outer join DEPARTMENTS D
   on E.dept_num = D.num
   group by E.dept_num, D.name;

select * from DEPARTMENTSFULL3;
```



dept_num	name	num_employees
NULL	NULL	1
10	ACCOUNTING	2
20	RESEARCH	8
30	SALES	5

4 rows in set (0.002 sec)

4.12. Views (V).

- Example:

```
create view EMPS as
    select E.num, E.surname, E.name, E.start_date,
        E.dept_num, D.name as dept_name
        from EMPLOYEES E left outer join DEPARTMENTS D
    on E.dept_num = D.num;


select * from EMPS;
```

num	surname	name	start_date	dept_num	dept_name
1000	PITT	BRAD	1984-01-01	20	RESEARCH
7369	SÁNCHEZ	SERGIO	1990-12-17	20	RESEARCH
7499	ARROYO	MARTA	1990-02-20	30	SALES
7521	SEGUI	RAUL	1991-02-22	30	SALES
7566	GUAL	JUDIT	1991-04-02	20	RESEARCH
7654	MARTÍN	MONICA	1991-09-29	NULL	NULL
7698	NEGRO	BARTOLOME	1991-05-01	30	SALES
7782	CRESPI	ENRIQUE	1991-06-09	10	ACCOUNTING
7788	GIL	JAIME	1991-11-09	20	RESEARCH
7844	TOVAR	LUIS	1991-09-08	30	SALES
7876	ALONSO	FERNANDO	1991-09-23	20	RESEARCH
7900	JUAREZ	XAVIER	1991-12-03	30	SALES
7902	FERNÁNDEZ	ANA	1991-12-03	20	RESEARCH
7934	MUÑOZ	ANTONIA	1992-01-23	10	ACCOUNTING
8000	BANDERAS	ANTONIO	1991-01-09	20	RESEARCH
8001	RUIZ	FERNANDA	1992-06-10	20	RESEARCH

16 rows in set (0.001 sec)

4.12. Views (VI).

- Example:

~~create view EMPS2 as~~
~~select E.num, E.surname, E.name, E.start_date,~~
~~E.dept_num, D.name as dept_name~~
~~from EMPLOYEES E right outer join DEPARTMENTS D~~
~~on E.dept_num = D.num;~~

select * from EMPS2;

num	surname	name	start_date	dept_num	dept_name
7782	CRESPI	ENRIQUE	1991-06-09	10	ACCOUNTING
7934	MUÑOZ	ANTONIA	1992-01-23	10	ACCOUNTING
1000	PITT	BRAD	1984-01-01	20	RESEARCH
7369	SÁNCHEZ	SERGIO	1990-12-17	20	RESEARCH
7566	GUAL	JUDIT	1991-04-02	20	RESEARCH
7788	GIL	JAIME	1991-11-09	20	RESEARCH
7876	ALONSO	FERNANDO	1991-09-23	20	RESEARCH
7902	FERNÁNDEZ	ANA	1991-12-03	20	RESEARCH
8000	BANDERAS	ANTONIO	1991-01-09	20	RESEARCH
8001	RUIZ	FERNANDA	1992-06-10	20	RESEARCH
7499	ARROYO	MARTA	1990-02-20	30	SALES
7521	SEGUI	RAUL	1991-02-22	30	SALES
7698	NEGRO	BARTOLOME	1991-05-01	30	SALES
7844	TOVAR	LUIS	1991-09-08	30	SALES
7900	JUAREZ	XAVIER	1991-12-03	30	SALES
NULL	NULL	NULL	NULL	NULL	PRODUCTION

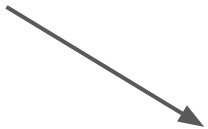
16 rows in set (0.001 sec)

4.12. Views (VII).

- Example:

```
create view EMPSFULL as
    select E.num, E.surname, E.name, E.start_date,
        E.dept_num, D.name as dept_name,
    E.manager as manager_num, M.name as manager_name, M.surname as manager_surname
        from EMPLOYEES E left outer join DEPARTMENTS D on E.dept_num = D.num
                                left outer join EMPLOYEES M on E.manager = M.num;
```

```
select * from EMPSFULL;
```



num	surname	name	start_date	dept_num	dept_name	manager_num	manager_name	manager_surname
1000	PITT	BRAD	1984-01-01	20	RESEARCH	NULL	NULL	NULL
7369	SÁNCHEZ	SERGIO	1990-12-17	20	RESEARCH	8001	FERNANDA	RUIZ
7499	ARROYO	MARTA	1990-02-20	30	SALES	7698	BARTOLOME	NEGRO
7521	SEGUI	RAUL	1991-02-22	30	SALES	7782	ENRIQUE	CRESPI
7566	GUAL	JUDIT	1991-04-02	20	RESEARCH	1000	BRAD	PITT
7654	MARTÍN	MONICA	1991-09-29	NULL	NULL	7698	BARTOLOME	NEGRO
7698	NEGRO	BARTOLOME	1991-05-01	30	SALES	1000	BRAD	PITT
7782	CRESPI	ENRIQUE	1991-06-09	10	ACCOUNTING	1000	BRAD	PITT
7788	GIL	JAIME	1991-11-09	20	RESEARCH	8000	ANTONIO	BANDERAS
7844	TOVAR	LUIS	1991-09-08	30	SALES	7698	BARTOLOME	NEGRO
7876	ALONSO	FERNANDO	1991-09-23	20	RESEARCH	7788	JAIME	GIL
7900	JUAREZ	XAVIER	1991-12-03	30	SALES	8001	FERNANDA	RUIZ
7902	FERNÁNDEZ	ANA	1991-12-03	20	RESEARCH	8000	ANTONIO	BANDERAS
7934	MUÑOZ	ANTONIA	1992-01-23	10	ACCOUNTING	8001	FERNANDA	RUIZ
8000	BANDERAS	ANTONIO	1991-01-09	20	RESEARCH	1000	BRAD	PITT
8001	RUIZ	FERNANDA	1992-06-10	20	RESEARCH	1000	BRAD	PITT

16 rows in set (0.001 sec)

4.12. Views (VIII).

- Example:

```
create or replace view EMPSFULL2 as
    select E.num, E.surname, E.name, E.start_date,
        E.dept_num, D.name as dept_name,
    M.num as manager_num, M.name as manager_name, M.surname as manager_surname
        from EMPLOYEES E left outer join DEPARTMENTS D on E.dept_num = D.num
                         left outer join EMPSFULL M on E.manager = M.num;
```

The behaviour of a view in the from clause it's just like another relation...

```
SELECT * FROM EMPSFULL2;
```

num	surname	name	start_date	dept_num	dept_name	manager_num	manager_name	manager_surname
7876	ALONSO	FERNANDO	1991-09-23	20	RESEARCH	7788	JAIME	GIL
7499	ARROYO	MARTA	1990-02-20	30	SALES	7698	BARTOLOME	NEGRO
8000	BANDERAS	ANTONIO	1991-01-09	20	RESEARCH	1000	BRAD	PITT
7782	CRESPI	ENRIQUE	1991-06-09	10	ACCOUNTING	1000	BRAD	PITT
7902	FERNÁNDEZ	ANA	1991-12-03	20	RESEARCH	8000	ANTONIO	BANDERAS
7788	GIL	JAIME	1991-11-09	20	RESEARCH	8000	ANTONIO	BANDERAS
7566	GUAL	JUDIT	1991-04-02	20	RESEARCH	1000	BRAD	PITT
7900	JUAREZ	XAVIER	1991-12-03	30	SALES	8001	FERNANDA	RUIZ
7654	MARTÍN	MONICA	1991-09-29	NULL	NULL	7698	BARTOLOME	NEGRO
7934	MUÑOZ	ANTONIA	1992-01-23	10	ACCOUNTING	8001	FERNANDA	RUIZ
7698	NEGRO	BARTOLOME	1991-05-01	30	SALES	1000	BRAD	PITT
1000	PITT	BRAD	1984-01-01	20	RESEARCH	NULL	NULL	NULL
8001	RUIZ	FERNANDA	1992-06-10	20	RESEARCH	1000	BRAD	PITT
7369	SÁNCHEZ	SERGIO	1990-12-17	20	RESEARCH	8001	FERNANDA	RUIZ
7521	SEGUI	RAUL	1991-02-22	30	SALES	7782	ENRIQUE	CRESPI
7844	TOVAR	LUIS	1991-09-08	30	SALES	7698	BARTOLOME	NEGRO

16 rows in set (0.004 sec)

4.13. Update of a View (I).

- In MySQL/MariaDB, views are not only query-able but **also updatable**. It means that you can use the INSERT or UPDATE statement to insert or update rows of the base table through the updatable view. In addition, you can use DELETE statement to remove rows of the underlying table through the view. (Source: here)

4.13. Update of a View (II).

- However, to create an updatable view (INSERT works), the SELECT statement that defines the view must not contain any of the following elements:
  - DISTINCT
  - Aggregate functions such as MIN, MAX, SUM, AVG, and COUNT.
  - GROUP BY clause.
  - HAVING clause.
  - UNION or UNION ALL clause.
  - Left join / Right join (or outer join).
  - Subquery in the SELECT clause or in the WHERE clause that refers to the table appeared in the FROM clause.
  - Reference to non-updatable view in the FROM clause.
  - Reference only to literal values.
  - Multiple references to any column of the base table.

4.13. Update of a View (III).

- Add a new tuple to EMPS view which we defined earlier:

```
insert into EMPSUPT (num, surname, name, start_date,
                        dept_num, dept_name) VALUES
                    (9999, 'GONZÁLEZ', 'SERGIO', '2019-1-3',
                    50, 'SECURITY');
insert into EMPSUPT (num, surname, name, start_date,
                    dept_num) VALUES
                    (9999, 'GONZÁLEZ', 'SERGIO', '2019-1-3',
                    30);
```

```
create view EMPSUPT as
   select E.num, E.surname, E.name, E.start_date,
        E.dept_num, D.name as dept_name
        from EMPLOYEES E left outer join DEPARTMENTS D
   on E.dept_num = D.num;
```

```
MariaDB [UNIT41]> insert into EMPSUPT (num, surname, name, start_date,
    -> dept_num, dept_name) VALUES
    -> (9999, 'GONZÁLEZ', 'SERGIO', '2019-1-3',
    -> 50, 'SECURITY');
ERROR 1471 (HY000): The target table EMPSUPT of the INSERT is not insertable-into
MariaDB [UNIT41]> insert into EMPSUPT (num, surname, name, start_date,
    -> dept_num) VALUES
    -> (9999, 'GONZÁLEZ', 'SERGIO', '2019-1-3',
    -> 30);
ERROR 1471 (HY000): The target table EMPSUPT of the INSERT is not insertable-into
```

4.13. Update of a View (IV).

- Create a new VIEW without left outer join (all the employees must have a department assigned!):

```
create view EMPSUPT2 as
   select E.num, E.surname, E.name, E.start_date
      from EMPLOYEES E;

insert into EMPSUPT2 (num, surname, name, start_date)
                  VALUES
                  (9999, 'GONZÁLEZ', 'SERGIO', '2019-1-3');
```

```
MariaDB [UNIT41]> insert into EMPSUPT2 (num, surname, name, start_date)
    -> VALUES
    -> (9999, 'GONZÁLEZ', 'SERGIO', '2019-1-3');
Query OK, 1 row affected (0.009 sec)
```

4.13. Update of a View (V).

- Create a new VIEW without left outer join (all the employees must have a department assigned!):

```
create view EMPSUPT3 as
   select E.num, E.surname, E.name, E.start_date,
          E.dept_num, D.name as dept_name
   from EMPLOYEES E, DEPARTMENTS D
   where E.dept_num = D.num;

insert into EMPSUPT3 (num, surname, name, start_date,
                      dept_num, dept_name) VALUES
                (9999, 'GONZÁLEZ', 'SERGIO', '2019-1-3',
                      30, 'SALES');
```

```
MariaDB [UNIT41]> insert into EMPSUPT3 (num, surname, name, start_date,
    -> dept_num, dept_name) VALUES
    -> (9999, 'GONZÁLEZ', 'SERGIO', '2019-1-3',
    -> 30, 'SALES');
ERROR 1393 (HY000): Can not modify more than one base table through a join view 'UNIT41.EMPSUPT3'
```

4.13. Update of a View (VI).

- Create a new VIEW without left outer join (all the employees must have a department assigned!):

```
create view EMPS_SALES as
   select E.num, E.surname, E.name, E.start_date,
          E.salary, E.commission
   from EMPLOYEES E
   where E.dept_num = 30;
```

- What happens if we insert (9998, 'DURAN', 'JAUME', '2019-1-3', 1500, 500) into *EMP_SALES?*

```
insert into EMPS_SALES
values
(9998, 'DURAN', 'JAUME', '2019-1-3', 1500, 500);
```

```
MariaDB [UNIT41]> insert into EMPS_SALES
    -> values
    -> (9998, 'DURAN', 'JAUME', '2019-1-3', 1500, 500);
Query OK, 1 row affected (0.005 sec)
```

4.14. Materialized Views (I).

- **Materializing a view**: create a physical table containing all the tuples in the result of the query defining the view.
- If relations used in the query are updated, the materialized view result becomes **out of date**…
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated.
- **CREATE MATERIALIZED VIEW doesn't exist in MySQL/MariaDB.**
- But you can emulate them… Check this link.

```
CREATE TABLE `user_stats` AS
  SELECT * FROM `DB-1`.USERS
WHERE ...
  UNION
  SELECT * FROM `DB-2`.USERS
WHERE ....
  UNION ...
```

4.14. Materialized Views (II).

Oracle's example (source here):

```
CREATE MATERIALIZED VIEW MV_Test
  NOLOGGING
  CACHE
  BUILD IMMEDIATE
  REFRESH FAST ON COMMIT
  AS
    SELECT V.*, P.*, V.ROWID as V_ROWID, P.ROWID as P_ROWID
    FROM TPM_PROJECTVERSION V,
         TPM_PROJECT P
    WHERE P.PROJECTID = V.PROJECTID
```

Let’s work!

Do the exercises:

- P04_views

4.15. Transactions (I).

- Transaction processing is used to maintain database integrity by ensuring that batches of SQL operations **execute completely or not at all**.
- In other words, a transaction is a **logical unit of work** meant to either be executed as a whole or to be cancelled entirely.

4.15. Transactions (II).

- Transactions are recoverable units of data access tasks in terms of database content manipulation.
- They also comprise units of recovery for the entire database in case of system crashes.
- They also provide basis for concurrency management in multi-user environment.

4.15. Transactions (III).

- Improper transaction management and control by the application software may result in:
  - customer orders, payments, and product shipment orders being lost in the case of a web store
  - failures in the registration of seat reservations or double-bookings to be made for train/airplane passengers
  - lost emergency call registrations at emergency response centers
  - etc.

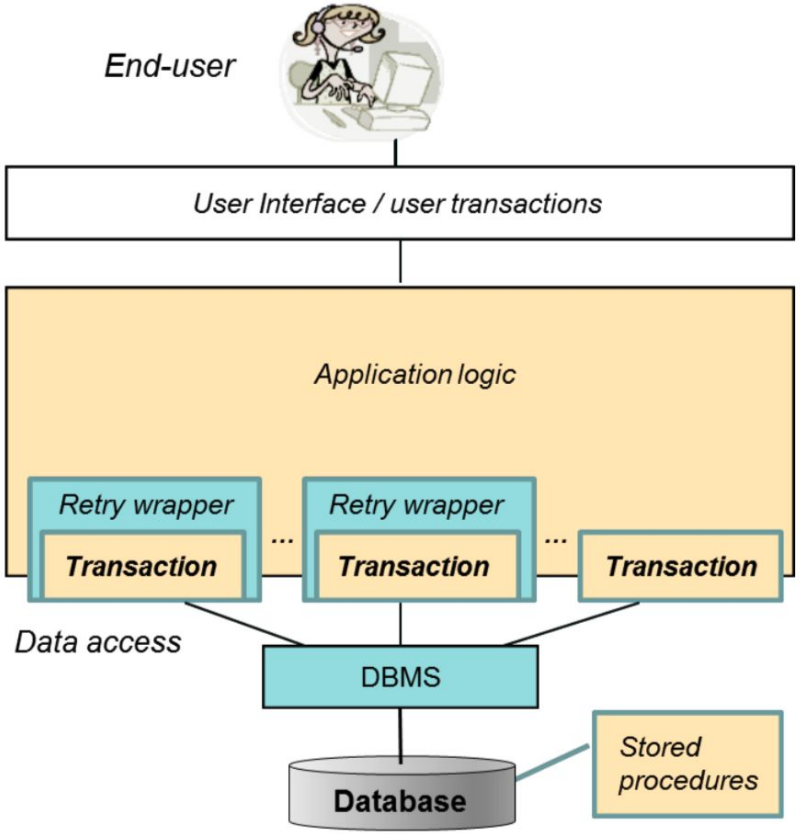
4.15. Transactions (IV).

- A typical textbook example of SQL transactions is the transferring of a certain amount (for example 100 €) from one account to another:

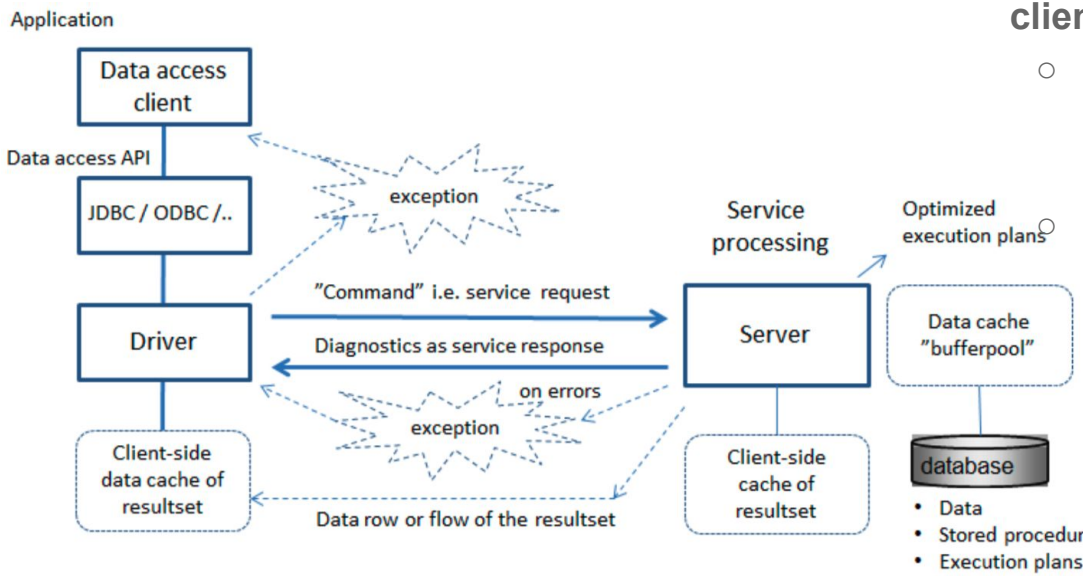
```
BEGIN TRANSACTION;
UPDATE Accounts SET balance = balance - 100 WHERE acctId = 101;
UPDATE Accounts SET balance = balance + 100 WHERE acctId = 202;
COMMIT;
```

- It can fail if:
  - An account doesn't exist. Therefore, one should inspect the available SQL diagnostics and check the number of rows that have been affected by each one of the two UPDATE commands.
  - The first account goes to a negative balance. You can implement this with a check constraint or with another query.

4.15. Transactions (V).



4.15. Transactions (VI).



- To properly understand SQL transactions, we need to agree on some basic concepts concerning the **client-server** handshaking dialogue:
  - To access a database, the application needs to **initiate a database connection** which sets up the context of an SQL-session.
  - From the database server point of view, the application uses database services in client/server mode by passing SQL commands as parameters to functions/methods via a data access API (application programming interface). Dialog with the server is based on the SQL language, and reliable data access is materialized with the proper use of SQL transactions.

4.15. Transactions (VII).

- MariaDB/MySQL: MyISAM engine doesn’t support transactions, so use **InnoDB engine**...

4.15. Transactions (VIII).

- In a nutshell:
  - **Transaction**: A block of SQL statements executed completely or not at all.
  - **Rollback**: The process of undoing specified SQL statements.
  - **Commit**: Writing unsaved SQL statements to the database tables.
  - **Savepoint**: A temporary placeholder in a transaction set to which you can issue a rollback (as opposed to rolling back an entire transaction).

4.15. Transactions (IX).

- Some DBMSs require that you explicitly mark the start and end of transaction blocks. In **SQL Server**, for example, you can do the following:
  BEGIN TRANSACTION;
  ...
  COMMIT TRANSACTION; / ROLLBACK;
- The equivalent code in **MySQL** is:
  START TRANSACTION
  …
  COMMIT / ROLLBACK
- **PostgreSQL** uses the ANSI SQL syntax:
  BEGIN; (or BEGIN TRANSACTION;)
  …
  COMMIT; (or END TRANSACTION;) / ROLLBACK;
- **Oracle**:
  START TRANSACTION or BEGIN
  …
  COMMIT / ROLLBACK

4.15. Transactions (X).

- The following is a complete **SQL Server** example:

```
BEGIN TRANSACTION
INSERT INTO Customers (cust_id, cust_name)
VALUES ('1000000010', 'Toys Emporium');
SAVE TRANSACTION StartOrder;
INSERT INTO Orders (order_num, order_date, cust_id)
VALUES (20100,'2001/12/1','1000000010');
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
INSERT INTO OrderItems(order_num, order_item,
prod_id, quantity, item_price)
VALUES(20010, 1, 'BR01', 100, 5.49);
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
INSERT INTO OrderItems(order_num, order_item,
prod_id, quantity, item_price)
VALUES(20010, 2, 'BR03', 100, 10.99);
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
COMMIT TRANSACTION
```

4.15. Transactions (XI).

- Simple ROLLBACK and COMMIT statements enable you to write or undo an entire transaction. Although this works for simple transactions, **more complex transactions might require partial commits or rollbacks**.
- In SQL, these placeholders are called **savepoints**. To create one in MySQL and Oracle, the SAVEPOINT statement is used, as follows:
  SAVEPOINT delete1;
- In SQL Server and Sybase, you do the following:
  SAVE TRANSACTION delete1;

4.15. Transactions (XII).

- **SAVEPOINT**: Sets a named transaction savepoint with a name of identifier. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set. In MariaDB:
  - SAVEPOINT myFirstSP;
- **ROLLBACK TO SAVEPOINT:** Rolls back a transaction to the named savepoint without terminating the transaction. In MariaDB:
  - ROLLBACK TO SAVEPOINT myFirstSP;
- **RELEASE SAVEPOINT**: Removes the named savepoint from the set of savepoints of the current transaction. All savepoints of the current transaction are deleted if you execute a **COMMIT**, or a ROLLBACK that does not name a savepoint. In MariaDB:
  - RELEASE SAVEPOINT myFirstSP;

4.15. Transactions (XIII).

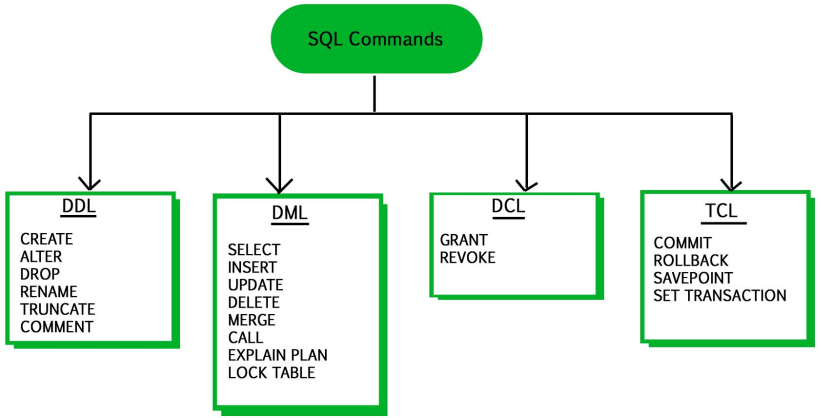
- When you write an SQL statement you perform an operation on some data. You can change data with DELETE, INSERT, or UPDATE statements.
- Two options:
    - Run the statement to change the data immediately.
    - Await a command to commit the changes to the database.
- By default, MySQL automatically commits the changes permanently to the database. To force MySQL/MariaDB not to commit changes automatically, you use the following statement:
    - SET autocommit = 0; OR SET autocommit = OFF;
- You use the following statement to enable the autocommit mode explicitly:
    - SET autocommit = 1; OR SET autocommit = ON;
- ORACLE's syntax:
    - autocommit off;
    - autocommit on;

4.15. Transactions (XIV).

- Some DBMS products, such as Oracle, implicitly commit transaction upon executing any SQL DDL statement (e.g. CREATE, ALTER or DROP of some database object, such as TABLE, INDEX, VIEW, etc.).

4.15. Transactions (X).

- **Transaction Control Language** (TCL) commands are used to manage transactions in a database.
- CAUTION: The exact syntax used to implement transaction processing differs from one DBMS to another. Check your DBMS documentation before proceeding.
- The key to managing transactions involves breaking your SQL statements into logical chunks and explicitly stating when data should be rolled back and when it should not.



Let’s work!

Do the exercises:

- P04_transactions
- P04_transactions_02

4.16. Integrity Constraints (I).

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency. Examples:
  - A savings account must have a balance greater than $10,000.00.
  - A salary of a bank employee must be at least $4.00 an hour.
  - A customer must have a (non-null) phone number.

4.16. Integrity Constraints (II).

- not null
- primary key
- foreign key
- unique
- check (P), where P is a predicate

4.16. Integrity Constraints (III).

- **primary key**
  - Every table needs an attribute to identify every row unambiguously, or set of attributes whose combined values are unique.
- **not null**
  - Declare fields of a table to be not null
- **unique**
  - The every value in the field/attribute/column is unique.

4.16. Integrity Constraints (IV).

- For example, the following SQL query creates a new table called CUSTOMERS and adds five columns. Here, the ID column is the primary key. Also ID, NAME, AGE, can not be NULL. Finally, the AGE column is set to UNIQUE, so that you cannot have two records with the same age.

```
CREATE TABLE CUSTOMERS(
   ID INT NOT NULL,
   NAME VARCHAR (20) NOT NULL,
   AGE INT NOT NULL UNIQUE,
   ADDRESS CHAR (25) ,
   SALARY DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

4.16. Integrity Constraints (V).

- **check (P)** (source here)
  - where P is a predicate
  - Example: ensure that department is one of SALES, RESEARCH, ACCOUNTING, and PRODUCTION:

In MariaDB 10.2.1 you can define constraints in 2 different ways:

- `CHECK(expression)` given as part of a column definition.
- `CONSTRAINT [constraint_name] CHECK (expression)`

```
CREATE TABLE `EMPLOYEES` (
  `num` int(11) NOT NULL,
  `surname` varchar(50) NOT NULL,
  `name` varchar(50) NOT NULL,
  `manager` int(11) DEFAULT NULL,
  `begin_date` date DEFAULT NULL,
  `salary` int(11) DEFAULT NULL,
  `comission` int(11) DEFAULT NULL,
  `department` VARCHAR(10) DEFAULT NULL,
  primary key (`num`),
  check (department in ('SALES', 'RESEARCH', 'ACCOUNTING',
'PRODUCTION'))
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

4.16. Integrity Constraints (VI).

- **check** (age >= 18)
- **check** (department in (select name from DEPARTMENTS))
    - why not use a foreign key here?
    - Unfortunately: subquery in check clause not supported by pretty much any database
- Alternative: **triggers** (later)

4.16. Integrity Constraints (V).

1.7. Relational integrity.

THE ENTITY INTEGRITY RULE

No component of the primary key of a base relation is allowed to accept nulls.

Remember that:

- Null may mean "**property does not apply**". For example, the supplier may be a country, in which case the attribute CITY has a null value because such property does not apply.
- Null may mean "**value is unknown**". For example, if the supplier is a person, then a null value for CITY attribute means we do not know the location of this supplier.
- Nulls cannot be in primary keys, **but can be in foreign keys**.
- EXAMPLE: DNI_partner can be NULL if you are not married.

1.7. Relational integrity.

THE REFERENTIAL INTEGRITY RULE

The database can not contain inconsistent foreign key values.

In other words, the valid values of a foreign key are:

- Existing values in the primary key of reference
- NULL values

Another way of saying it:

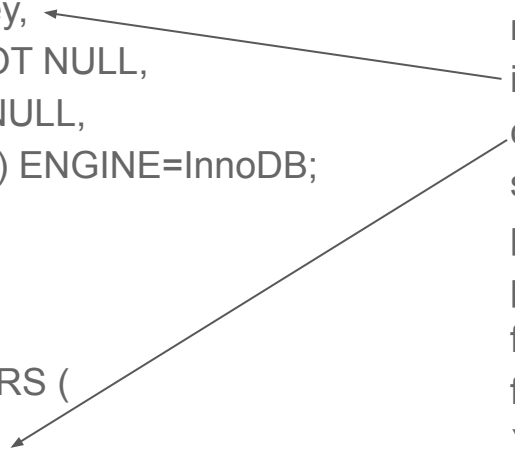
- The database must not contain any unmatched foreign key values.

4.17. Cascading Actions in Referential Integrity (I).

```
create table ITEMS (
        ID int primary key,
        category text NOT NULL,
        color text NOT NULL,
        primary key (ID)) ENGINE=InnoDB;
```

```
create table SALES (
        num int,
        itemID int,
        custID int,
        salesDate date,
        price float,
        primary key (num),
        foreign key (itemID) references ITEMS(ID),
        foreign key (custID) references CUSTOMERS(ID)
        ) ENGINE=InnoDB;
```

```
create table CUSTOMERS (
        ID int primary key,
        name VARCHAR(30),
        gender char NOT NULL,
        age int,
        primary key (num)) ENGINE=InnoDB;
```



If you delete a certain customer, what does it happen with his/her sales??

4.17. Cascading Actions in Referential Integrity (II).

```
create table SALES (
        [...]
        primary key (num),
        foreign key (itemID) references ITEMS(ID),
        foreign key (custID) references CUSTOMERS(ID)
                on delete cascade
                on update cascade
        ) ENGINE=InnoDB;
```

If you delete a certain article, what should we do?

- alternative actions to cascade:
  - **set null**
  - **set default**
  - **restrict**

on delete ?
on update ?

4.18. The ALTER and DROP command (I).

- **add a column**:

```
ALTER TABLE table_name
  ADD new_column_name column_definition
    [ FIRST | AFTER column_name ];
```

```
ALTER TABLE websites
  ADD host_name varchar(40)
    AFTER server_name;
```

- **modify a column**:

```
ALTER TABLE table_name
  MODIFY column_name column_definition
    [ FIRST | AFTER column_name ];
```

```
ALTER TABLE websites
  MODIFY host_name
varchar(50);
```

4.18. The ALTER and DROP command (II).

- **rename a column**:

```
ALTER TABLE table_name
  CHANGE COLUMN old_name new_name
    column_definition
    [ FIRST | AFTER column_name ]
```

```
ALTER TABLE websites
  CHANGE COLUMN
host_name hname
    varchar(25);
```

- **rename a table**:

```
ALTER TABLE table_name
  RENAME TO new_table_name;
```

```
ALTER TABLE websites
  RENAME TO sites;
```

4.18. The ALTER and DROP command (III).

- **add pk**:

```
ALTER TABLE table_name
  ADD PRIMARY KEY old_name new_name
    column_definition;
```

```
ALTER TABLE goods
  ADD PRIMARY KEY (id);
```

- **add fk**:

```
ALTER TABLE table_name
  ADD FOREIGN KEY (column_name)
REFERENCES table_name(coulmn_name)
```

```
ALTER TABLE users
  ADD CONSTRAINT
fk_grade_id FOREIGN KEY
(grade_id) REFERENCES
grades(id);
```

4.18. The ALTER and DROP command (IV).

- **drop a database**:

```
DROP {DATABASE | SCHEMA}
  [IF EXISTS] db_name
```

```
DROP DATABASE SQL1_NORMAL;
```

- **drop a table**:

```
DROP [TEMPORARY] TABLE [IF EXISTS] [/*COMMENT TO SAVE*/]
    tbl_name [, tbl_name] ...
    [WAIT n|NOWAIT]
    [RESTRICT | CASCADE]
```

```
DROP TABLE Employees,
Customers;
```

- **drop a column**:

```
ALTER TABLE table_name
  DROP COLUMN column_name;
```

```
ALTER TABLE websites
  DROP COLUMN host_name;
```

4.18. The ALTER and DROP command (IV).

- **drop pk**:

```
ALTER TABLE table_name
  DROP PRIMARY KEY;
```

```
ALTER TABLE customers DROP
PRIMARY KEY;
```

- **drop fk**:

```
ALTER TABLE <table_name>
  DROP FOREIGN KEY key_name;
```

```
ALTER TABLE customers
  DROP FOREIGN KEY fk1_customers;
```

4.18. The ALTER and DROP command (V).

- **All the ALTER reference**:
  - https://mariadb.com/kb/en/library/alter/
- **All the DROP reference:**
  - https://mariadb.com/kb/en/library/drop/

Let’s work!

Do the exercises:

- P04_integrity_alter_drop (I’ll give you the exercise solved)
- P04_BORJA_PHONE

4.19. Built-in Data Types in SQL (I).

- **date**: Dates, containing a (4 digit) year, month and date
  - Example: date ‘2005-7-27’
- **time**: Time of day, in hours, minutes and seconds.
  - Example: time ‘09:00:30’ time ‘09:00:30.75’
- **datetime**: date plus time of day (not inserted automatically)
  - Example: datetime ‘2005-7-27 09:00:30.75’
- **timestamp**: date plus time of day (inserted automatically)
  - Example: timestamp ‘2005-7-27 09:00:30.75’
- **interval**: period of time
  - Example: interval 2 day
  - Example: interval 2 month
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values (a date will be returned)

4.20. Date functions and operators.

Operator	Example	Result source
+	date '2012-08-08' + interval 2 day	2012-08-10
+	time '01:00' + interval 3 hour	04:00:00.000
+	timestamp '2012-08-08 01:00' + interval 29 hour	2012-08-09 06:00:00.000
+	timestamp '2012-10-31 01:00' + interval 1 month	2012-11-30 01:00:00.000
+	interval '2' day + interval 3 hour	2 03:00:00.000
+	interval '3' year + interval 5 month	3-5
-	date '2012-08-08' - interval 2 day	2012-08-06
-	time '01:00' - interval 3 hour	22:00:00.000
-	timestamp '2012-08-08 01:00' - interval 29 hour	2012-08-06 20:00:00.000
-	timestamp '2012-10-31 01:00' - interval 1 month	2012-09-30 01:00:00.000
-	date '2012-08-08' - interval 2 day - interval 3 hour	2012-08-05 21:00:00
-	date '2012-08-08' - interval 3 year - interval 5 month	2009-03-08

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **ADDDATE:** Add days or another interval to a date.

```
SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
+-----------------------------------------+
| ADDDATE('2008-01-02', INTERVAL 31 DAY) |
+-----------------------------------------+
| 2008-02-02                              |
+-----------------------------------------+
```

  - **ADDTIME:** Adds a time to a time or datetime.

```
SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
+-----------------------------------------------------------+
| ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002') |
+-----------------------------------------------------------+
| 2008-01-02 01:01:01.000001                                |
+-----------------------------------------------------------+
```

  - **CONVERT_TZ:** Converts a datetime from on time zone to another.

```
SELECT CONVERT_TZ('2016-01-01 12:00:00','+00:00','+10:00');
+-------------------------------------------------------+
| CONVERT_TZ('2016-01-01 12:00:00','+00:00','+10:00') |
+-------------------------------------------------------+
| 2016-01-01 22:00:00                                   |
+-------------------------------------------------------+
```

Time Zone Conversion (more examples here)

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **DATE:** Extracts the date portion of a datetime.

```
SELECT DATE('2013-07-18 12:21:32');
+-----------------------------+
| DATE('2013-07-18 12:21:32') |
+-----------------------------+
| 2013-07-18                  |
+-----------------------------+
```

  - **DATEDIFF:** Difference in days between two date/time values.

```
SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
+-----------------------------------------------+
| DATEDIFF('2007-12-31 23:59:59','2007-12-30') |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+

SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
+-----------------------------------------------+
| DATEDIFF('2010-11-30 23:59:59','2010-12-31') |
+-----------------------------------------------+
|                                           -31 |
+-----------------------------------------------+
```

```
SELECT d, DATEDIFF(NOW(),d) FROM t1;
+---------------------+--------------------+
| d                   | DATEDIFF(NOW(),d) |
+---------------------+--------------------+
| 2007-01-30 21:31:07 |               1574 |
| 1983-10-15 06:42:51 |              10082 |
| 2011-04-21 12:34:56 |                 32 |
| 2011-10-30 06:31:41 |               -160 |
| 2011-01-30 14:03:25 |                113 |
| 2004-10-07 11:19:34 |               2419 |
+---------------------+--------------------+
```

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **DATE_ADD:** Date arithmetic - addition.

```
SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
+-----------------------------------------+
| DATE_ADD('2008-01-02', INTERVAL 31 DAY) |
+-----------------------------------------+
| 2008-02-02                              |
+-----------------------------------------+
```

  - **DATE_FORMAT:** Formats the date value according to the format string.

```
SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
+-------------------------------------------+
| DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y') |
+-------------------------------------------+
| Sunday October 2009                       |
+-------------------------------------------+
```

```
SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
+-------------------------------------------+
| DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s') |
+-------------------------------------------+
| 22:23:00                                  |
+-------------------------------------------+
```

Check all options here.

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **DATE_SUB:** Date arithmetic - subtraction.

```
SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
+-------------------------------------------+
| DATE_SUB('1998-01-02', INTERVAL 31 DAY) |
+-------------------------------------------+
| 1997-12-02                                |
+-------------------------------------------+
```

  - **DAY:** Synonym for DAYOFMONTH().
  - **DAYNAME:** Return the name of the weekday.

```
SELECT DAYNAME('2007-02-03');
+-----------------------+
| DAYNAME('2007-02-03') |
+-----------------------+
| Saturday              |
+-----------------------+
```

  - **DAYOFMONTH:** Returns the day of the month.
  - **DAYOFWEEK:** Returns the day of the week index (1 = Sunday, 2 = Monday, ..., 7 = Saturday).

```
SELECT DAYOFMONTH('2007-02-03');
+-----------------------------+
| DAYOFMONTH('2007-02-03') |
+-----------------------------+
|                           3 |
+-----------------------------+
```

```
SELECT DAYOFWEEK('2007-02-03');
+----------------------------+
| DAYOFWEEK('2007-02-03') |
+----------------------------+
|                          7 |
+----------------------------+
```

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **DAYOFYEAR:** Returns the day of the year.

```
SELECT DAYOFYEAR('2018-02-16');
+--------------------------+
| DAYOFYEAR('2018-02-16') |
+--------------------------+
|                       47 |
+--------------------------+
```

  - **EXTRACT:** Extracts a portion of the date.

```
SELECT EXTRACT(YEAR FROM '2009-07-02');
+-----------------------------------+
| EXTRACT(YEAR FROM '2009-07-02') |
+-----------------------------------+
|                              2009 |
+-----------------------------------+

SELECT EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03');
+-------------------------------------------------+
| EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03') |
+-------------------------------------------------+
|                                          200907 |
+-------------------------------------------------+
```

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **FROM_DAYS:** Returns a date given a day.

```
SELECT FROM_DAYS(730669);
+--------------------+
| FROM_DAYS(730669) |
+--------------------+
| 2000-07-03         |
+--------------------+
```

  - **GET_FORMAT:** Returns a format string.

```
SELECT GET_FORMAT(DATE, 'EUR');
+-------------------------+
| GET_FORMAT(DATE, 'EUR') |
+-------------------------+
| %d.%m.%Y                |
+-------------------------+
```

```
SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
+--------------------------------------------------+
| DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR')) |
+--------------------------------------------------+
| 03.10.2003                                       |
+--------------------------------------------------+
```

  - **HOUR:** Returns the hour.

```
SELECT HOUR('10:05:03');
+------------------+
| HOUR('10:05:03') |
+------------------+
|               10 |
+------------------+
```

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **LAST_DAY:** Returns the last day of the month.

```
SELECT LAST_DAY('2003-02-05');
+-------------------------+
| LAST_DAY('2003-02-05') |
+-------------------------+
| 2003-02-28             |
+-------------------------+
```

  - **LOCALTIME:** Synonym for NOW().
  - **LOCALTIMESTAMP:** Synonym for NOW().
  - **MAKEDATE:** Returns a date given a year and day.
  - **MAKETIME:** Returns a time.

```
SELECT MAKETIME(13,57,33);
+--------------------+
| MAKETIME(13,57,33) |
+--------------------+
| 13:57:33           |
+--------------------+
```

```
SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
+------------------+------------------+
| MAKEDATE(2011,31) | MAKEDATE(2011,32) |
+------------------+------------------+
| 2011-01-31       | 2011-02-01       |
+------------------+------------------+

SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
+------------------+------------------+
| MAKEDATE(2011,365) | MAKEDATE(2014,365) |
+------------------+------------------+
| 2011-12-31       | 2014-12-31       |
+------------------+------------------+

SELECT MAKEDATE(2011,0);
+-----------------+
| MAKEDATE(2011,0) |
+-----------------+
| NULL            |
+-----------------+
```

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **MICROSECOND**: Returns microseconds from a date or datetime.

```
SELECT MICROSECOND('12:00:00.123456');
+-----------------------------------+
| MICROSECOND('12:00:00.123456') |
+-----------------------------------+
|                            123456 |
+-----------------------------------+
```

```
SELECT MINUTE('2013-08-03 11:04:03');
+--------------------------------+
| MINUTE('2013-08-03 11:04:03') |
+--------------------------------+
|                              4 |
+--------------------------------+
```

  - **MINUTE:** Returns a minute from 0 to 59.
  - **MONTH:** Returns a month from 1 to 12.
  - **MONTHNAME:** Returns the full name of the month.
  - **NOW:** Returns the current date and time.
  - **PERIOD_ADD:** Add months to a period.

```
SELECT MONTH('2019-01-03');
+----------------------+
| MONTH('2019-01-03') |
+----------------------+
|                    1 |
+----------------------+
```

```
SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2010-03-27 13:13:25 |
+---------------------+
```

```
SELECT PERIOD_ADD(200801,2);
+-----------------------+
| PERIOD_ADD(200801,2) |
+-----------------------+
|                200803 |
+-----------------------+
```

```
SELECT MONTHNAME('2019-02-03');
+--------------------------+
| MONTHNAME('2019-02-03') |
+--------------------------+
| February                 |
+--------------------------+
```

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **PERIOD_DIFF:** Number of months between two periods.

```
SELECT PERIOD_DIFF(200802,200703);
+-----------------------------+
| PERIOD_DIFF(200802,200703) |
+-----------------------------+
|                          11 |
+-----------------------------+
```

  - **QUARTER:** Returns year quarter from 1 to 4.

```
SELECT QUARTER('2008-04-01');
+-------------------------+
| QUARTER('2008-04-01') |
+-------------------------+
|                     2 |
+-------------------------+
```

  - **SECOND:** Returns the second of a time.

```
SELECT SECOND('10:05:03');
+--------------------+
| SECOND('10:05:03') |
+--------------------+
|                  3 |
+--------------------+
```

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **SEC_TO_TIME:** Converts a second to a time.

```
SELECT SEC_TO_TIME(12414);
+--------------------+
| SEC_TO_TIME(12414) |
+--------------------+
| 03:26:54           |
+--------------------+
```

```
INSERT INTO custorder
  VALUES ('Kevin', 'yes' , STR_TO_DATE('1-01-2012', '%d-%m-%Y') ) ;
```

  - **STR_TO_DATE:** Converts a string to date (it's the inverse of the DATE_FORMAT(), all the options here).

```
SELECT STR_TO_DATE('Wednesday, June 2, 2014', '%W, %M %e, %Y');
+-----------------------------------------------------+
| STR_TO_DATE('Wednesday, June 2, 2014', '%W, %M %e, %Y') |
+-----------------------------------------------------+
| 2014-06-02                                          |
+-----------------------------------------------------+
```

  - **SUBDATE:** Subtract a date unit or number of days.

```
SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
+-----------------------------------------+
| DATE_SUB('2008-01-02', INTERVAL 31 DAY) |
+-----------------------------------------+
| 2007-12-02                              |
+-----------------------------------------+
```

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **SUBTIME:** Subtracts a time from a date/time.

```
SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
+-----------------------------------------------------------+
| SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002') |
+-----------------------------------------------------------+
| 2007-12-30 22:58:58.999997                                |
+-----------------------------------------------------------+
```

  - **SYSDATE:** Returns the current date and time.
  - **TIME:** Extracts the time.

```
SELECT TIME('2003-12-31 01:02:03');
+-----------------------------+
| TIME('2003-12-31 01:02:03') |
+-----------------------------+
| 01:02:03                    |
+-----------------------------+
```

```
SELECT TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001'
+-----------------------------------------------------------------+
| TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001') |
+-----------------------------------------------------------------+
| -00:00:00.000001                                                |
+-----------------------------------------------------------------+
```

  - **TIMEDIFF:** Returns the difference between two date/times.

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **TIMESTAMP:** Return the datetime, or add a time to a date/time.
  - **TIMESTAMPADD:** Add interval to a date or datetime.
  - **TIMESTAMPDIFF:** Difference between two datetimes.

```
SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
+-----------------------------------------------+
| TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01') |
+-----------------------------------------------+
|                                             3 |
+-----------------------------------------------+

SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
+----------------------------------------------+
| TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01') |
+----------------------------------------------+
|                                           -1 |
+----------------------------------------------+

SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
+-------------------------------------------------------+
| TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55') |
+-------------------------------------------------------+
|                                                128885 |
+-------------------------------------------------------+
```

```
SELECT TIMESTAMP('2003-12-31');
+-------------------------+
| TIMESTAMP('2003-12-31') |
+-------------------------+
| 2003-12-31 00:00:00     |
+-------------------------+

SELECT TIMESTAMP('2003-12-31 12:00:00','6:30:00');
+-----------------------------------------------+
| TIMESTAMP('2003-12-31 12:00:00','6:30:00') |
+-----------------------------------------------+
| 2003-12-31 18:30:00                           |
+-----------------------------------------------+
```

```
SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
+-------------------------------------+
| TIMESTAMPADD(MINUTE,1,'2003-01-02') |
+-------------------------------------+
| 2003-01-02 00:01:00                 |
+-------------------------------------+

SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
+-----------------------------------+
| TIMESTAMPADD(WEEK,1,'2003-01-02') |
+-----------------------------------+
| 2003-01-09                        |
+-----------------------------------+
```

4.20. Date functions and operators.

- **Date and Time Functions**:
  - **TIME_FORMAT:** Formats the time value according to the format string.
  - **TIME_TO_SEC:** Returns the time argument, converted to seconds.

```
SELECT TIME_TO_SEC('00:39:38');
+--------------------------+
| TIME_TO_SEC('00:39:38') |
+--------------------------+
|                     2378 |
+--------------------------+
```

  - **TO_DAYS:** Number of days since year 0.
  - **TO_SECONDS:** Number of seconds since year 0.

```
SELECT TO_SECONDS('2013-06-13');
+-----------------------------+
| TO_SECONDS('2013-06-13') |
+-----------------------------+
|                63538300800 |
+-----------------------------+
```

```
SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
+-----------------------------------------------+
| TIME_FORMAT('100:00:00', '%H %k %h %I %l') |
+-----------------------------------------------+
| 100 100 04 04 4                               |
+-----------------------------------------------+
```

```
SELECT TO_DAYS('2007-10-07');
+-----------------------+
| TO_DAYS('2007-10-07') |
+-----------------------+
|                733321 |
+-----------------------+

SELECT TO_DAYS('0000-01-01');
+-----------------------+
| TO_DAYS('0000-01-01') |
+-----------------------+
|                     1 |
+-----------------------+

SELECT TO_DAYS(950501);
+-----------------+
| TO_DAYS(950501) |
+-----------------+
|          728779 |
+-----------------+
```

4.20. Date functions and operators (VII).

- **Date and Time Functions**:
  - **UTC_DATE:** Returns the current UTC date.

```
SELECT UTC_DATE(), UTC_DATE() + 0;
+------------+------------------+
| UTC_DATE() | UTC_DATE() + 0   |
+------------+------------------+
| 2010-03-27 |         20100327 |
+------------+------------------+
```

Coordinated Universal Time

  - **UTC_TIME:** Returns the current UTC time.

```
SELECT UTC_TIME(), UTC_TIME() + 0;
+------------+------------------+
| UTC_TIME() | UTC_TIME() + 0   |
+------------+------------------+
| 17:32:34   |  173234.000000   |
+------------+------------------+
```

  - **UTC_TIMESTAMP:** Returns the current UTC date and time.

```
SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
+---------------------+------------------------+
| UTC_TIMESTAMP()     | UTC_TIMESTAMP() + 0    |
+---------------------+------------------------+
| 2010-03-27 17:33:16 | 20100327173316.000000  |
+---------------------+------------------------+
```

4.20. Date functions and operators (VII).

- **Date and Time Functions**:
  - **WEEK:** Returns the week number.
  - **WEEKDAY:** Returns the weekday index (0 = Monday, 1 = Tuesday, ... 6 = Sunday).
  - **WEEKOFYEAR:** Returns the calendar week of the date as a number in the range from 1 to 53.
  - **YEAR:** Returns the year for the given date.
  - **YEARWEEK:** Returns year and week for a date.

```
SELECT WEEK('2008-02-20');
+----------------------+
| WEEK('2008-02-20') |
+----------------------+
|                    7 |
+----------------------+
```

```
SELECT WEEKDAY('2008-02-03 22:23:00');
+--------------------------------+
| WEEKDAY('2008-02-03 22:23:00') |
+--------------------------------+
|                              6 |
+--------------------------------+
```

```
SELECT WEEKOFYEAR('2008-02-20');
+--------------------------+
| WEEKOFYEAR('2008-02-20') |
+--------------------------+
|                        8 |
+--------------------------+
```

```
SELECT YEAR('1987-01-01');
+---------------------+
| YEAR('1987-01-01') |
+---------------------+
|                1987 |
+---------------------+
```

```
SELECT YEARWEEK('1987-01-01');
+-------------------------+
| YEARWEEK('1987-01-01') |
+-------------------------+
|                  198652 |
+-------------------------+
```

4.20. Date functions and operators (VIII).

- **Complete reference for Date & Time Functions**:
  - https://mariadb.com/kb/en/library/date-time-functions/

Let’s work!

Do the exercises:

- P04_dates

4.21. String functions and operators.

- **BIN**: Returns binary value.

```
SELECT BIN(12);
+---------+
| BIN(12) |
+---------+
| 1100    |
+---------+
```

- **CHAR:**

```
SELECT CHAR(77,97,114,'105',97,'68',66);
+-----------------------------------+
| CHAR(77,97,114,'105',97,'68',66) |
+-----------------------------------+
| MariaDB                           |
+-----------------------------------+
```

- **CHAR_LENGTH**: Length of the string in characters.

```
SELECT CHAR_LENGTH('MariaDB');
+------------------------+
| CHAR_LENGTH('MariaDB') |
+------------------------+
|                      7 |
+------------------------+
```

- **CHR:**

```
SELECT CHR(67);
+---------+
| CHR(67) |
+---------+
| C       |
+---------+

SELECT CHR('67');
+-----------+
| CHR('67') |
+-----------+
| C         |
+-----------+
```

4.21. String functions and operators.

- **CONCAT**: Returns concatenated string.

```
SELECT CONCAT('Ma', 'ria', 'DB');
+----------------------------+
| CONCAT('Ma', 'ria', 'DB') |
+----------------------------+
| MariaDB                    |
+----------------------------+
```

- **CONCAT_WS**: Concatenate with separator.

```
SELECT CONCAT_WS(',','First name','Second name','Last Name');
+-----------------------------------------------------------+
| CONCAT_WS(',','First name','Second name','Last Name') |
+-----------------------------------------------------------+
| First name,Second name,Last Name                          |
+-----------------------------------------------------------+
```

- **ELT**: Returns the N'th element from a set of strings.

```
SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
+-------------------------------------+
| ELT(1, 'ej', 'Heja', 'hej', 'foo') |
+-------------------------------------+
| ej                                  |
+-------------------------------------+

SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
+-------------------------------------+
| ELT(4, 'ej', 'Heja', 'hej', 'foo') |
+-------------------------------------+
| foo                                 |
+-------------------------------------+
```

4.21. String functions and operators.

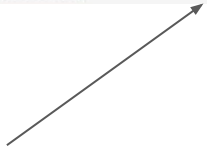
- **EXTRACTVALUE**: Returns the text of the first text node matched by the XPath expression.

```
SELECT
   EXTRACTVALUE('<cases><case>example</case></cases>', '/cases/case') AS 'Base Example',
   EXTRACTVALUE('<cases><case>example</case></cases>', '/cases/case/text()') AS 'text() Example';

+--------------+-----------------+
| Base Example | text() Example |
+--------------+-----------------+
| example      | example        |
+--------------+-----------------+
```

- **FIELD**: Returns the index position of a string in a list.

```
SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo')
   AS 'Field Results';
+-----------------+
| Field Results |
+-----------------+
|               2 |
+-----------------+
```

```
SELECT FIND_IN_SET('b','a,b,c,d') AS "Found Results";
+-----------------+
| Found Results |
+-----------------+
|               2 |
+-----------------+
```



- **FIND_IN_SET**: Returns the position of a string in a set of strings.

4.21. String functions and operators.

- **FORMAT**: Formats a number.

```
SELECT FORMAT(1234567890.09876543210, 4) AS 'Format';
+---------------------+
| Format              |
+---------------------+
| 1,234,567,890.0988  |
+---------------------+
```

```
SELECT HEX(255);
+-----------+
| HEX(255)  |
+-----------+
| FF        |
+-----------+
```

```
SELECT HEX('MariaDB');
+------------------+
| HEX('MariaDB')   |
+------------------+
| 4D617269614442   |
+------------------+
```

- **HEX**: Returns hexadecimal value.
- **INSERT**: Replaces a part of a string with another string.

```
SELECT INSERT('Quadratic', 3, 4, 'What');
+-----------------------------------+
| INSERT('Quadratic', 3, 4, 'What') |
+-----------------------------------+
| QuWhattic                         |
+-----------------------------------+
```

```
mysql> SELECT INSTR('myteststring','st');
+----------------------------+
| INSTR('myteststring','st') |
+----------------------------+
|                          5 |
+----------------------------+
1 row in set (0.03 sec)
```

- **INSTR**: Returns the position of a string within a string.

4.21. String functions and operators.

- **LCASE**: Synonym for LOWER().
- **LEFT**: Returns the leftmost characters from a string.

```
SELECT LEFT('MariaDB', 5);
+--------------------+
| LEFT('MariaDB', 5) |
+--------------------+
| Maria              |
+--------------------+
```

- **LENGTH**: Length of the string in bytes.

```
SELECT LENGTH('MariaDB');
+--------------------+
| LENGTH('MariaDB') |
+--------------------+
|                  7 |
+--------------------+
```

- **LOCATE**: Returns the position of a substring in a string.

```
SELECT LOCATE('bar', 'foobarbar');
+-----------------------------+
| LOCATE('bar', 'foobarbar') |
+-----------------------------+
|                           4 |
+-----------------------------+
```

4.21. String functions and operators.

- **LOWER**: Returns a string with all characters changed to lowercase.

```
SELECT LOWER('QUADRATICALLY');
+-------------------------+
| LOWER('QUADRATICALLY') |
+-------------------------+
| quadratically          |
+-------------------------+
```

- **LPAD**: Returns the string left-padded with another string to a given length.

```
SELECT LPAD('hello',10,'.');
+-----------------------+
| LPAD('hello',10,'.') |
+-----------------------+
| .....hello           |
+-----------------------+
```

- **LTRIM**: Returns the string with leading space characters removed.

```
SELECT QUOTE(LTRIM('   MariaDB   '));
+-----------------------------------+
| QUOTE(LTRIM('   MariaDB   ')) |
+-----------------------------------+
| 'MariaDB   '                      |
+-----------------------------------+
```

- **MID**: Synonym for SUBSTRING(str,pos,len).

```
SELECT MID('abcd',2,2);
+-----------------+
| MID('abcd',2,2) |
+-----------------+
| bc              |
+-----------------+
```

4.21. String functions and operators.

- **ORD**: Return ASCII or character code.

```
SELECT ORD('2');
+----------+
| ORD('2') |
+----------+
|       50 |
+----------+
```

```
MariaDB [SQL1_NORMAL]> SELECT POSITION('de' IN 'abcdefgh');
+-----------------------------+
| POSITION('de' IN 'abcdefgh') |
+-----------------------------+
|                           4 |
+-----------------------------+
1 row in set (0.00 sec)
```

- **POSITION**: Returns the position of a substring in a string.
- **QUOTE**: Returns quoted, properly escaped string.
- **REPEAT**: Returns a string repeated a number of times.
- **REPLACE**: Replace occurrences of a string.

```
SELECT QUOTE("Don't!");
+-------------------+
| QUOTE("Don't!") |
+-------------------+
| 'Don\'t!'         |
+-------------------+

SELECT QUOTE(NULL);
+--------------+
| QUOTE(NULL) |
+--------------+
| NULL         |
+--------------+
```

```
SELECT REPLACE('www.mariadb.org', 'w', 'Ww');
+---------------------------------------+
| REPLACE('www.mariadb.org', 'w', 'Ww') |
+---------------------------------------+
| WwWwWw.mariadb.org                    |
+---------------------------------------+
```

4.21. String functions and operators.

- **REVERSE**: Reverses the order of a string.

```
SELECT REVERSE('desserts');
+----------------------+
| REVERSE('desserts') |
+----------------------+
| stressed             |
+----------------------+
```

- **RIGHT**: Returns the rightmost N characters from a string.

```
SELECT RIGHT('MariaDB', 2);
+---------------------+
| RIGHT('MariaDB', 2) |
+---------------------+
| DB                  |
+---------------------+
```

- **RPAD**: Returns the string right-padded with another string to a given length.

```
SELECT RPAD('hello',10,'.');
+-----------------------+
| RPAD('hello',10,'.') |
+-----------------------+
| hello.....            |
+-----------------------+
```

4.21. String functions and operators.

- **RTRIM**: Returns the string with trailing space characters removed.

```
SELECT QUOTE(RTRIM('MariaDB    '));
+-----------------------------------+
| QUOTE(RTRIM('MariaDB    ')) |
+-----------------------------------+
| 'MariaDB'                         |
+-----------------------------------+
```

- **SPACE**: Returns a string of space characters.

```
SELECT QUOTE(SPACE(6));
+------------------+
| QUOTE(SPACE(6)) |
+------------------+
| '      '         |
+------------------+
```

- **STRCMP**: Compares two strings in sort order.

```
SELECT STRCMP('text', 'text2');
+-----------------------------+
| STRCMP('text', 'text2') |
+-----------------------------+
|                          -1 |
+-----------------------------+

SELECT STRCMP('text2', 'text');
+-----------------------------+
| STRCMP('text2', 'text') |
+-----------------------------+
|                           1 |
+-----------------------------+

SELECT STRCMP('text', 'text');
+-----------------------------+
| STRCMP('text', 'text') |
+-----------------------------+
|                           0 |
+-----------------------------+
```

4.21. String functions and operators.

- **SUBSTR**: SUBSTR() is a synonym for SUBSTRING().
- **SUBSTRING**: Returns a substring from string starting at a given position.
- **SUBSTRING_INDEX**: Returns the substring from string before count occurrences of a delimiter.

```
SELECT SUBSTRING_INDEX('www.mariadb.org', '.', 2);
+--------------------------------------------+
| SUBSTRING_INDEX('www.mariadb.org', '.', 2) |
+--------------------------------------------+
| www.mariadb                                |
+--------------------------------------------+

SELECT SUBSTRING_INDEX('www.mariadb.org', '.', -2);
+---------------------------------------------+
| SUBSTRING_INDEX('www.mariadb.org', '.', -2) |
+---------------------------------------------+
| mariadb.org                                 |
+---------------------------------------------+
```

```
SELECT SUBSTRING('Knowledgebase',5);
+-------------------------------+
| SUBSTRING('Knowledgebase',5) |
+-------------------------------+
| ledgebase                     |
+-------------------------------+

SELECT SUBSTRING('MariaDB' FROM 6);
+-----------------------------------+
| SUBSTRING('MariaDB' FROM 6) |
+-----------------------------------+
| DB                                |
+-----------------------------------+

SELECT SUBSTRING('Knowledgebase',3,7);
+---------------------------------+
| SUBSTRING('Knowledgebase',3,7) |
+---------------------------------+
| owledge                         |
+---------------------------------+

SELECT SUBSTRING('Knowledgebase', -4);
+--------------------------------+
| SUBSTRING('Knowledgebase', -4) |
+--------------------------------+
| base                           |
+--------------------------------+

SELECT SUBSTRING('Knowledgebase', -8, 4);
+-----------------------------------+
| SUBSTRING('Knowledgebase', -8, 4) |
+-----------------------------------+
| edge                              |
+-----------------------------------+

SELECT SUBSTRING('Knowledgebase' FROM -8 FOR 4);
+-------------------------------------------+
| SUBSTRING('Knowledgebase' FROM -8 FOR 4) |
+-------------------------------------------+
| edge                                      |
+-------------------------------------------+
```

4.21. String functions and operators.

- **TRIM**: Returns a string with all given prefixes or suffixes removed.
- **UCASE**: Synonym for UPPER().
- **UPPER**: Changes string to uppercase.

```
SELECT TRIM('  bar   ')\G
*************************** 1. row ***************************
TRIM('  bar   '): bar

SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx')\G
*************************** 1. row ***************************
TRIM(LEADING 'x' FROM 'xxxbarxxx'): barxxx

SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx')\G
*************************** 1. row ***************************
TRIM(BOTH 'x' FROM 'xxxbarxxx'): bar

SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz')\G
*************************** 1. row ***************************
TRIM(TRAILING 'xyz' FROM 'barxxyz'): barx
```

```
SELECT UPPER(surname), givenname FROM users ORDER BY surname;
+-----------------+-------------+
| UPPER(surname) | givenname   |
+-----------------+-------------+
| ABEL           | Jacinto     |
| CASTRO         | Robert      |
| COSTA          | Phestos     |
| MOSCHELLA      | Hippolytos  |
+-----------------+-------------+
```

4.21. String functions and operators.

Examples of use:

- SELECT E.surname, E.name,
  DATE_FORMAT(E.begin_date,'%W %D %M %Y %T') AS begin_date
  FROM EMPLOYEES E
  WHERE MONTHNAME(E.begin_date) = 'June';
- UPDATE your_table
  SET your_field = REPLACE(your_field, 'articles/updates/', 'articles/news/')
  WHERE your_field LIKE '%articles/updates/%'
- UPDATE film SET title = REPLACE(title, "AGENT TRUMAN", "AGENT WILSON");
- INSERT INTO CONTRACTES VALUES (null, 1, 1, 'N', 'CON_U01PT',
  str_to_date('31/01/2004','%d/%m/%Y'), str_to_date('12/03/2005', '%d/%m/%Y'));

Source: https://stackoverflow.com/questions/5956993/mysql-string-replace

4.21. String functions and operators.

- Complete reference for String Functions:
  - https://mariadb.com/kb/en/library/string-functions/

Let’s work!

Do the exercises:

- P04_strings

4.22. Control Flow Functions (I).

CASE OPERATOR:

```
CASE value
    WHEN [compare_value] THEN
    result [WHEN [compare_value]
    THEN
    result ...] [ELSE result]
END

CASE
    WHEN [condition] THEN result
    [WHEN [condition] THEN result ...]
    [ELSE result]
END
```

```
SELECT CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END;
+-------------------------------------------------------------+
| CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END |
+-------------------------------------------------------------+
| one                                                         |
+-------------------------------------------------------------+

SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
+--------------------------------------------+
| CASE WHEN 1>0 THEN 'true' ELSE 'false' END |
+--------------------------------------------+
| true                                       |
+--------------------------------------------+


SELECT CASE BINARY 'B' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
+-----------------------------------------------------+
| CASE BINARY 'B' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END |
+-----------------------------------------------------+
|                                                NULL |
+-----------------------------------------------------+
```

4.22. Control Flow Functions.

IF Function:

IF(expr1,expr2,expr3)

```
SELECT IF(1>2,2,3);
+-------------+
| IF(1>2,2,3) |
+-------------+
|           3 |
+-------------+
```

```
SELECT IF(1<2,'yes','no');
+--------------------+
| IF(1<2,'yes','no') |
+--------------------+
| yes                |
+--------------------+
```

```
SELECT IF(STRCMP('test','test1'),'no','yes');
+----------------------------------------+
| IF(STRCMP('test','test1'),'no','yes') |
+----------------------------------------+
| no                                     |
+----------------------------------------+
```

IFNULL (or COALESCE):

IFNULL(expr1,expr2)

```
SELECT IFNULL(1/0,'yes');
+--------------------+
| IFNULL(1/0,'yes') |
+--------------------+
| yes                |
+--------------------+
```

```
SELECT IFNULL(1,0);
+-------------+
| IFNULL(1,0) |
+-------------+
|           1 |
+-------------+
```

NULLIF:

NULLIF(expr1,expr2)

```
SELECT NULLIF(1,1);
+-------------+
| NULLIF(1,1) |
+-------------+
|        NULL |
+-------------+
```

```
SELECT NULLIF(1,2);
+-------------+
| NULLIF(1,2) |
+-------------+
|           1 |
+-------------+
```

4.23. User-Defined Types (I).

- **create type** construct in SQL creates user-defined type
  - Example: **create type** Dollars as **numeric** (12,2)
    create table departmenta (
    dept_name varchar (20),
    building varchar (15),
    budget Dollars);
  - Not supported by MariaDB/MySQL, but supported by ORACLE.

4.24. Domains (I).

- **create domain** construct in SQL-92 creates user-defined domain types:
  - create domain person_name char(20) not null
- **Types** and **domains** are similar. Domains can have **constraints**, such as **not null**, specified on them.
  - create domain degree_level varchar(10)
    constraint degree_level_test
    check (value in (’Bachelors’, ’Masters’, ’Doctorate’));
- With create domain you can create a subtype that is based of one existing type (adding constraints to it).
- With create type you can create composite types or enum or others that they are structurally different to existing types.
- Not supported by MariaDB/MySQL, but supported by ORACLE.

4.25. Large-Object Types (I).

- Large objects (photos, videos, CAD files, etc.) are stored as a large object:
  - **blob**: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)
  - **clob**: character large object -- object is a large collection of character data
  - When a query returns a large object, a pointer is returned rather than the large object itself.

4.26. Authorization.

- With any **multiuser computer system**, **security** is a particularly **important issue** to address.
- Without adequate **security controls**, malicious users might invade our database, view confidential information, and make unauthorized changes to database information.
- We will see:
  - User management and authentication.
  - Privilege management and roles.

4.26. Authorization.

- To understand what we are going to do connect to MariaDB from localhost and from HeidiSQL (connecting localhost VS IP address).
- VERY IMPORTANT THE FOLLOWING COMMANDS MUST BE EXECUTED WITH ROOT PRIVILEGES.
- In MariaDB you can be root doing:
  - $> su
  - $> mysql

```
MariaDB [(none)]> SHOW GRANTS FOR 'alumne'@'%';
+-----------------------------------------------------------------------------------------------------------------+
| Grants for alumne@%                                                                                             |
+-----------------------------------------------------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'alumne'@'%' IDENTIFIED BY PASSWORD '*FEBAFAABF3035BF32237F30A0B3D5CF3B6BA7E4C' |
+-----------------------------------------------------------------------------------------------------------------+
1 row in set (0.000 sec)
```

4.26. Authorization: CREATE USER.

- The **CREATE USER** statement in MySQL allows us to create new MySQL/MariaDB accounts (=a database account that allows the user to log into the MySQL/MariaDB DBMS).
- Syntax:
  - CREATE USER user_account IDENTIFIED BY password;

4.26. Authorization: CREATE USER.

- **Example 1:**
  - To create a new user “user1” to connect to MySQL/MariaDB from the **localhost** with the password “alu01”:
  - CREATE USER 'user1'@'localhost' IDENTIFIED BY 'alu01';
- **Example 2:**
  - To create a new user “user1” to connect to MySQL/MariaDB from from **any host** with the password “alu02”:
  - CREATE USER 'user1'@'%' IDENTIFIED BY 'alu01';

4.26. Authorization: CREATE USER.

- **Example 3:**
  - To create two new users “user2” and “user3” **with a single statement** to connect to MySQL/MariaDB from the localhost with the password “alu02” and “alu03”:
  - CREATE USER

    'user2'@'localhost' IDENTIFIED BY 'alu02',

    'user3'@'localhost' IDENTIFIED BY 'alu03';

4.26. Authorization: SHOW GRANTS.

- The “SHOW GRANTS” statement is used to view the permissions of a user account.
- The syntax for the SHOW GRANTS is:
  - SHOW GRANTS FOR user-account;
- Example:
  - SHOW GRANTS FOR alumne@'%';

```
MariaDB [(none)]> SHOW GRANTS FOR alumne@'%';
+-----------------------------------------------------------------------------------------------------------------+
| Grants for alumne@%                                                                                             |
+-----------------------------------------------------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'alumne'@'%' IDENTIFIED BY PASSWORD '*FEBAFAABF3035BF32237F30A0B3D5CF3B6BA7E4C' |
+-----------------------------------------------------------------------------------------------------------------+
1 row in set (0.000 sec)
```

4.26. Authorization: GRANT/REVOKE privileges.

- We have already seen how to create user using the create user statement, but that statement **only creates a new user but does not grant any privileges to the user account**.
- To grant privileges to a user account, the GRANT statement is used.
- Syntax:
  - GRANT privileges_names ON object TO user_list;

4.26. Authorization: GRANT/REVOKE privileges.

- Example:
  - **Before** granting “UNIT04” database to “user2”:

```
root@DBVMPC:/home/alumne# mysql -u user2 -h localhost -palu02 UNIT04;
ERROR 1044 (42000): Access denied for user 'user2'@'localhost' to database 'UNIT04'
```

  - Execute as root user:
    - GRANT SELECT ON UNIT04.* TO user2@'localhost';
  - **After** granting “UNIT04” database to “user2”:

```
root@DBVMPC:/home/alumne# mysql -u user2 -h localhost -palu02 UNIT04;
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 268
Server version: 10.3.15-MariaDB-1 Debian 10

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [UNIT04]>
```

4.26. Authorization: GRANT/REVOKE privileges.

Privilege Name	Description
SELECT	Execute select statement on tables.
INSERT	Execute insert statement on tables.
UPDATE	Execute update statement on tables.
DELETE	Execute delete statement on tables.
INDEX	Create an INDEX on an existing table.
CREATE	Execute CREATE TABLE statements.
ALTER	Perform ALTER TABLE on table.
DROP	Execute DROP TABLE statements.
GRANT OPTION	Granted privileges to other users (the user must be granted to the same privileges).
ALL	grants all permissions except GRANT OPTION.

4.26. Authorization: GRANT/REVOKE privileges.

- **Example 1:**
  - Granting SELECT privilege to “user3” in the table “R1” inside the database “UNIT04” only from localhost:
  - GRANT SELECT ON UNIT04.R1 TO 'user3'@'localhost';
- **Example 2:**
  - Granting SELECT, INSERT, UPDATE, and DELETE privileges to “user3” in the table “R1” inside the database “UNIT04” only from localhost:
  - GRANT SELECT, INSERT, UPDATE, DELETE ON UNIT04.R1 TO 'user3'@'localhost';

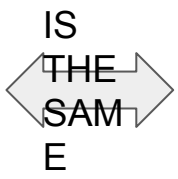
4.26. Authorization: GRANT/REVOKE privileges.

- **Example 3:**
  - Granting **all** the privileges to user3 in table UNIT04.R1:
  - GRANT ALL ON UNIT04.R1 TO 'user3'@'localhost';
- **Example 4:**
  - Granting **all** the privileges to user3 in table UNIT04.R1 with grant option:
  - GRANT ALL ON UNIT04.R1 TO 'user3'@'localhost' WITH GRANT OPTION;

4.26. Authorization: GRANT/REVOKE privileges.

- **Example 5:**
  - If you use IDENTIFIED BY in a GRANT statement you also create the user:

GRANT ALL PRIVILEGES ON *.* TO 'alumne'@'192.168.56.%' IDENTIFIED BY 'alualualu' WITH GRANT OPTION;



CREATE USER 'alumne'@'192.168.56.%' IDENTIFIED BY 'alualualu';

GRANT ALL PRIVILEGES ON *.* TO 'alumne'@'192.168.56.%' WITH GRANT OPTION;

4.26. Authorization: GRANT/REVOKE privileges.

- **Example 6:**
  - We will see Functions/Procedures later, but there is a execute privilege for them:
  - GRANT EXECUTE ON [PROCEDURE | FUNCTION] object TO user;

4.26. Authorization: GRANT/REVOKE privileges.

- **Revoking privileges from a table:**
  - The Revoke statement is used to revoke some or all of the privileges which have been granted to a user.
  - Syntax:
    - REVOKE privileges ON object FROM user_list;

4.26. Authorization: GRANT/REVOKE privileges.

- **Example 1:**
  - Revoking SELECT privilege to “user3” in the table “R1” inside the database “UNIT04” only from localhost:
  - REVOKE SELECT ON UNIT04.R1 FROM 'user3'@'localhost';
- **Example 2:**
  - Revoking SELECT, INSERT, UPDATE, and DELETE privileges to “user3” in the table “R1” inside the database “UNIT04” only from localhost:
  - REVOKE SELECT, INSERT, UPDATE, DELETE ON UNIT04.R1 FROM 'user3'@'localhost';

4.26. Authorization: GRANT/REVOKE privileges.

- **Example 3:**
  - Revoking **all** the privileges to user3 in table UNIT04.R1:
  - REVOKE ALL ON UNIT04.R1 FROM 'user3'@'localhost';
- **Example 4:**
  - Revoking only the **GRANT OPTION** to user3 in table UNIT04.R1 with grant option:
  - REVOKE GRANT OPTION ON UNIT04.R1 FROM 'user3'@'localhost';

4.26. Authorization.

- **GRANT reference:**
  - https://mariadb.com/kb/en/grant/
- **REVOKE reference:**
  - https://mariadb.com/kb/en/revoke/

4.26. Authorization: Roles.

- MariaDB reference: “A **role** bundles a number of privileges together. It assists larger organizations where, typically, a number of users would have the same privileges, and, previously, the only way to change the privileges for a group of users was by changing each user's privileges individually”.
- Conceptually, you can compare a role with a group in Active Directory.

4.26. Authorization: Roles.

- **Syntax:**
  - create role role_name [IDENTIFIED BY password];
- Example 1:
  - Create a role “professor” and grant it to user “sgonzalez”.
  - **create role** professor [IDENTIFIED BY *password*];
  - **grant** professor **to** sgonzalez;

4.26. Authorization: Roles.

- Privileges can be granted to roles:
  - **grant select** on EMPLOYEES to professor;
- Roles can be granted to users, as well as to other roles
  - create role assistant_professor
  - grant assistant_professor to professor;
    - Professor inherits all privileges of assistant_professor
- Chain of roles:
  - **create role** head_department;
  - **grant** professor **to** head_department;
  - **grant** head_department **to** mcabot;

4.26. Authorization: Roles.

- **SET ROLE:**
  - Even though the user was granted the professor role. He/she needs to set the role first: SET ROLE professor;
  - Check the example here: https://mariadb.com/kb/en/library/roles_overview/
  - Anyway, users maybe have a default group (we'll see this later).

4.26. Authorization: Roles.

- **WITH ADMIN**
  - The optional WITH ADMIN clause determines whether the current user, the current role or another user or role has use of the newly created role. If the clause is omitted, WITH ADMIN CURRENT_USER is treated as the default, which means that the current user will be able to GRANT this role to users.
  - CREATE ROLE journalist;
    VS
  - CREATE ROLE developer WITH ADMIN lorinda;

4.26. Authorization: Roles.

- **Roles, setting a user's default role:**
  - ALTER USER user
    [DEFAULT ROLE
    {role [, role] ...
    |ALL [EXCEPT role [, role] ...]
    | NONE }} }
  - ALTER USER sgonzalez DEFAULT ROLE professor;

4.26. Authorization.

- **Roles reference:**
  - https://mariadb.com/kb/en/roles/

4.26. Authorization: Views.

- **Authorization on Views:**
  - create view V_DEPARTMENTS AS
    select D.num, D.name, D.town_code, T.name as town_name, count(E.num) as num_employees
    from DEPARTMENTS D
    left outer join EMPLOYEES E on E.dept_num = D.num
    left outer join TOWNS T on T.code= D.town_code
    group by D.num, D.name, D.town_code, T.name;
  - grant select on V_DEPARTMENTS to ceo_staff
  - Suppose that a ceo_staff member issues
    - select *
      from V_DEPARTMENTS;
  - What if
    - ceo_staff does not have permissions on EMPLOYEES, DEPARTMENTS and/or TOWNS?
    - creator of view did not have some permissions on EMPLOYEES, DEPARTMENTS and/or TOWNS?

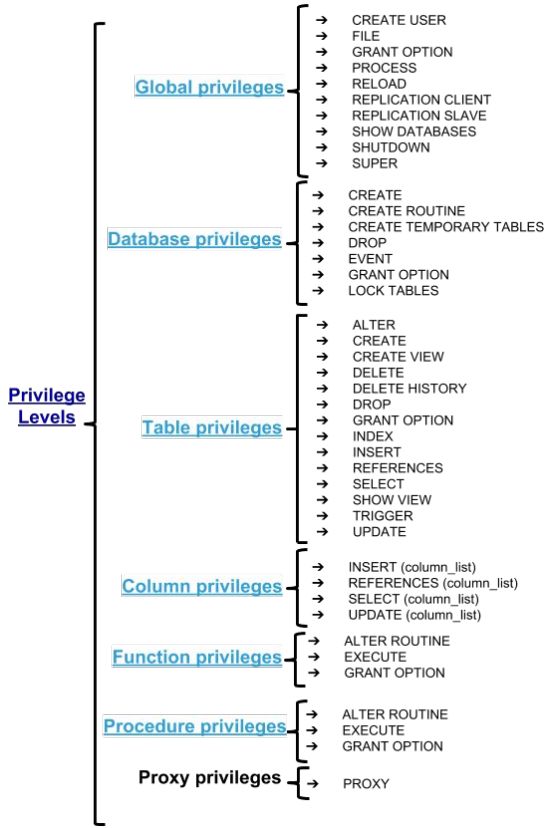
4.26. Authorization: Privilege Management.

- There are many things that we have NOT seen…
- Check the links to MariaDB reference...

4.26. Authorization: Privilege Management.

GRANT Syntax for MariaDB

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user_specification [ user_options ...]

user_specification:
  username [authentication_option]

authentication_option:
  IDENTIFIED BY 'password'
  | IDENTIFIED BY PASSWORD 'password_hash'
  | IDENTIFIED {VIA|WITH} authentication_plugin
  | IDENTIFIED {VIA|WITH} authentication_plugin {USING|AS} 'authentication_string'
  | IDENTIFIED {VIA|WITH} authentication_plugin {USING|AS} PASSWORD('password')

GRANT PROXY ON username
    TO username [, username] ...
    [WITH GRANT OPTION]

user_options:
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH with_option [with_option] ...]

object_type:
    TABLE
  | FUNCTION
  | PROCEDURE

priv_level:
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name

with_option:
    GRANT OPTION
  | resource_option

resource_option:
  MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
  | MAX_STATEMENT_TIME time

tls_option:
  SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'
```



Let’s work!

Do the exercises:

- P04_auth (for course 2019/2020 do only from 1 to 10).

4.27. Indexes (I).

- Install this database and this one. After that run this query in both of them.
- The results depend on the computer:

```
MariaDB [TENNIS2]> SELECT CONCAT_WS(' ', P1.p_name, P1.p_surname) AS Player1,
    -> CONCAT_WS(' ', P2.p_name, P2.p_surname) AS Player2, t_name AS Tournament, t_end_date AS `End date`, IF (mr_w
er:', P1.p_name, P1.p_surname), CONCAT_WS(' ', 'Winner:', P2.p_name, P2.p_surname)) AS Winner
    -> FROM REGISTRATIONS R1, REGISTRATIONS R2, MATCHES, MATCH_RESULTS, TOURNAMENTS, PLAYERS P1, PLAYERS P2
    -> WHERE
    -> m_id = mr_m_id AND
    ->   ((R1.r_num = m_r_num1 AND R2.r_num = m_r_num2) OR
    ->     (R1.r_num = m_r_num2 AND R2.r_num = m_r_num1) ) AND
    ->         t_id = m_t_id AND
    ->         t_num_rounds = m_round AND
    ->         t_type = 'Singles' AND
    ->         R2.r_p_id = P2.p_id AND
    ->         R1.r_p_id = P1.p_id AND
    ->         P1.p_id < P2.p_id;
+-----------------+-----------------------+-----------------+------------+---------------------------+
| Player1         | Player2               | Tournament      | End date   | Winner                    |
+-----------------+-----------------------+-----------------+------------+---------------------------+
| Roger Federer   | Rafael Nadal          | French Open     | 2007-06-10 | Winner: Rafael Nadal      |
| Roger Federer   | Rafael Nadal          | Wimbledon       | 2007-07-08 | Winner: Roger Federer     |
| Roger Federer   | Novak Djokovic        | US Open         | 2007-09-09 | Winner: Roger Federer     |
| Mikhail Youzhny | Philipp Kohlschreiber | BMW Open        | 2007-05-06 | Winner: Philipp Kohlschreiber |
| Tommy Robredo   | David Ferrer          | Heineken Open   | 2007-01-14 | Winner: David Ferrer      |
| Juan Carlos Ferrero | Guillermo Canas   | Brasil Open 2007 | 2007-02-18 | Winner: Guillermo Canas   |
| James Blake     | Radek Stepanek        | Countrywide Classic | 2007-07-22 | Winner: Radek Stepanek |
| Roger Federer   | Fernando Gonzalez     | Australian Open | 2007-01-28 | Winner: Roger Federer     |
+-----------------+-----------------------+-----------------+------------+---------------------------+
8 rows in set (9 min 9.435 sec)
```

```
MariaDB [TENNIS]> SELECT CONCAT_WS(' ', P1.p_name, P1.p_surname) AS Player1,
    -> CONCAT_WS(' ', P2.p_name, P2.p_surname) AS Player2, t_name AS Tournament, t_end_date AS `End date`, IF (mr_w
er:', P1.p_name, P1.p_surname), CONCAT_WS(' ', 'Winner:', P2.p_name, P2.p_surname)) AS Winner
    -> FROM REGISTRATIONS R1, REGISTRATIONS R2, MATCHES, MATCH_RESULTS, TOURNAMENTS, PLAYERS P1, PLAYERS P2
    -> WHERE
    -> m_id = mr_m_id AND
    ->   ((R1.r_num = m_r_num1 AND R2.r_num = m_r_num2) OR
    ->     (R1.r_num = m_r_num2 AND R2.r_num = m_r_num1) ) AND
    ->         t_id = m_t_id AND
    ->         t_num_rounds = m_round AND
    ->         t_type = 'Singles' AND
    ->         R2.r_p_id = P2.p_id AND
    ->         R1.r_p_id = P1.p_id AND
    ->         P1.p_id < P2.p_id;
+-----------------+-----------------------+-----------------+------------+---------------------------+
| Player1         | Player2               | Tournament      | End date   | Winner                    |
+-----------------+-----------------------+-----------------+------------+---------------------------+
| Roger Federer   | Rafael Nadal          | French Open     | 2007-06-10 | Winner: Rafael Nadal      |
| Roger Federer   | Rafael Nadal          | Wimbledon       | 2007-07-08 | Winner: Roger Federer     |
| Roger Federer   | Fernando Gonzalez     | Australian Open | 2007-01-28 | Winner: Roger Federer     |
| Roger Federer   | Novak Djokovic        | US Open         | 2007-09-09 | Winner: Roger Federer     |
| Tommy Robredo   | David Ferrer          | Heineken Open   | 2007-01-14 | Winner: David Ferrer      |
| James Blake     | Radek Stepanek        | Countrywide Classic | 2007-07-22 | Winner: Radek Stepanek |
| Mikhail Youzhny | Philipp Kohlschreiber | BMW Open        | 2007-05-06 | Winner: Philipp Kohlschreiber |
| Juan Carlos Ferrero | Guillermo Canas   | Brasil Open 2007 | 2007-02-18 | Winner: Guillermo Canas   |
+-----------------+-----------------------+-----------------+------------+---------------------------+
8 rows in set (0.038 sec)
```

9 minutes 9.435 seconds VS 0.038 seconds !

4.27. Indexes (II).

- There are four main kinds of indexes:
  - **primary keys (unique and not null). PRIMARY KEY.**
  - **unique indexes (unique and can be null). UNIQUE.**
  - **plain indexes (not necessarily unique). INDEX.**
  - **full-text indexes (for full-text searching)**
- Index implementation depends on the DBMS.
- Foreign keys are also indexed.

4.27. Indexes (III).

PRIMARY KEY INDEX

```
CREATE TABLE `Employees` (
  `ID` TINYINT(3) UNSIGNED NOT NULL AUTO_INCREMENT,
  `First_Name` VARCHAR(25) NOT NULL,
  `Last_Name` VARCHAR(25) NOT NULL,
  `Position` VARCHAR(25) NOT NULL,
  `Home_Address` VARCHAR(50) NOT NULL,
  `Home_Phone` VARCHAR(12) NOT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=Aria;
```

Aria Storage Engine

UNIQUE INDEX

```
CREATE TABLE `Employees` (
  `ID` TINYINT(3) UNSIGNED NOT NULL,
  `First_Name` VARCHAR(25) NOT NULL,
  `Last_Name` VARCHAR(25) NOT NULL,
  `Position` VARCHAR(25) NOT NULL,
  `Home_Address` VARCHAR(50) NOT NULL,
  `Home_Phone` VARCHAR(12) NOT NULL,
  `Employee_Code` VARCHAR(25) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY (`Employee_Code`)
) ENGINE=Aria;
```

UNIQUE INDEX

```
CREATE TABLE t1 (a INT NOT NULL, b INT, UNIQUE (a,b));

INSERT INTO t1 values (1,1), (2,2);

SELECT * FROM t1;
+---+------+
| a | b    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
```

PLAIN INDEX

```
CREATE TABLE t2 (a INT NOT NULL, b INT, INDEX (a,b));

INSERT INTO t2 values (1,1), (2,2), (2,2);

SELECT * FROM t2;
+---+------+
| a | b    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 2 |    2 |
+---+------+
```

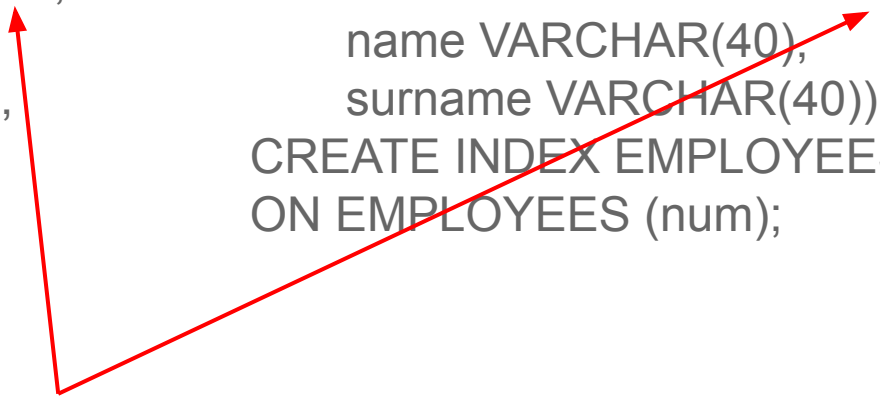
FULL-TEXT INDEX

```
CREATE TABLE ft_myisam(copy TEXT,FULLTEXT(copy)) ENGINE=MyISAM;

INSERT INTO ft_myisam(copy) VALUES ('Once upon a time'),
  ('There was a wicked witch'), ('Who ate everybody up');

SELECT * FROM ft_myisam WHERE MATCH(copy) AGAINST('wicked');
+--------------------------+
| copy                     |
+--------------------------+
| There was a wicked witch |
+--------------------------+
```

4.27. Indexes (IV).

- CREATE TABLE EMPLOYEES (
  num INTEGER NOT NULL,
  name VARCHAR(40),
  surname VARCHAR(40),
  PRIMARY KEY (num));

- CREATE TABLE EMPLOYEES (
  num INTEGER NOT NULL,
  name VARCHAR(40),
  surname VARCHAR(40));
  CREATE INDEX EMPLOYEES_num_i
  ON EMPLOYEES (num);



num must be defined as NOT NULL, otherwise the index could not have been created.

4.27. Indexes (V).

- **Choosing Indexes**:
  - In general, you should only **add indexes to match** the queries your application uses (**JOINs**!). Any extra will waste resources. You saw what happened with the query in the first slide…
  - Using the EXPLAIN statement on your queries can help you decide which columns need indexing.

```
MariaDB [TENNIS]> EXPLAIN SELECT CONCAT_WS(' ', P1.p_name, P1.p_surname) AS Player1,  CONCAT_WS(' ', P2.p_name, P2.p_surname) AS Player2, t_name AS Tournament, t_end_date AS `End date`, IF (mr_winner = R1.r_num,
CONCAT_WS(' ', 'Winner:', P1.p_name, P1.p_surname), CONCAT_WS(' ', 'Winner:', P2.p_name, P2.p_surname)) AS Winner FROM REGISTRATIONS R1, REGISTRATIONS R2, MATCHES, MATCH_RESULTS, TOURNAMENTS, PLAYERS P1, PLAYERS
P2 WHERE m_id = mr_m_id AND   ((R1.r_num = m_r_num1 AND R2.r_num = m_r_num2) OR     (R1.r_num = m_r_num2 AND R2.r_num = m_r_num1) ) AND         t_id = m_t_id AND         t_num_rounds = m_round AND         t_type
= 'Singles' AND         R2.r_p_id = P2.p_id AND         R1.r_p_id = P1.p_id AND         P1.p_id < P2.p_id;
+------+-------------+---------------+--------+-------------------------------+---------+---------+-------------------------+------+-------------------------------------------------+
| id   | select_type | table         | type   | possible_keys                 | key     | key_len | ref                     | rows | Extra                                           |
+------+-------------+---------------+--------+-------------------------------+---------+---------+-------------------------+------+-------------------------------------------------+
|    1 | SIMPLE      | TOURNAMENTS   | ALL    | PRIMARY                       | NULL    | NULL    | NULL                    |   16 | Using where                                     |
|    1 | SIMPLE      | MATCHES       | ref    | PRIMARY,m_t_id,m_r_num1,m_r_num2 | m_t_id  | 5       | TENNIS.TOURNAMENTS.t_id |    1 | Using where                                     |
|    1 | SIMPLE      | MATCH_RESULTS | eq_ref | PRIMARY                       | PRIMARY | 4       | TENNIS.MATCHES.m_id     |    1 |                                                 |
|    1 | SIMPLE      | P1            | ALL    | PRIMARY                       | NULL    | NULL    | NULL                    |  309 | Using join buffer (flat, BNL join)              |
|    1 | SIMPLE      | R1            | ref    | PRIMARY,r_p_id                | r_p_id  | 4       | TENNIS.P1.p_id          |    2 | Using where; Using index                        |
|    1 | SIMPLE      | P2            | ALL    | PRIMARY                       | NULL    | NULL    | NULL                    |  309 | Using where; Using join buffer (flat, BNL join) |
|    1 | SIMPLE      | R2            | ref    | PRIMARY,r_p_id                | r_p_id  | 4       | TENNIS.P2.p_id          |    2 | Using where; Using index                        |
+------+-------------+---------------+--------+-------------------------------+---------+---------+-------------------------+------+-------------------------------------------------+
7 rows in set (0.007 sec)
```

4.27. Indexes (VI).

- **Choosing Indexes**:
  - Another example with EXPLAIN:

```
MariaDB [TENNIS]> EXPLAIN SELECT c_name AS CountryName, count(p_id) AS NumberOfPlayers
    -> FROM COUNTRIES LEFT OUTER JOIN  PLAYERS ON c_id = p_c_id
    -> GROUP BY c_name
    -> ORDER BY c_name;
+------+-------------+-----------+------+---------------+--------+---------+----------------------+------+---------------------------------+
| id   | select_type | table     | type | possible_keys | key    | key_len | ref                  | rows | Extra                           |
+------+-------------+-----------+------+---------------+--------+---------+----------------------+------+---------------------------------+
|    1 | SIMPLE      | COUNTRIES | ALL  | NULL          | NULL   | NULL    | NULL                 |  213 | Using temporary; Using filesort |
|    1 | SIMPLE      | PLAYERS   | ref  | p_c_id        | p_c_id | 15      | TENNIS.COUNTRIES.c_id |    3 | Using index                     |
+------+-------------+-----------+------+---------------+--------+---------+----------------------+------+---------------------------------+
2 rows in set (0.003 sec)
```

4.27. Indexes (VII).

- **Choosing Indexes**:
  - We said that if a query contains something like LIKE '%word%', without a **fulltext index** you are using a full table scan every time, which is very slow.
  - If you are building a large table then for best performance **add the index after the table is populated with data**. This is to increase the insert performance and remove the index overhead during inserts.

4.27. Indexes (VIII).

- **Viewing Indexes**:
  - You can view which indexes are present on a table, as well as details about them, with the SHOW INDEX statement.
  - If you want to know how to re-create an index, run SHOW CREATE TABLE.

```
MariaDB [TENNIS]> SELECT CONCAT_WS(' ', P1.p_name, P1.p_surname) AS Player1,
    -> CONCAT_WS(' ', P2.p_name, P2.p_surname) AS Player2, t_name AS Tournament, t_end_date AS `End date`, IF (mr_w
er:', P1.p_name, P1.p_surname), CONCAT_WS(' ', 'Winner:', P2.p_name, P2.p_surname)) AS Winner
    -> FROM REGISTRATIONS R1, REGISTRATIONS R2, MATCHES, MATCH_RESULTS, TOURNAMENTS, PLAYERS P1, PLAYERS P2
    -> WHERE
    -> m_id = mr_m_id AND
    ->   ((R1.r_num = m_r_num1 AND R2.r_num = m_r_num2) OR
    ->     (R1.r_num = m_r_num2 AND R2.r_num = m_r_num1) ) AND
    ->         t_id = m_t_id AND
    ->         t_num_rounds = m_round AND
    ->         t_type = 'Singles' AND
    ->         R2.r_p_id = P2.p_id AND
    ->         R1.r_p_id = P1.p_id AND
    ->         P1.p_id < P2.p_id;
+--------------------+--------------------+-----------------+------------+-----------------------------+
| Player1            | Player2            | Tournament      | End date   | Winner                      |
+--------------------+--------------------+-----------------+------------+-----------------------------+
| Roger Federer      | Rafael Nadal       | French Open     | 2007-06-10 | Winner: Rafael Nadal        |
| Roger Federer      | Rafael Nadal       | Wimbledon       | 2007-07-08 | Winner: Roger Federer       |
| Roger Federer      | Fernando Gonzalez  | Australian Open | 2007-01-28 | Winner: Roger Federer       |
| Roger Federer      | Novak Djokovic     | US Open         | 2007-09-09 | Winner: Roger Federer       |
| Tommy Robredo      | David Ferrer       | Heineken Open   | 2007-01-14 | Winner: David Ferrer        |
| James Blake        | Radek Stepanek     | Countrywide Classic | 2007-07-22 | Winner: Radek Stepanek      |
| Mikhail Youzhny    | Philipp Kohlschreiber | BMW Open        | 2007-05-06 | Winner: Philipp Kohlschreiber |
| Juan Carlos Ferrero | Guillermo Canas    | Brasil Open 2007 | 2007-02-18 | Winner: Guillermo Canas     |
+--------------------+--------------------+-----------------+------------+-----------------------------+
8 rows in set (0.038 sec)
```

```
MariaDB [TENNIS]> show index from PLAYERS;
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table   | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| PLAYERS |          0 | PRIMARY  |            1 | p_id        | A         |         309 |     NULL | NULL   |      | BTREE      |         |               |
| PLAYERS |          1 | p_c_id   |            1 | p_c_id      | A         |         103 |     NULL | NULL   | YES  | BTREE      |         |               |
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
2 rows in set (0.001 sec)
```

4.27. Indexes (IX).

- **When to Create/Remove an Index**:
  - If an index is rarely used (or not used at all) then remove it to increase INSERT, and UPDATE performance.
  - If user statistics are enabled, the Information Schema INDEX_STATISTICS table stores the index usage.
  - If the slow query log is enabled and the log_queries_not_using_indexes server system variable is ON, the queries which do not use indexes are logged.
- Those concepts are for an advanced subject about Databases...

4.27. Indexes (X).

- MariaDB references: https://mariadb.com/kb/en/optimization-and-indexes/

4.27. Indexes (XI).

- **A JOIN WITHOUT INDEXES IN BOTH COLUMNS IS VERY SLOW!! Thus, define PRIMARY KEYS and FOREIGN KEYS!**

Let’s work!

More queries:

- P04_BORJA_PHONE_queries
- P04_TENNIS

Sources.

- **M. J. Ramos, A. Ramos and F. Montero.** Sistemas gestores de bases de datos. McGrawHill: 1th Edition, 2006.
- **Abraham Silberschatz, Henry F. Korth and S. Sudarshan**. *Database System Concepts (Chapter 4)*. McGrawHill: 6th Edition, 2010.<http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html>
- https://en.wikipedia.org/wiki/Join_(SQL)
- http://dbtech.uom.gr/mod/resource/view.php?id=817
- Apunts de la UIB del professor Miquel Manresa (1996).
- https://www.studytonight.com/dbms/
- http://www.informit.com/articles/article.aspx?p=174375&seqNum=2
- http://www.mysqltutorial.org/mysql-left-join.aspx
- https://mariadb.com/kb/en/library/getting-started-with-indexes/