Unit 7: Intermediate SQL.

2022/2023

Contents

Sample database

```
$ sudo su
$ apt get update
$ apt get install postgresql postgresql-client
$ cp MiniSoccerLeague_postgresql.sql /var/lib/postgresql/
$ su - postgres
$ cd
$ psql
\i MiniSoccerLeague_postgresql.sql
```

PostgreSQL

```
$ sudo apt update
$ sudo apt install mariadb-server mariadb-client
$ sudo mariadb < MiniSoccerLeague_mariadb.sql
```

MariaDB

Sample database

```
MariaDB [minisoccerleague]> select * from teams;
+-----+-------------------+------+------+
| id  | name              | year | town |
+-----+-------------------+------+------+
| AMG | C. D. Amigos      | 1920 | NULL |
| ARS | Arsenal           | 1886 | LO   |
| CHE | Chelsea           | 1905 | LO   |
| LIV | Liverpool         | 1892 | LI   |
| MAC | Manchester City   | 1880 | MA   |
| MAU | Manchester United | 1878 | MA   |
| RMA | R. C. D. Mallorca | 1916 | PM   |
| TOT | Tottenham Hotspur | 1882 | LO   |
+-----+-------------------+------+------+
8 rows in set (0,000 sec)
```

```
MariaDB [minisoccerleague]> select * from towns;
+----+-------------------+---------+
| id | name              | country |
+----+-------------------+---------+
| AT | Athens            | NULL    |
| LI | Liverpool         | UK      |
| LO | London            | UK      |
| MA | Manchester        | UK      |
| PM | Palma de Mallorca | ES      |
+----+-------------------+---------+
5 rows in set (0,000 sec)
```

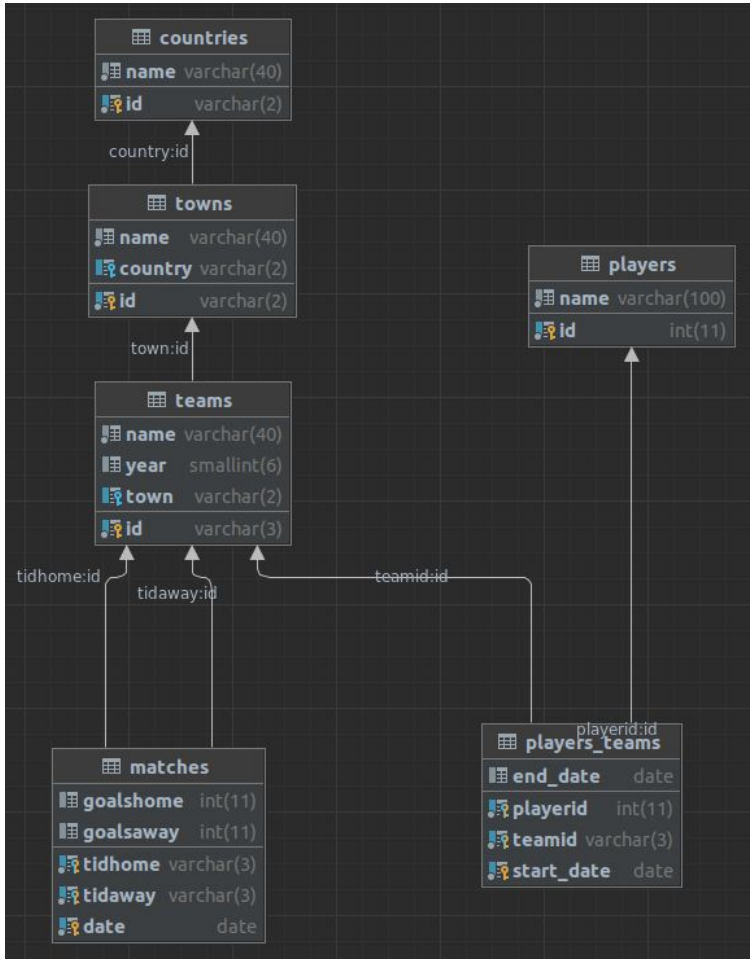
```
MariaDB [minisoccerleague]> select * from countries;
+----+----------------+
| id | name           |
+----+----------------+
| ES | Spain          |
| IT | Italy          |
| UK | United Kingdom |
+----+----------------+
3 rows in set (0,000 sec)
```

```
MariaDB [minisoccerleague]> select * from matches;
+---------+---------+-----------+-----------+------------+
| tidhome | tidaway | goalshome | goalsaway | date       |
+---------+---------+-----------+-----------+------------+
| ARS     | CHE     |         0 |         0 | 2017-01-21 |
| ARS     | CHE     |         2 |         0 | 2017-12-03 |
| ARS     | LIV     |         1 |         3 | 2016-11-20 |
| ARS     | LIV     |         0 |         0 | 2018-01-17 |
                    ■ ■ ■
| TOT     | MAC     |         6 |         0 | 2017-11-25 |
| TOT     | MAU     |         2 |         2 | 2016-11-28 |
| TOT     | MAU     |         1 |         0 | 2018-01-27 |
+---------+---------+-----------+-----------+------------+
60 rows in set (0,000 sec)
```

```
minisoccerleague=# select * from players;
 id |  name
----+---------
  1 | Ronaldo
  2 | Rooney
  3 | Mbappe
(3 rows)
```

```
minisoccerleague=# select * from players_teams;
 playerid | teamid | start_date |  end_date
----------+--------+------------+------------
        1 | MAU    | 2003-08-12 | 2007-08-31
        1 | MAU    | 2021-09-01 | 2022-11-22
        2 | MAU    | 2004-09-01 | 2017-11-22
(3 rows)
```

Sample database



Generated with DataGrip (license for students/teachers).

7.1. Types of Joined Relations.

6.7. Select clause examples.

Find the names, surname and department name of all the employees:

```
select employees.name, employees.surname, departments.name
from employees, departments
where employees.dept_num = departments.num
order by departments.name, employees.surname;
```

This is a join!!

Find the names, surname and department name of all the employees who earn more than $2500:

```
select E.name, E.surname, E.salary, D.name
from employees as E, departments as D
where E.dept_num = D.num and
      E.salary > 2500
order by E.salary desc;
```

```
MariaDB [samplecompany]> select E.name, E.surname, E.salary, D.name
    -> from employees as E, departments as D
    -> where E.dept_num = D.num and
    ->       E.salary > 2500
    -> order by E.salary desc;
+------------+------------+--------+------------+
| name       | surname    | salary | name       |
+------------+------------+--------+------------+
| BARTOLOME  | GOMIS      |   3005 | SALES      |
| JESUS      | GILBERTO   |   3000 | RESEARCH   |
| ANA        | FERNÁNDEZ  |   3000 | RESEARCH   |
| JUAN       | JIMÉNEZ    |   2900 | RESEARCH   |
| MARIA      | CEREZO     |   2885 | ACCOUNTING |
| ANTONIO    | RUIZ       |   2885 | RESEARCH   |
| FERNANDA   | RUIZ       |   2885 | RESEARCH   |
+------------+------------+--------+------------+
7 rows in set (0,001 sec)
```

6.7. Select clause examples.

INNER JOIN

```
select employees.name, employees.surname, departments.name
from employees, departments
where employees.dept_num = departments.num
order by departments.name, employees.surname;
```

```
select employees.name, employees.surname, departments.name
from employees join departments
on employees.dept_num = departments.num
order by departments.name, employees.surname;
```

Are the same!!

```
select employees.name, employees.surname, departments.name
from employees inner join departments
on employees.dept_num = departments.num
order by departments.name, employees.surname;
```

7.1. Types of Joined Relations.

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 ↔ WHERE).

ANSI-standard SQL specifies five **types of JOIN**:

INNER | LEFT OUTER | FULL OUTER | RIGHT OUTER | CROSS

7.1. Types of Joined Relations.

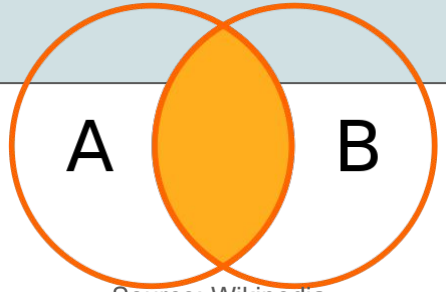
INNER JOIN

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 6** were **INNER JOINS**.



Source: Wikipedia.

7.1. Types of Joined Relations.

INNER JOIN

```
select * from towns, countries;
```

MariaDB

```
+----+--------------------+---------+----+----------------+
| id | name               | country | id | name           |
+----+--------------------+---------+----+----------------+
| AT | Athens             | NULL    | ES | Spain          |
| AT | Athens             | NULL    | IT | Italy          |
| AT | Athens             | NULL    | UK | United Kingdom |
| LI | Liverpool          | UK      | ES | Spain          |
| LI | Liverpool          | UK      | IT | Italy          |
| LI | Liverpool          | UK      | UK | United Kingdom |
| LO | London             | UK      | ES | Spain          |
| LO | London             | UK      | IT | Italy          |
| LO | London             | UK      | UK | United Kingdom |
| MA | Manchester         | UK      | ES | Spain          |
| MA | Manchester         | UK      | IT | Italy          |
| MA | Manchester         | UK      | UK | United Kingdom |
| PM | Palma de Mallorca  | ES      | ES | Spain          |
| PM | Palma de Mallorca  | ES      | IT | Italy          |
| PM | Palma de Mallorca  | ES      | UK | United Kingdom |
+----+--------------------+---------+----+----------------+
15 rows in set (0,001 sec)
```

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

Red color: **NULLs never satisfy the join-predicate**.
Athens is not here because it is not satisfied T.country=C.id

```
select * from towns T, countries C where T.country=C.id;
```

MariaDB

```
+----+--------------------+---------+----+----------------+
| id | name               | country | id | name           |
+----+--------------------+---------+----+----------------+
| LI | Liverpool          | UK      | UK | United Kingdom |
| LO | London             | UK      | UK | United Kingdom |
| MA | Manchester         | UK      | UK | United Kingdom |
| PM | Palma de Mallorca  | ES      | ES | Spain          |
+----+--------------------+---------+----+----------------+
4 rows in set (0,040 sec)
```

7.1. Types of Joined Relations.

INNER JOIN

Explicit INNER JOIN:

```
select *
from towns T
  INNER JOIN countries C ON T.country = C.id;
```

Implicit inner join:

```
select *
from towns T,
  countries C
where T.country = C.id;
```

One can further classify inner joins as:

- equi-joins
- natural joins
- cross-joins

7.1. Types of Joined Relations.

INNER JOIN equi-joins

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.

7.1. Types of Joined Relations.

INNER JOIN natural joins

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**.

A natural join assumes **stability and consistency in column names** which can change during vendor mandated version upgrades.

```
SELECT *
FROM TABLE1
 NATURAL JOIN TABLE2;
```

7.1. Types of Joined Relations.

INNER JOIN cross-joins

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.

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**.

Implicit cross-joins:

```
SELECT *
FROM TABLE1
 CROSS JOIN TABLE2;
```

Implicit cross-joins:

```
SELECT *
FROM TABLE1,
 TABLE2;
```

7.1. Types of Joined Relations.

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.

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).

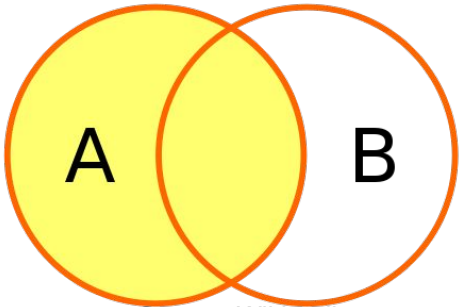
‘left’ and ‘right’ refer to the two sides of the JOIN keyword.

7.1. Types of Joined Relations.

LEFT OUTER JOIN

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.



Source: Wikipedia.

7.1. Types of Joined Relations.

LEFT OUTER JOIN

```
select *
from towns T
  inner join countries C on T.country = C.id;
```

```
+----+------------------+---------+----+----------------+
| id | name             | country | id | name           |
+----+------------------+---------+----+----------------+
| LI | Liverpool        | UK      | UK | United Kingdom |
| LO | London           | UK      | UK | United Kingdom |
| MA | Manchester       | UK      | UK | United Kingdom |
| PM | Palma de Mallorca | ES     | ES | Spain          |
+----+------------------+---------+----+----------------+
4 rows in set (0,000 sec)
```

Only the rows matching

Table on the left

Table on the right

```
select *
from towns T
  left outer join countries C on T.country = C.id;
```

```
+----+------------------+---------+------+----------------+
| id | name             | country | id   | name           |
+----+------------------+---------+------+----------------+
| AT | Athens           | NULL    | NULL | NULL           |
| LI | Liverpool        | UK      | UK   | United Kingdom |
| LO | London           | UK      | UK   | United Kingdom |
| MA | Manchester       | UK      | UK   | United Kingdom |
| PM | Palma de Mallorca | ES     | ES   | Spain          |
+----+------------------+---------+------+----------------+
5 rows in set (0,000 sec)
```

All values in the left table

7.1. Types of Joined Relations.

LEFT OUTER JOIN

Alternative syntaxes:

Oracle and **Informix** support the deprecated syntax:

```
select *
from towns T,
  countries C
where T.country = C.id (+);
```

Sybase supports the syntax (Microsoft SQL Server deprecated this syntax since version 2000):

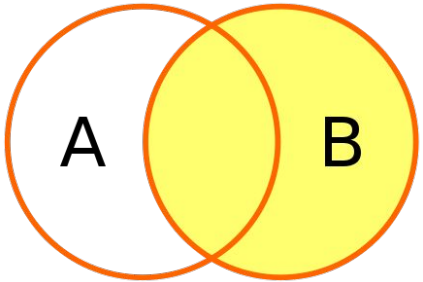
```
select *
from towns T,
  countries C
where T.country *= C.id;
```

7.1. Types of Joined Relations.

RIGHT OUTER JOIN

The result of a **right outer join** (or simply **right join**) for tables A and B **always contains all rows of the "right" table** (A), even if the join-condition does not find any matching row in the "left" table (B).

A right outer join returns **all the values from an inner join plus all values in the right table** that do not match the left table, including rows with NULL values in the link column.



Source: Wikipedia.

7.1. Types of Joined Relations.

RIGHT OUTER JOIN

```
select *
from towns T
  inner join countries C on T.country = C.id;
```

```
+----+-------------------+---------+----+----------------+
| id | name              | country | id | name           |
+----+-------------------+---------+----+----------------+
| LI | Liverpool         | UK      | UK | United Kingdom |
| LO | London            | UK      | UK | United Kingdom |
| MA | Manchester        | UK      | UK | United Kingdom |
| PM | Palma de Mallorca | ES      | ES | Spain          |
+----+-------------------+---------+----+----------------+
4 rows in set (0,000 sec)
```

Only the rows matching

Table on the left

Table on the right

```
select *
from towns T
  right outer join countries C on T.country = C.id;
```

```
+------+-------------------+---------+----+----------------+
| id   | name              | country | id | name           |
+------+-------------------+---------+----+----------------+
| LI   | Liverpool         | UK      | UK | United Kingdom |
| LO   | London            | UK      | UK | United Kingdom |
| MA   | Manchester        | UK      | UK | United Kingdom |
| PM   | Palma de Mallorca | ES      | ES | Spain          |
| NULL | NULL              | NULL    | IT | Italy          |
+------+-------------------+---------+----+----------------+
5 rows in set (0,000 sec)
```

All values in the right table

7.1. Types of Joined Relations.

LEFT JOIN VS RIGHT JOIN

Table on the left

Table on the right

```
select *
from towns T
  left outer join countries C on T.country = C.id;
```

```
+----+--------------------+---------+------+----------------+
| id | name               | country | id   | name           |
+----+--------------------+---------+------+----------------+
| AT | Athens             | NULL    | NULL | NULL           |
| LI | Liverpool          | UK      | UK   | United Kingdom |
| LO | London             | UK      | UK   | United Kingdom |
| MA | Manchester         | UK      | UK   | United Kingdom |
| PM | Palma de Mallorca  | ES      | ES   | Spain          |
+----+--------------------+---------+------+----------------+
5 rows in set (0,000 sec)
```

All values in the left table

Table on the left

Table on the right

```
select *
from towns T
  right outer join countries C on T.country = C.id;
```

```
+------+--------------------+---------+----+----------------+
| id   | name               | country | id | name           |
+------+--------------------+---------+----+----------------+
| LI   | Liverpool          | UK      | UK | United Kingdom |
| LO   | London             | UK      | UK | United Kingdom |
| MA   | Manchester         | UK      | UK | United Kingdom |
| PM   | Palma de Mallorca  | ES      | ES | Spain          |
| NULL | NULL               | NULL    | IT | Italy          |
+------+--------------------+---------+----+----------------+
5 rows in set (0,000 sec)
```

All values in the right table

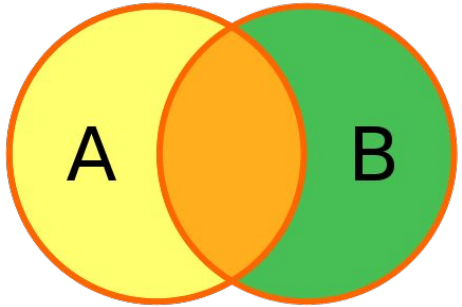
Right and left outer joins may replace each other as long as the table order is switched.

7.1. Types of Joined Relations.

FULL OUTER JOIN

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).



Source: Wikipedia.

7.1. Types of Joined Relations.

FULL OUTER JOIN

Table on the left

Table on the right

MariaDB

```
select *
from towns T
  inner join countries C on T.country = C.id;
```

PostgreSQL

```
select *
from towns T
  full outer join countries C on T.country = C.id;
```

```
+----+-------------------+---------+----+----------------+
| id | name              | country | id | name           |
+----+-------------------+---------+----+----------------+
| LI | Liverpool         | UK      | UK | United Kingdom |
| LO | London            | UK      | UK | United Kingdom |
| MA | Manchester        | UK      | UK | United Kingdom |
| PM | Palma de Mallorca | ES      | ES | Spain          |
+----+-------------------+---------+----+----------------+
4 rows in set (0,000 sec)
```

Only the rows matching

```
 id |       name        | country | id |      name
----+-------------------+---------+----+----------------
 AT | Athens            |         |    |
 LI | Liverpool         | UK      | UK | United Kingdom
 LO | London            | UK      | UK | United Kingdom
 MA | Manchester        | UK      | UK | United Kingdom
 PM | Palma de Mallorca | ES      | ES | Spain
    |                   |         | IT | Italy
(6 rows)
```

All values in both tables

```
MariaDB [minisoccerleague]> select *
    -> from towns T
    ->   full outer join countries C on T.country = C.id;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your
MariaDB server version for the right syntax to use near 'full outer join countries C on T.country =
C.id' at line 3
```

Not implemented in MariaDB!!

7.1. Types of Joined Relations.

FULL OUTER JOIN

Way to do it in MySQL/MariaDB

```
select *
from towns T
  left outer join countries C on T.country = C.id
union
select *
from towns T
  right outer join countries C on T.country = C.id;
```

7.1. Types of Joined Relations.

LEFT VS RIGHT VS FULL

Table on the left

Table on the right

```
select *
from towns T
  left outer join countries C
  on T.country = C.id;
```

```
+----+--------------------+---------+------+----------------+
| id | name               | country | id   | name           |
+----+--------------------+---------+------+----------------+
| AT | Athens             | NULL    | NULL | NULL           |
| LI | Liverpool          | UK      | UK   | United Kingdom |
| LO | London             | UK      | UK   | United Kingdom |
| MA | Manchester         | UK      | UK   | United Kingdom |
| PM | Palma de Mallorca  | ES      | ES   | Spain          |
+----+--------------------+---------+------+----------------+
5 rows in set (0,000 sec)
```

All values in the left table

Table on the left

Table on the right

```
select *
from towns T
  right outer join countries C
  on T.country = C.id;
```

```
+------+-----------------+---------+----+----------------+
| id   | name            | country | id | name           |
+------+-----------------+---------+----+----------------+
| LI   | Liverpool       | UK      | UK | United Kingdom |
| LO   | London          | UK      | UK | United Kingdom |
| MA   | Manchester      | UK      | UK | United Kingdom |
| PM   | Palma de Mallorca | ES    | ES | Spain          |
| NULL | NULL            | NULL    | IT | Italy          |
+------+-----------------+---------+----+----------------+
5 rows in set (0,000 sec)
```

All values in the right table

Table on the left

Table on the right

```
select *
from towns T
  full outer join countries C
on T.country = C.id;
```

```
 id |       name        | country | id |      name
----+-------------------+---------+----+----------------
 AT | Athens            |         |    |
 LI | Liverpool         | UK      | UK | United Kingdom
 LO | London            | UK      | UK | United Kingdom
 MA | Manchester        | UK      | UK | United Kingdom
 PM | Palma de Mallorca | ES      | ES | Spain
    |                   |         | IT | Italy
(6 rows)
```

All values in both tables

```
CREATE TABLE countries (
  id varchar(2) NOT NULL PRIMARY KEY,
  name varchar(40) NOT NULL
);
CREATE TABLE towns (
  id varchar(2) NOT NULL PRIMARY KEY,
  name varchar(40) NOT NULL,
  country varchar(2) REFERENCES countries (id) ON DELETE
  SET NULL
);
```

ALWAYS OUTER JOIN

```
select *
from towns T
  left outer join countries C on T.country = C.id;
```

NULLs allowed in foreign key

7.1. Types of Joined Relations.

Condition in WHERE clause vs. Condition in ON clause.

```
select *
from towns T
  left outer join countries C on T.country = C.id;
```

The left outer without condition

```
 id |      name        | country | id |      name
----+------------------+---------+----+----------------
 AT | Athens           |         |    |
 LI | Liverpool        | UK      | UK | United Kingdom
 LO | London           | UK      | UK | United Kingdom
 MA | Manchester       | UK      | UK | United Kingdom
 PM | Palma de Mallorca | ES     | ES | Spain
(5 rows)
```

```
select *
from towns T
  left outer join countries C on T.country = C.id
  and T.name like 'L%';
```

The join is made only when T.name like 'L%'

```
 id |      name        | country | id |      name
----+------------------+---------+----+----------------
 AT | Athens           |         |    |
 LI | Liverpool        | UK      | UK | United Kingdom
 LO | London           | UK      | UK | United Kingdom
 MA | Manchester       | UK      |    |
 PM | Palma de Mallorca | ES     |    |
(5 rows)
```

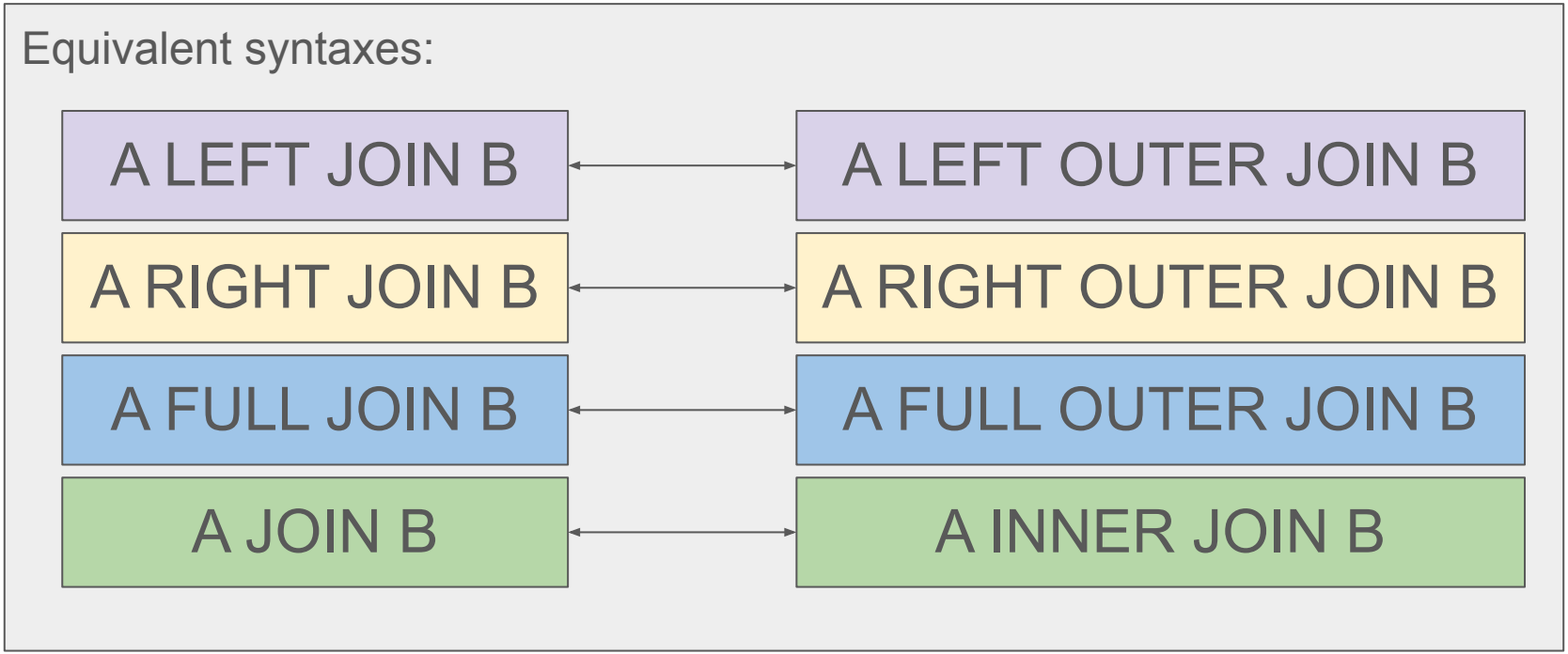
```
select *
from towns T
  left outer join countries C on T.country = C.id
where T.name like 'L%';
```

The left outer without condition and after showing only where T.name like 'L%'

```
 id |   name    | country | id |      name
----+-----------+---------+----+----------------
 LO | London    | UK      | UK | United Kingdom
 LI | Liverpool | UK      | UK | United Kingdom
(2 rows)
```

7.1. Types of Joined Relations.

NOTATION



```
select P.name,
  T.name,
  PT.start_date,
  PT.end_date
from players P
  inner join players_teams PT on P.id = PT.playerid
  inner join teams T on PT.teamid = T.id;
```

```
  name   |       name        | start_date |  end_date
---------+-------------------+------------+------------
 Ronaldo | Manchester United | 2003-08-12 | 2007-08-31
 Ronaldo | Manchester United | 2021-09-01 | 2022-11-22
 Rooney  | Manchester United | 2004-09-01 | 2017-11-22
(3 rows)
```

```
select P.name,
  T.name,
  PT.start_date,
  PT.end_date
from players P
  left outer join players_teams PT on P.id = PT.playerid
  left outer join teams T on PT.teamid = T.id;
```

```
  name   |       name        | start_date |  end_date
---------+-------------------+------------+------------
 Ronaldo | Manchester United | 2003-08-12 | 2007-08-31
 Ronaldo | Manchester United | 2021-09-01 | 2022-11-22
 Rooney  | Manchester United | 2004-09-01 | 2017-11-22
 Mbappe  |                   |            |
(4 rows)
```

```
select P.name,
  T.name,
  PT.start_date,
  PT.end_date
from players P
  right outer join players_teams PT on P.id = PT.playerid
  right outer join teams T on PT.teamid = T.id;
```

```
  name   |       name        | start_date |  end_date
---------+-------------------+------------+------------
 Ronaldo | Manchester United | 2003-08-12 | 2007-08-31
 Ronaldo | Manchester United | 2021-09-01 | 2022-11-22
 Rooney  | Manchester United | 2004-09-01 | 2017-11-22
         | R. C. D. Mallorca |            |
         | Chelsea           |            |
         | Tottenham Hotspur |            |
         | C. D. Amigos      |            |
         | Liverpool         |            |
         | Manchester City   |            |
         | Arsenal           |            |
(10 rows)
```

```
select P.name,
  T.name,
  PT.start_date,
  PT.end_date
from players P
  full outer join players_teams PT on P.id = PT.playerid
  full outer join teams T on PT.teamid = T.id;
```

```
  name   |       name        | start_date |  end_date
---------+-------------------+------------+------------
 Ronaldo | Manchester United | 2003-08-12 | 2007-08-31
 Ronaldo | Manchester United | 2021-09-01 | 2022-11-22
 Rooney  | Manchester United | 2004-09-01 | 2017-11-22
 Mbappe  |                   |            |
         | R. C. D. Mallorca |            |
         | Chelsea           |            |
         | Tottenham Hotspur |            |
         | C. D. Amigos      |            |
         | Liverpool         |            |
         | Manchester City   |            |
         | Arsenal           |            |
(11 rows)
```

```
select P.name,
  T.name,
  PT.start_date,
  PT.end_date
from players P
  left outer join players_teams PT on P.id = PT.playerid
  right outer join teams T on PT.teamid = T.id;
```

```
  name   |       name        | start_date |  end_date
---------+-------------------+------------+------------
 Ronaldo | Manchester United | 2003-08-12 | 2007-08-31
 Ronaldo | Manchester United | 2021-09-01 | 2022-11-22
 Rooney  | Manchester United | 2004-09-01 | 2017-11-22
         | R. C. D. Mallorca |            |
         | Chelsea           |            |
         | Tottenham Hotspur |            |
         | C. D. Amigos      |            |
         | Liverpool         |            |
         | Manchester City   |            |
         | Arsenal           |            |
(10 rows)
```

```
select P.name,
  T.name,
  PT.start_date,
  PT.end_date
from players P
  right outer join players_teams PT on P.id = PT.playerid
  left outer join teams T on PT.teamid = T.id;
```

```
  name   |       name        | start_date |  end_date
---------+-------------------+------------+------------
 Ronaldo | Manchester United | 2003-08-12 | 2007-08-31
 Ronaldo | Manchester United | 2021-09-01 | 2022-11-22
 Rooney  | Manchester United | 2004-09-01 | 2017-11-22
(3 rows)
```

7.1. Types of Joined Relations.

SELF-JOIN

```
create database people;
use people;
create table people (
  id_card varchar(9),
  name varchar(30) NOT NULL,
  surname1 varchar(30) NOT NULL,
  surname2 varchar(30) NOT NULL,
  f ather varchar(9),
  mother varchar(9),
  primary key (id_card),
  foreign key (father) references people (id_card),
  foreign key (mother) references people (id_card));
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');
```

Babies must have an id card...

Unknown mother...

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

Unknown father...

7.1. Types of Joined Relations.

SELF-JOIN

A self-join is joining a table to itself.

Example, people and their fathers:

```
SELECT P.name,
  P.surname1,
  P.surname2,
  F.name as fathername,
  F.surname1 as fathersurname1,
  F.surname2 as fathersurname2
FROM people as P
  LEFT OUTER JOIN people as F ON P.father =
F.id_card
ORDER BY P.surname1,
  P.surname2,
  P.name;
```

MariaDB

name	surname1	surname2	fathername	fathersurname1	fathersurname2
Manuel	Gomis	Munar	NULL	NULL	NULL
Antonio	Gomis	Mut	Manuel	Gomis	Munar
Francisco	Gomis	Mut	Manuel	Gomis	Munar
Josep	Gomis	Mut	Manuel	Gomis	Munar
Antonio	Gual	Mateu	NULL	NULL	NULL
Antonio	Gual	Mir	Antonio	Gual	Mateu
Francisca	Gual	Mir	Antonio	Gual	Mateu
Francisca	Gual	Mir	NULL	NULL	NULL
Pixedis	Gual	Mir	Antonio	Gual	Mateu
Francisca	Mir	Amer	NULL	NULL	NULL
Antonia	Mut	Xamena	NULL	NULL	NULL

11 rows in set (0,000 sec)

7.1. Types of Joined Relations.

SELF-JOIN

Example, people and their fathers and mothers:

```
SELECT P.name,
  P.surname1,
  P.surname2,
  F.name as fathername,
  F.surname1 as fathersurname1,
  F.surname2 as fathersurname2,
  M.name as mothername,
  M.surname1 as mothersurname1,
  M.surname2 as mothersurname2
FROM people as P
  LEFT OUTER JOIN people as F ON P.father = F.id_card
  LEFT OUTER JOIN people as M ON P.mother = M.id_card
ORDER BY P.surname1,
  P.surname2,
  P.name;
```

MariaDB

name	surname1	surname2	fathername	fathersurname1	fathersurname2	mothername	mothersurname1	mothersurname2
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 (0,000 sec)

7.1. Types of Joined Relations.

P07_outer_joins

Publicat el dia Ahir

P07_outer_joins

Documents de Google

Mostra el material

Sample database

```
$ sudo su
$ adduser alumne
$ apt-get update
$ apt-get install postgresql postgresql-client
$ su - postgres
$ cp samplecompany_postgresql.sql ~
$ cd
$ psql
\i samplecompany_postgresql.sql
```

PostgreSQL

```
$ sudo apt update
$ sudo apt install mariadb-server mariadb-client
$ sudo mariadb < samplecompany_mariadb.sql
```

MariaDB

7.2. Window Functions.

It provides a way to perform advanced calculations without having to use subqueries or join operations.

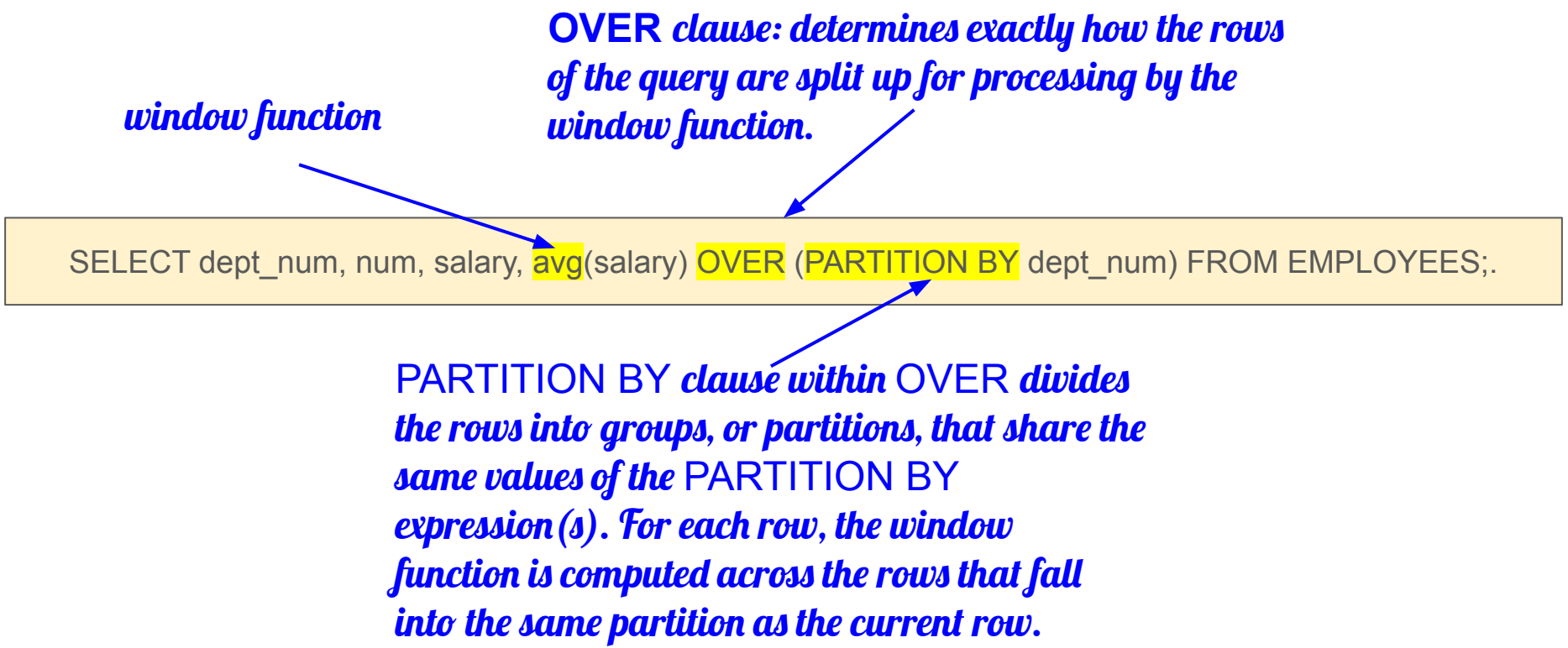
PROBLEM: I want calculations and also the rows of the table.

```
select E1.dept_num,
  E1.num,
  E1.salary,
  T.avg_salary
from employees E1,
  (
    select E.dept_num,
      avg(E.salary) avg_salary
    from employees E
    group by E.dept_num
  ) as T,
  departments D
where D.num = E1.dept_num
  and E1.dept_num = T.dept_num;
```

dept_num	num	salary	avg_salary
10	7782	2885	2287.5000000000000000
10	7934	1690	2287.5000000000000000
20	7566	2900	2178.8888888888888889
20	8001	2885	2178.8888888888888889
20	7369	1040	2178.8888888888888889
20	7788	3000	2178.8888888888888889
20	7876	1430	2178.8888888888888889
20	7877	1430	2178.8888888888888889
20	7902	3000	2178.8888888888888889
20	8000	2885	2178.8888888888888889
20	1000	1040	2178.8888888888888889
30	7654	1600	1735.8333333333333333
30	7698	3005	1735.8333333333333333
30	7521	1625	1735.8333333333333333
30	7499	1500	1735.8333333333333333
30	7844	1350	1735.8333333333333333
30	7900	1335	1735.8333333333333333

(17 rows)

7.2. Window Functions.



7.2. Window Functions.

MariaDB starting with 10.2.

A **window function** performs a calculation across a **set of table rows** that are somehow **related to the current row**.

This is **comparable to** the type of calculation that can be done with an **aggregate function** (but it's not the same!)

```
select dept_num,
  avg(salary)
from employees
group by dept_num;
```

PostgreSQL

```
 dept_num |          avg
----------+-----------------------
       30 | 1735.8333333333333333
       10 | 2287.5000000000000000
       20 | 2178.8888888888888889
(3 rows)
```

```
select dept_num,
  num,
  salary,
  avg(salary) OVER (PARTITION BY dept_num)
from employees;
```

PostgreSQL

```
 dept_num | num  | salary |          avg
----------+------+--------+-----------------------
       10 | 7782 |   2885 | 2287.5000000000000000
       10 | 7934 |   1690 | 2287.5000000000000000
       20 | 7566 |   2900 | 2178.8888888888888889
       20 | 8001 |   2885 | 2178.8888888888888889
       20 | 7369 |   1040 | 2178.8888888888888889
       20 | 7788 |   3000 | 2178.8888888888888889
       20 | 7876 |   1430 | 2178.8888888888888889
       20 | 7877 |   1430 | 2178.8888888888888889
       20 | 7902 |   3000 | 2178.8888888888888889
       20 | 8000 |   2885 | 2178.8888888888888889
       20 | 1000 |   1040 | 2178.8888888888888889
       30 | 7654 |   1600 | 1735.8333333333333333
       30 | 7698 |   3005 | 1735.8333333333333333
       30 | 7521 |   1625 | 1735.8333333333333333
       30 | 7499 |   1500 | 1735.8333333333333333
       30 | 7844 |   1350 | 1735.8333333333333333
       30 | 7900 |   1335 | 1735.8333333333333333
(17 rows)
```

Partition

Partition

Partition

See next slides…

7.2. Window Functions.

```
select dept_num,
  num,
  salary,
  avg(salary) OVER (PARTITION BY dept_num)
from employees;
```

dept_num	num	salary	avg
10	7782	2885	2287.5000000000000000
10	7934	1690	2287.5000000000000000
20	7566	2900	2178.8888888888888889
20	8001	2885	2178.8888888888888889
20	7369	1040	2178.8888888888888889
20	7788	3000	2178.8888888888888889
20	7876	1430	2178.8888888888888889
20	7877	1430	2178.8888888888888889
20	7902	3000	2178.8888888888888889
20	8000	2885	2178.8888888888888889
20	1000	1040	2178.8888888888888889
30	7654	1600	1735.8333333333333333
30	7698	3005	1735.8333333333333333
30	7521	1625	1735.8333333333333333
30	7499	1500	1735.8333333333333333
30	7844	1350	1735.8333333333333333
30	7900	1335	1735.8333333333333333

(17 rows)

```
select dept_num,
  avg(salary)
from employees
group by dept_num;
```

dept_num	avg
30	1735.8333333333333333
10	2287.5000000000000000
20	2178.8888888888888889

(3 rows)

7.2. Window Functions.

*Columns come directly from the **employees table***

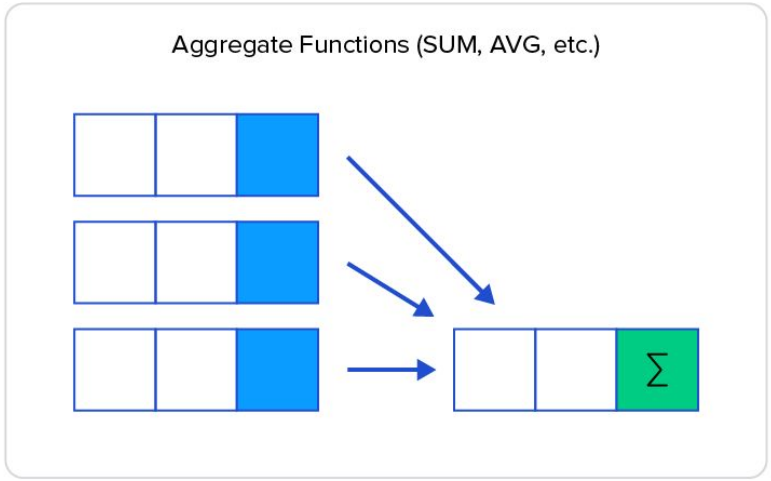
dept_num	num	salary	avg
10	7782	2885	2287.5000000000000000
10	7934	1690	2287.5000000000000000
20	7566	2900	2178.8888888888888889
20	8001	2885	2178.8888888888888889
20	7369	1040	2178.8888888888888889
20	7788	3000	2178.8888888888888889
20	7876	1430	2178.8888888888888889
20	7877	1430	2178.8888888888888889
20	7902	3000	2178.8888888888888889
20	8000	2885	2178.8888888888888889
20	1000	1040	2178.8888888888888889
30	7654	1600	1735.8333333333333333
30	7698	3005	1735.8333333333333333
30	7521	1625	1735.8333333333333333
30	7499	1500	1735.8333333333333333
30	7844	1350	1735.8333333333333333
30	7900	1335	1735.8333333333333333

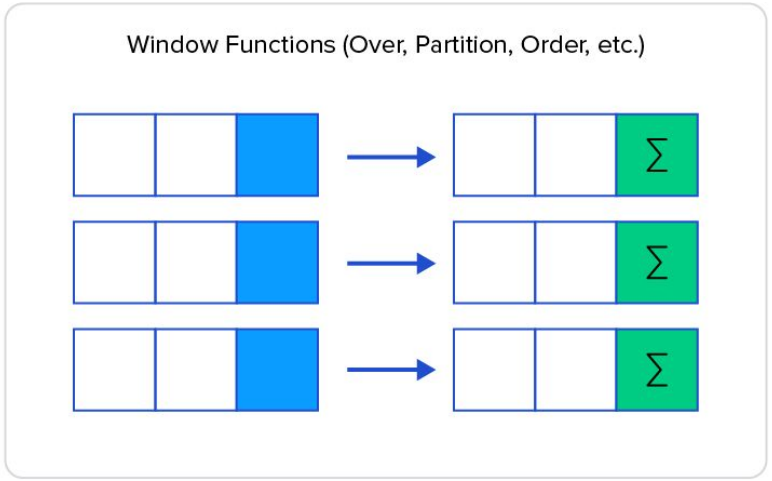
(17 rows)

*There is one output row for each row in the table employees (there is **no where condition**)*

*Represents an **average** taken across all the table **rows** that have the **same dept_num** value as the current row.*

7.2. Window Functions.





Source: https://www.toptal.com/sql/intro-to-sql-windows-functions

7.2. Window Functions.

OVER() is the entire rowset.

```
SELECT COUNT(*) OVER() As
NumEmployees,
    name,
    surname,
    start_date
FROM employees
ORDER BY start_date;
```

```
SELECT (
      SELECT COUNT(*)
      FROM employees
   ) as NumEmployees,
   name,
   surname,
   start_date
FROM employees
ORDER BY start_date;
```

Without window function.

```
 numemployees |   name    |  surname   | start_date
--------------+-----------+------------+------------
           17 | BRAD      | POTTER     | 2004-01-01
           17 | MARTA     | ARROYO     | 2010-02-20
           17 | MARIA     | CEREZO     | 2010-06-09
           17 | JESUS     | GILBERTO   | 2010-11-09
           17 | SERGIO    | SÁNCHEZ    | 2010-12-17
           17 | REBECA    | SALA       | 2011-02-22
           17 | ANTONIA   | MUÑOZ      | 2016-01-23
           17 | ANA       | FERNÁNDEZ  | 2016-12-03
           17 | ANTONIO   | RUIZ       | 2017-01-09
           17 | JUAN      | JIMÉNEZ    | 2017-04-02
           17 | BARTOLOME | GOMIS      | 2017-05-01
           17 | MONICA    | MARTÍN     | 2017-09-29
           17 | XAVIER    | GIMENO     | 2017-12-03
           17 | FERNANDA  | RUIZ       | 2018-06-10
           17 | LUIS      | TOVAR      | 2018-09-08
           17 | FERNANDO  | ALONSO     | 2018-09-23
           17 | LAURA     | ALONSO     | 2020-01-23
(17 rows)
```

7.2. Window Functions.

Number of employees who started the **same month of the same year** -> **partition by**

```
SELECT COUNT(*) OVER (
    PARTITION BY to_char(start_date, 'Month'),
    to_char(start_date, 'YYYY')
  ) As NumPerMonth,
  concat(
    to_char(start_date, 'Month'),
    ' ',
    to_char(start_date, 'YYYY')
  ) As TheMonth,
  name,
  surname
FROM employees
ORDER BY start_date;
```

PostgreSQL

numpermonth	themonth	name	surname
1	January 2004	BRAD	POTTER
1	February 2010	MARTA	ARROYO
1	June 2010	MARIA	CEREZO
1	November 2010	JESUS	GILBERTO
1	December 2010	SERGIO	SÁNCHEZ
1	February 2011	REBECA	SALA
1	January 2016	ANTONIA	MUÑOZ
1	December 2016	ANA	FERNÁNDEZ
1	January 2017	ANTONIO	RUIZ
1	April 2017	JUAN	JIMÉNEZ
1	May 2017	BARTOLOME	GOMIS
1	September 2017	MONICA	MARTÍN
1	December 2017	XAVIER	GIMENO
1	June 2018	FERNANDA	RUIZ
2	September 2018	LUIS	TOVAR
2	September 2018	FERNANDO	ALONSO
1	January 2020	LAURA	ALONSO

(17 rows)

7.2. Window Functions.

Number of employees who started the **same year and the order when they started** -> **partition by**

```
SELECT COUNT(*) OVER (
      PARTITION BY to_char(start_date, 'YYYY')
      ORDER BY start_date
   ) As NumPerMonth,
   to_char(start_date, 'YYYY') As TheYear,
   start_date name,
   surname
FROM employees
ORDER BY start_date;
```

PostgreSQL

numpermonth	theyear	name	surname
1	2004	2004-01-01	POTTER
1	2010	2010-02-20	ARROYO
2	2010	2010-06-09	CEREZO
3	2010	2010-11-09	GILBERTO
4	2010	2010-12-17	SÁNCHEZ
1	2011	2011-02-22	SALA
1	2016	2016-01-23	MUÑOZ
2	2016	2016-12-03	FERNÁNDEZ
1	2017	2017-01-09	RUIZ
2	2017	2017-04-02	JIMÉNEZ
3	2017	2017-05-01	GOMIS
4	2017	2017-09-29	MARTÍN
5	2017	2017-12-03	GIMENO
1	2018	2018-06-10	RUIZ
2	2018	2018-09-08	TOVAR
3	2018	2018-09-23	ALONSO
1	2020	2020-01-23	ALONSO

(17 rows)

7.2. Window Functions.

You can also **control the order in which rows are processed by window functions** using **ORDER BY within OVER**. (The window ORDER BY does not even have to match the order in which the rows are output).

```
SELECT dept_num, num, salary,
        rank() OVER (PARTITION BY dept_num ORDER BY salary DESC)
FROM EMPLOYEES;
```

rank ***function produces a numerical rank for each distinct*** ORDER BY ***value in the current row's partition, using the order defined by the*** ORDER BY ***clause.***

dept_num	num	salary	rank
10	7782	2885	1
10	7934	1690	2
20	7902	3000	1
20	7788	3000	1
20	7566	2900	3
20	8001	2885	4
20	8000	2885	4
20	7876	1430	6
20	7877	1430	6
20	7369	1040	8
20	1000	1040	8
30	7698	3005	1
30	7521	1625	2
30	7654	1600	3
30	7499	1500	4
30	7844	1350	5
30	7900	1335	6

(17 rows)

7.2. Window Functions.

The **window frame** is a set of rows related to the current row where the window function is used for calculation.

The window frame can be a different set of rows for the next row in the query result, since it depends on the current row being processed.

By default, if ORDER BY is supplied then the frame consists of all rows from the start of the partition up through the current row, plus any following rows that are equal to the current row according to the ORDER BY clause.

When ORDER BY is omitted, the default frame consists of all rows in the partition.

```
SELECT salary,
    sum(salary) OVER ()
FROM employees;
```

```
SELECT salary,
    sum(salary) OVER (
        ORDER BY salary
    )
FROM employees;
```

```
select sum(salary)
from employees;
```

```
+-------------+
| sum(salary) |
+-------------+
|       34600 |
+-------------+
1 row in set (0,000 sec)
```

Partition

salary	sum
1040	34600
1040	34600
1500	34600
1625	34600
2900	34600
1600	34600
3005	34600
2885	34600
3000	34600
1350	34600
1430	34600
1430	34600
1335	34600
3000	34600
1690	34600
2885	34600
2885	34600

(17 rows)

salary	sum
1040	2080
1040	2080
1335	3415
1350	4765
1430	7625
1430	7625
1500	9125
1600	10725
1625	12350
1690	14040
2885	22695
2885	22695
2885	22695
2900	25595
3000	31595
3000	31595
3005	34600

(17 rows)

7.2. Window Functions.

```
SELECT salary,
   sum(salary) OVER ()
FROM employees;
```

salary	sum
1040	34600
1040	34600
1500	34600
1625	34600
2900	34600
1600	34600
3005	34600
2885	34600
3000	34600
1350	34600
1430	34600
1430	34600
1335	34600
3000	34600
1690	34600
2885	34600
2885	34600

(17 rows)

Since there is no ORDER BY *in the* OVER *clause, the window frame is the same as the partition, which for lack of* PARTITION BY *is the whole table.*

```
SELECT salary,
   sum(salary) OVER (
      ORDER BY salary
   )
FROM employees;
```

salary	sum
1040	2080
1040	2080
1335	3415
1350	4765
1430	7625
1430	7625
1500	9125
1600	10725
1625	12350
1690	14040
2885	22695
2885	22695
2885	22695
2900	25595
3000	31595
3000	31595
3005	34600

(17 rows)

The sum is taken from the first (lowest) salary up through the current one, including any duplicates of the current one (notice the results for the duplicated salaries).

7.2. Window Functions.

Some common uses of window functions include **ranking**, **cumulative sum**, and **moving average** calculations.

Ranking: You can use the ROW_NUMBER() function to assign a unique rank to each row in a result set based on a specific order.

```
SELECT num,
    salary,
    ROW_NUMBER() OVER (
        ORDER BY salary DESC
    ) AS rank
FROM employees;
```

PostgreSQL

```
 num  | salary | rank
------+--------+------
 7698 |   3005 |    1
 7902 |   3000 |    2
 7788 |   3000 |    3
 7566 |   2900 |    4
 8001 |   2885 |    5
 7782 |   2885 |    6
 8000 |   2885 |    7
 7934 |   1690 |    8
 7521 |   1625 |    9
 7654 |   1600 |   10
 7499 |   1500 |   11
 7876 |   1430 |   12
 7877 |   1430 |   13
 7844 |   1350 |   14
 7900 |   1335 |   15
 7369 |   1040 |   16
 1000 |   1040 |   17
(17 rows)
```

7.2. Window Functions.

Some common uses of window functions include **ranking**, **cumulative sum**, and **moving average** calculations.

Cumulative Sum: You can use the SUM() function along with the OVER() clause to calculate a cumulative sum of a specific column.

```
SELECT num,
    salary,
    SUM(salary) OVER (
        ORDER BY num ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS cumulative_salaries
FROM employees;
```

PostgreSQL

```
 num  | salary | cumulative_salaries
------+--------+---------------------
 1000 |   1040 |                1040
 7369 |   1040 |                2080
 7499 |   1500 |                3580
 7521 |   1625 |                5205
 7566 |   2900 |                8105
 7654 |   1600 |                9705
 7698 |   3005 |               12710
 7782 |   2885 |               15595
 7788 |   3000 |               18595
 7844 |   1350 |               19945
 7876 |   1430 |               21375
 7877 |   1430 |               22805
 7900 |   1335 |               24140
 7902 |   3000 |               27140
 7934 |   1690 |               28830
 8000 |   2885 |               31715
 8001 |   2885 |               34600
(17 rows)
```

7.2. Window Functions.

Some common uses of window functions include **ranking**, **cumulative sum**, and **moving average** calculations.

Moving Average: You can use the AVG() function along with the OVER() clause to calculate a moving average of a specific column.

```
SELECT num,
    salary,
    AVG(salary) OVER (
        ORDER BY num ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS moving_salary
FROM employees;
```

PostgreSQL

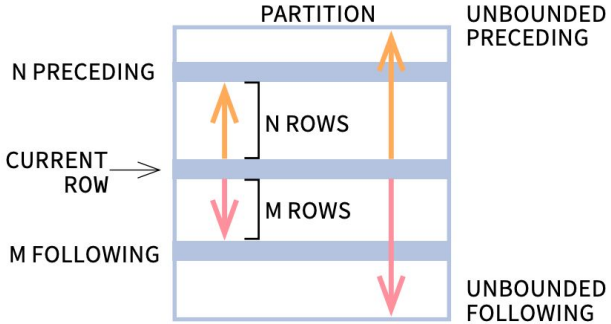
```
 num  | salary |     moving_salary
------+--------+-----------------------
 1000 |   1040 | 1040.0000000000000000
 7369 |   1040 | 1040.0000000000000000
 7499 |   1500 | 1193.3333333333333333
 7521 |   1625 | 1388.3333333333333333
 7566 |   2900 | 2008.3333333333333333
 7654 |   1600 | 2041.6666666666666667
 7698 |   3005 | 2501.6666666666666667
 7782 |   2885 | 2496.6666666666666667
 7788 |   3000 | 2963.3333333333333333
 7844 |   1350 | 2411.6666666666666667
 7876 |   1430 | 1926.6666666666666667
 7877 |   1430 | 1403.3333333333333333
 7900 |   1335 | 1398.3333333333333333
 7902 |   3000 | 1921.6666666666666667
 7934 |   1690 | 2008.3333333333333333
 8000 |   2885 | 2525.0000000000000000
 8001 |   2885 | 2486.6666666666666667
(17 rows)
```

7.2. Window Functions.

The purpose of the **ROWS** clause is to specify the window frame in relation to the current row. The syntax is:
ROWS BETWEEN lower_bound AND upper_bound

The bounds can be any of these five options:

- UNBOUNDED PRECEDING – All rows before the current row.
- n PRECEDING – n rows before the current row.
- CURRENT ROW – Just the current row.
- n FOLLOWING – n rows after the current row.
- UNBOUNDED FOLLOWING – All rows after the current row.



Source:
https://learnsql.com/blog/sql-window-functions-rows-clause/

7.2. Window Functions.

Function	Return Type	Description
row_number()	**bigint**	**number of the current row within its partition, counting from 1**
rank()	**bigint**	**rank of the current row with gaps; same as** row_number **of its first peer**
dense_rank()	**bigint**	**rank of the current row without gaps; this function counts peer groups**
percent_rank()	**double precision**	**relative rank of the current row: (**rank **- 1) / (total rows - 1)**
cume_dist()	**double precision**	**relative rank of the current row: (number of rows preceding or peer with current row) / (total rows)**
ntile(***num_buckets*** integer)	**integer**	**integer ranging from 1 to the argument value, dividing the partition as equally as possible**
lag(***value***anyelement [,***offset*** integer [,***default***anyelement]])	**same type as *value***	**returns *value* evaluated at the row that is *offset* rows before the current row within the partition; if there is no such row, instead return *default* (which must be of the same type as *value*). Both *offset* and *default* are evaluated with respect to the current row. If omitted, *offset* defaults to 1 and *default* to null**

7.2. Window Functions.

Function	Return Type	Description
lead(***value*** anyelement [,***offset*** integer [,***default*** anyelement]])	**same type as *value***	**returns *value* evaluated at the row that is *offset* rows after the current row within the partition; if there is no such row, instead return *default* (which must be of the same type as *value*). Both *offset* and *default* are evaluated with respect to the current row. If omitted, *offset* defaults to 1 and *default* to null**
first_value(***value*** any)	**same type as *value***	**returns *value* evaluated at the row that is the first row of the window frame**
last_value(***value*** any)	**same type as *value***	**returns *value* evaluated at the row that is the last row of the window frame**
nth_value(***value*** any, ***nth*** integer)	**same type as *value***	**returns *value* evaluated at the row that is the *nth* row of the window frame (counting from 1); null if no such row**

- Full list of window functions here: https://www.postgresql.org/docs/current/functions-window.html

7.2. Window Functions.

Window Functions Overview
Window functions perform calculations across a set of rows related to the current row.

AVG
Returns the average value.

BIT_AND
Bitwise AND.

BIT_OR
Bitwise OR.

BIT_XOR
Bitwise XOR.

COUNT
Returns count of non-null values.

CUME_DIST
Window function that returns the cumulative distribution of a given row.

DENSE_RANK
Rank of a given row with identical values receiving the same result, no skipping.

FIRST_VALUE
Returns the first result from an ordered set.

JSON_ARRAYAGG
Returns a JSON array containing an element for each value in a given set of JSON or SQL values.

JSON_ARRAYAGG
Returns a JSON array containing an element for each value in a given set of JSON or SQL values.

JSON_OBJECTAGG
Returns a JSON object containing key-value pairs.

LAG
Accesses data from a previous row in the same result set without the need for a self-join.

LAST_VALUE
Returns the last value in a list or set of values.

LEAD
Accesses data from a following row in the same result set without the need for a self-join.

MAX
Returns the maximum value.

MEDIAN
Window function that returns the median value of a range of values.

MIN
Returns the minimum value.

NTH_VALUE
Returns the value evaluated at the specified row number of the window frame.

NTILE
Returns an integer indicating which group a given row falls into.

7.2. Window Functions.

PERCENTILE_CONT
Continuous percentile.

PERCENTILE_DISC
Discrete percentile.

RANK
Rank of a given row with identical values receiving the same result.

ROW_NUMBER
Row number of a given row with identical values receiving a different result.

STD
Population standard deviation.

STDDEV
Population standard deviation.

STDDEV_POP
Returns the population standard deviation.

STDDEV_SAMP
Standard deviation.

SUM
Sum total.

VAR_POP
Population standard variance.

VAR_SAMP
Returns the sample variance.

VARIANCE
Population standard variance.

Aggregate Functions as Window Functions
It is possible to use aggregate functions as window functions.

ColumnStore Window Functions
Summary of window function use with the ColumnStore engine

Window Frames
Some window functions operate on window frames.

- Full list of window functions here: https://www.postgresql.org/docs/current/tutorial-window.html

7.2. Window Functions.

They are forbidden elsewhere, such as in GROUP BY, HAVING and WHERE clauses.

Window functions are permitted only in the SELECT list and the ORDER BY clause of the query.

We saw it very quickly...

7.2. Window Functions.

P07_window_function

Publicat el dia 16:10

7.3. Subqueries.

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

The nesting can be done in the following SQL query

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

CASE 1:

A_i can be replaced be a subquery that generates a single value.

CASE 2

r_i can be replaced by any valid subquery

CASE 3:

P can be replaced with an expression of the form:

B <operation> (subquery)

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

7.3. Subqueries.

CASE 1:

Show name and surname of employees with their department name:

```
select E.name, E.surname, E.dept_num,
        (select count(*)
        from employees E1
        where E1.dept_num = E.dept_num) as n_dept
    from employees E
    order by E.dept_num;
```

name	surname	dept_num	n_dept
MARIA	CEREZO	10	2
ANTONIA	MUÑOZ	10	2
JUAN	JIMÉNEZ	20	9
FERNANDA	RUIZ	20	9
SERGIO	SÁNCHEZ	20	9
JESUS	GILBERTO	20	9
FERNANDO	ALONSO	20	9
LAURA	ALONSO	20	9
ANA	FERNÁNDEZ	20	9
ANTONIO	RUIZ	20	9
BRAD	POTTER	20	9
MONICA	MARTÍN	30	6
BARTOLOME	GOMIS	30	6
REBECA	SALA	30	6
MARTA	ARROYO	30	6
LUIS	TOVAR	30	6
XAVIER	GIMENO	30	6

(17 rows)

7.3. Subqueries.

CASE 2:

Show name, surname, salary and department number from employees but also the average salary in their department:

```
select E.name, E.surname, E.salary,
            E.dept_num, S.avgsalary
from employees as E,
            (select dept_num,
                  AVG(salary) as avgsalary
            from employees
            group by dept_num) as S
where E.dept_num = S.dept_num;
```

name	surname	salary	dept_num	avgsalary
BRAD	POTTER	1040	20	2178.8888888888888889
SERGIO	SÁNCHEZ	1040	20	2178.8888888888888889
MARTA	ARROYO	1500	30	1735.8333333333333333
REBECA	SALA	1625	30	1735.8333333333333333
JUAN	JIMÉNEZ	2900	20	2178.8888888888888889
MONICA	MARTÍN	1600	30	1735.8333333333333333
BARTOLOME	GOMIS	3005	30	1735.8333333333333333
MARIA	CEREZO	2885	10	2287.5000000000000000
JESUS	GILBERTO	3000	20	2178.8888888888888889
LUIS	TOVAR	1350	30	1735.8333333333333333
FERNANDO	ALONSO	1430	20	2178.8888888888888889
LAURA	ALONSO	1430	20	2178.8888888888888889
XAVIER	GIMENO	1335	30	1735.8333333333333333
ANA	FERNÁNDEZ	3000	20	2178.8888888888888889
ANTONIA	MUÑOZ	1690	10	2287.5000000000000000
ANTONIO	RUIZ	2885	20	2178.8888888888888889
FERNANDA	RUIZ	2885	20	2178.8888888888888889

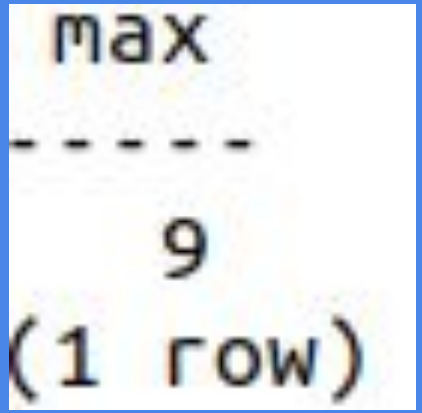
(17 rows)

7.3. Subqueries.

CASE 2:

Find the the maximum number of employees of all the departments:

```
select MAX(T.num_employees)
from (
        select  E.dept_num, count(*) as num_employees
                from employees E
                group by E.dept_num
        ) as T;
```



7.3. Subqueries.

CASE 3:

Show the departments whose average salary is greater than the average of salaries of all employees:

```
 dept_num |          avg
----------+-----------------------
       10 | 2287.5000000000000000
       20 | 2178.8888888888888889
(2 rows)
```

```
select dept_num, avg(salary)
    from employees
    group by dept_num
    having avg(salary) > (select avg(salary) from employees);
```

7.3. Subqueries.

CASE 3:

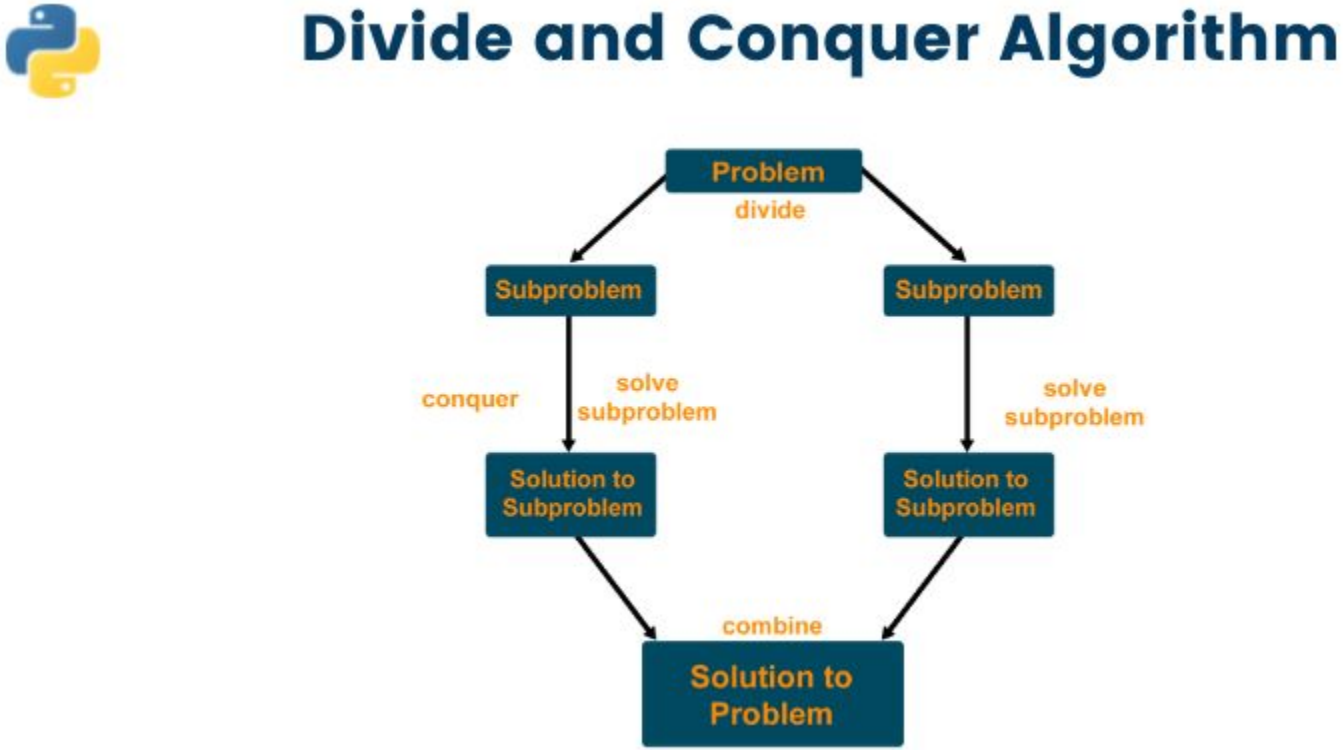
List all employees who are managers:

```
select name, surname
      from employees
      where num IN
            (select distinct manager
            from employees);
```

name	surname
BRAD	POTTER
BARTOLOME	GOMIS
MARIA	CEREZO
JESUS	GILBERTO
ANTONIO	RUIZ
FERNANDA	RUIZ

(6 rows)

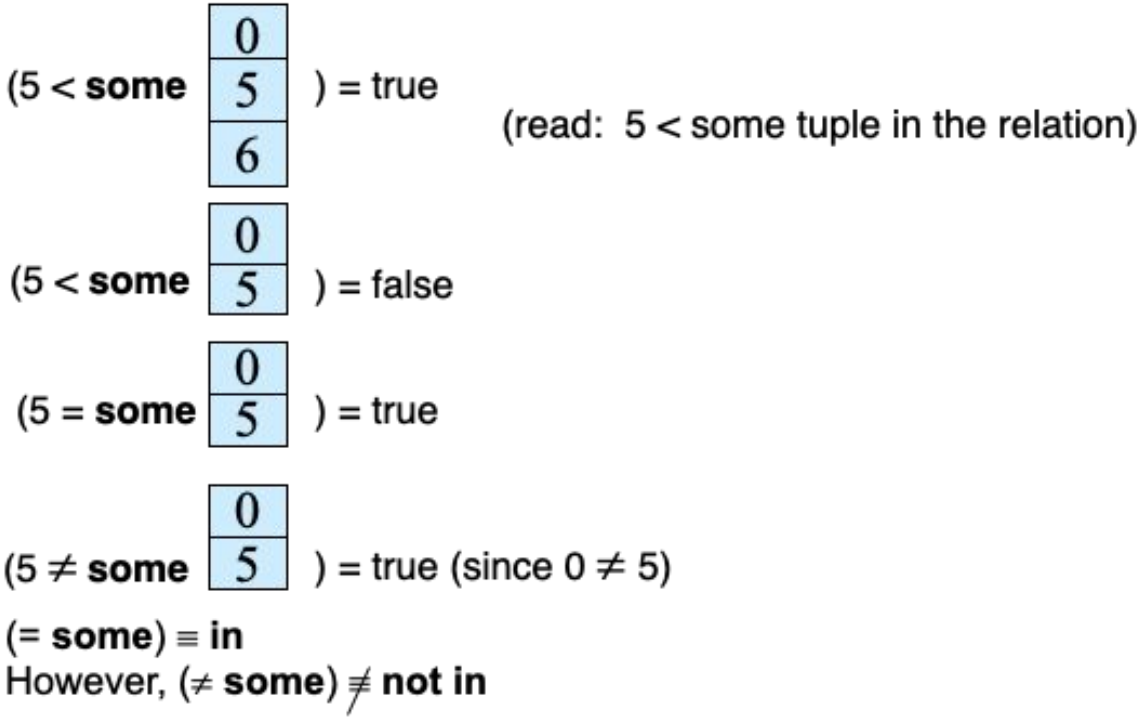
7.3. Subqueries.



Source: https://favtutor.com/blogs/divide-and-conquer-algorithm

7.3. Subqueries (+ set operations).

some



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

7.3. Subqueries (+ set operations).

some

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

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

=

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

```
 salary
--------
   2885
   1690
(2 rows)
```

name	surname	salary
ANTONIO	RUIZ	2885
BARTOLOME	GOMIS	3005
FERNANDA	RUIZ	2885
MARIA	CEREZO	2885
JESUS	GILBERTO	3000
JUAN	JIMÉNEZ	2900
ANA	FERNÁNDEZ	3000

name	surname	salary
JUAN	JIMÉNEZ	2900
BARTOLOME	GOMIS	3005
MARIA	CEREZO	2885
JESUS	GILBERTO	3000
ANA	FERNÁNDEZ	3000
ANTONIO	RUIZ	2885
FERNANDA	RUIZ	2885

7.3. Subqueries (+ set operations).

all

$(5 < \textbf{all}\ \{0, 5, 6\}) = \text{false}$

$(5 < \textbf{all}\ \{6, 10\}) = \text{true}$

$(5 = \textbf{all}\ \{4, 5\}) = \text{false}$

$(5 \neq \textbf{all}\ \{4, 6\}) = \text{true}$ (since $5 \neq 4$ and $5 \neq 6$)

$(\neq \textbf{all}) \equiv \textbf{not in}$

However, $(= \textbf{all}) \neq \textbf{in}$

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

7.3. Subqueries (+ set operations).

all

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

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

name	surname	salary	dept_num
JUAN	JIMÉNEZ	2900	20
BARTOLOME	GOMIS	3005	30
JESUS	GILBERTO	3000	20
ANA	FERNÁNDEZ	3000	20

(4 rows)

7.3. Subqueries (+ set operations).

exists

The **exists** construct returns the value true if the r**esult of the subquery is nonempty**.

exists $r \Leftrightarrow r \neq \emptyset$

not exists $r \Leftrightarrow r = \emptyset$

7.3. Subqueries (+ set operations).

exists

Find the salaries of all employees that are less than the largest salary:

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

```
 salary
--------
   1040
   1335
   1350
   1430
   1500
   1600
   1625
   1690
   2885
   2900
   3000
(11 rows)
```

```
SELECT DISTINCT salary
FROM employees;
```

```
 salary
--------
   1690
   3000
   1335
   2900
   1430
   2885
   3005
   1600
   1500
   1040
   1350
   1625
(12 rows)
```

```
SELECT DISTINCT E.salary
FROM employees E
WHERE NOT EXISTS
      (SELECT *
      FROM employees F
      WHERE
            E.salary < F.salary);
```

```
 salary
--------
   3005
(1 row)
```

Correlated subquery: the inner query.

Correlation name: variable E in the outer query.

7.3. Subqueries (+ set operations).

exists

W3schools:

- "The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns TRUE if the subquery returns one or more records".

```
samplecompany=# insert into towns values ('PMI', 'Palma de Mallorca');
INSERT 0 1
samplecompany=# select code, name
from towns T
where not exists (
    select *
    from departments D
    where D.town_code = T.code
  );
 code |       name
------+------------------
 PMI  | Palma de Mallorca
(1 row)
```

PostgreSQL

7.3. Subqueries (+ set operations).

any

All the employees but the ones with the lowest salary...

W3schools:
"The ANY operator:

- returns a boolean value as a result
- returns **TRUE if ANY of the subquery values meet the condition**

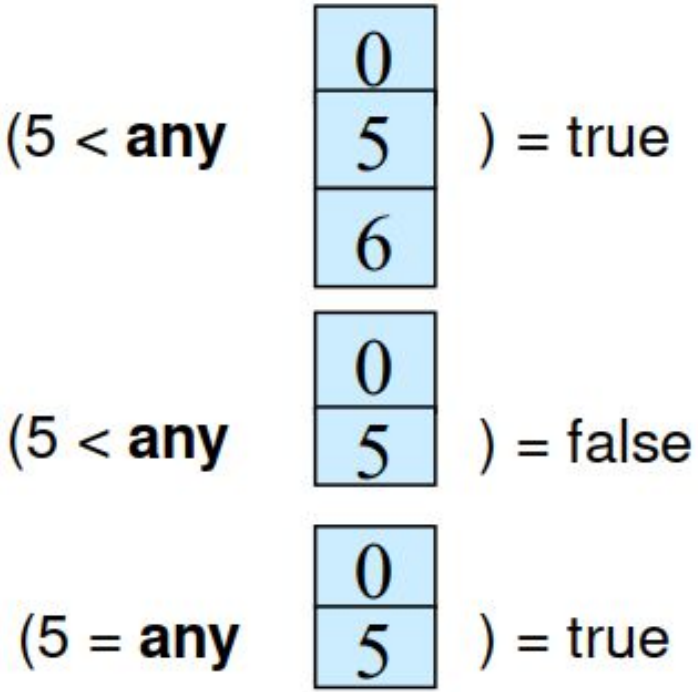
ANY means that the condition will be true if the operation is true for any of the values in the range".

```
samplecompany=# select num,
  salary
from employees
where salary > any (
    select distinct salary
    from employees
  )
order by salary;
 num  | salary
------+--------
 7900 |   1335
 7844 |   1350
 7877 |   1430
 7876 |   1430
 7499 |   1500
 7654 |   1600
 7521 |   1625
 7934 |   1690
 8001 |   2885
 8000 |   2885
 7782 |   2885
 7566 |   2900
 7902 |   3000
 7788 |   3000
 7698 |   3005
(15 rows)
```

PostgreSQL

7.3. Subqueries (+ set operations).

any



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

7.3. Subqueries (+ set operations).

Deletion of tuples from a given relation.

Insertion of new tuples into a given relation.

Updating values in some tuples of a given relation.

7.3. Subqueries (+ modification of the database).

Deletion with subqueries

Deletion of tuples from a given relation.

Delete all employees:
delete from employees;

Delete all employees from the Sales department:

```
delete from employees
where dept_num = 30;

delete from employees
where dept_num = (select num
                  from departments
                  where name='SALES');
```

Delete all tuples in the "employees" relation for those instructors associated with a department located in MADRID:

```
delete from employees
where dept_num IN (select num
                   from departments D, towns T
                   where T.code = D.town_code and
                   T.name='MADRID');
```

7.3. Subqueries (+ modification of the database).

Deletion with subqueries

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

```
delete from employees
where salary < (select avg(salary)
                from employees);
```

```
+-------------+
| avg(salary) |
+-------------+
|  2035.2941  |
+-------------+
```

```
      avg
-----------------------
2035.2941176470588235
```

```
samplecompany=# delete from employees
where salary < (select avg (salary)
                from employees);
DELETE 10
```

```
MariaDB [samplecompany]> delete from employees
    -> where salary < (select avg (salary)
    ->                 from employees);
Query OK, 10 rows affected (0,007 sec)
```

Problem: As we delete tuples from deposit, the average salary changes.

The DBMS first solves the subquery and, with the result of the subquery, executes the main query.

7.3. Subqueries (+ modification of the database).

Insertion with subqueries

Let's imagine that we split the "employees" relation into another table called "managers" and we want to insert the managers there:

```
CREATE TABLE managers (
  num int NOT NULL primary key,
  surname varchar(50) NOT NULL,
  name varchar(50) NOT NULL
  start_date date DEFAULT NULL,
  salary smallint DEFAULT NULL,
  commission smallint DEFAULT NULL,
  dept_num smallint DEFAULT NULL references departments (num)
);
CREATE INDEX idx_managers_fk ON managers (dept_num);
```

```
insert into managers
        select num, surname, name, start_date,
                salary, commission, dept_num
        from employees
        where occu_code = 'MAN';
```

```
samplecompany=# insert into managers
select num, surname, name, start_date, salary, commission, dept_num
from employees
where occu_code = 'MAN';
INSERT 0 5
samplecompany=# select *
samplecompany-# from managers;
 num  | surname |    name    | start_date | salary | commission | dept_num
------+---------+------------+------------+--------+------------+----------
 7566 | JIMÉNEZ | JUAN       | 2017-04-02 |   2900 |            |       20
 7698 | GOMIS   | BARTOLOME  | 2017-05-01 |   3005 |            |       30
 7782 | CEREZO  | MARIA      | 2010-06-09 |   2885 |            |       10
 8000 | RUIZ    | ANTONIO    | 2017-01-09 |   2885 |            |       20
 8001 | RUIZ    | FERNANDA   | 2018-06-10 |   2885 |            |       20
(5 rows)
```

7.3. Subqueries (+ modification of the database).

Updating with subqueries

Increase salaries and commissions of employees whose salary is over $2,500 by 3%, and all others by a 5%:

Two statements:

```
update EMPLOYEES
set salary = salary * 1.03,
        commission = commission * 1.03
where salary > 2500;

update EMPLOYEES
set salary = salary * 1.05,
        commission = commission * 1.05
where salary <= 2500;
```

Order matters, can you see why?

It is best implemented using a case statement (**next slide**).

7.3. Subqueries (+ modification of the database).

Updating with subqueries

Increase salaries and commissions of employees whose salary is over $2,500 by 3%, and all others by a 5%:

```
update employees
set commission = case
    when salary <= 2500 then commission * 1.05
    else commission * 1.03
  end,
  salary = case
    when salary <= 2500 then salary * 1.05
    else salary * 1.03
  end;
```

https://www.postgresql.org/docs/current/functions-conditional.html

https://mariadb.com/kb/en/case-operator/

7.3. Subqueries (+ modification of the database).

Updating with subqueries

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

```
update employees E
set E.salary = (
    select avg(salary)
    from employees F
    where E.dept_num = F.dept_num
      and F.salary <= 1800
  )
where E.salary < 1200;
```

```
MariaDB [samplecompany]> select num, salary
    -> from employees;
+------+--------+
| num  | salary |
+------+--------+
| 1000 |   1040 |
| 7369 |   1040 |
| 7499 |   1500 |
| 7521 |   1625 |
| 7566 |   2900 |
| 7654 |   1600 |
| 7698 |   3005 |
| 7782 |   2885 |
| 7788 |   3000 |
| 7844 |   1350 |
| 7876 |   1430 |
| 7877 |   1430 |
| 7900 |   1335 |
| 7902 |   3000 |
| 7934 |   1690 |
| 8000 |   2885 |
| 8001 |   2885 |
+------+--------+
17 rows in set (0,000 sec)
```

```
MariaDB [samplecompany]> select F.dept_num, avg(F.salary)
    -> from employees F
    -> where F.salary <= 1800
    -> group by dept_num;
+----------+---------------+
| dept_num | avg(F.salary) |
+----------+---------------+
|       10 |     1690.0000 |
|       20 |     1235.0000 |
|       30 |     1482.0000 |
+----------+---------------+
3 rows in set (0,000 sec)
```

7.4. Views.

Views allow you to **encapsulate the details of the structure of your tables**, which might change as your application evolves, behind consistent interfaces.

Views can be used in almost any place a real table can be used. **Building views upon other views** is not uncommon.

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;
```

7.4. Views.

Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.

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.

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.

A **view** provides a mechanism to **hide certain data** from the view of **certain users**.

7.4. Views.

```
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;
```

```
 num |    name     | num_employees
-----+-------------+---------------
  10 | ACCOUNTING  |             2
  20 | RESEARCH    |             9
  30 | SALES       |             6
(3 rows)
```

```
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;
```

```
 num |    name     | num_employees
-----+-------------+---------------
  40 | PRODUCTION  |             0
  10 | ACCOUNTING  |             2
  30 | SALES       |             6
  20 | RESEARCH    |             9
(4 rows)
```

7.4. Views.

Views are also **updatable** (inserts, updates, and deletes).

There are also **materialized views**: They are also views (logical) but the result of the query will get stored in disk.

But you'll see this next year…

7.5. String functions and operators.

```
select name,
 LENGTH(name),
 surname,
 LENGTH(surname)
from employees;
```

LENGTH(str) - returns the length in bytes of the string str.

name	length	surname	length
BRAD	4	POTTER	6
SERGIO	6	SÁNCHEZ	7
MARTA	5	ARROYO	6
REBECA	6	SALA	4
JUAN	4	JIMÉNEZ	7
MONICA	6	MARTÍN	6
BARTOLOME	9	GOMIS	5
MARIA	5	CEREZO	6
JESUS	5	GILBERTO	8
LUIS	4	TOVAR	5
FERNANDO	8	ALONSO	6
LAURA	5	ALONSO	6
XAVIER	6	GIMENO	6
ANA	3	FERNÁNDEZ	9
ANTONIA	7	MUÑOZ	5
ANTONIO	7	RUIZ	4
FERNANDA	8	RUIZ	4

(17 rows)

name	LENGTH(name)	surname	LENGTH(surname)
BRAD	4	POTTER	6
SERGIO	6	SÁNCHEZ	7
MARTA	5	ARROYO	6
REBECA	6	SALA	4
JUAN	4	JIMÉNEZ	7
MONICA	6	MARTÍN	6
BARTOLOME	9	GOMIS	5
MARIA	5	CEREZO	6
JESUS	5	GILBERTO	8
LUIS	4	TOVAR	5
FERNANDO	8	ALONSO	6
LAURA	5	ALONSO	6
XAVIER	6	GIMENO	6
ANA	3	FERNÁNDEZ	9
ANTONIA	7	MUÑOZ	5
ANTONIO	7	RUIZ	4
FERNANDA	8	RUIZ	4

17 rows in set (0,006 sec)

7.5. String functions and operators.

```
select name,
 LENGTH(name),
 CHAR_LENGTH(name)
from employees;
```

CHAR_LENGTH(str) - returns the length of the string str in characters, rather than bytes.

```
+---------------------+--------------------------+
| length(_utf8 '€')   | char_length(_utf8 '€')   |
+---------------------+--------------------------+
|                   3 |                        1 |
+---------------------+--------------------------+
1 row in set (0,000 sec)
```

```
select length(_utf8 '€'),
 char_length(_utf8 '€');
```

```
+-----------+--------------+--------------------+
| name      | LENGTH(name) | CHAR_LENGTH(name)  |
+-----------+--------------+--------------------+
| BRAD      |            4 |                  4 |
| SERGIO    |            6 |                  6 |
| MARTA     |            5 |                  5 |
| REBECA    |            6 |                  6 |
| JUAN      |            4 |                  4 |
| MONICA    |            6 |                  6 |
| BARTOLOME |            9 |                  9 |
| MARIA     |            5 |                  5 |
| JESUS     |            5 |                  5 |
| LUIS      |            4 |                  4 |
| FERNANDO  |            8 |                  8 |
| LAURA     |            5 |                  5 |
| XAVIER    |            6 |                  6 |
| ANA       |            3 |                  3 |
| ANTONIA   |            7 |                  7 |
| ANTONIO   |            7 |                  7 |
| FERNANDA  |            8 |                  8 |
+-----------+--------------+--------------------+
17 rows in set (0,000 sec)
```

7.5. String functions and operators.

```
select name,
 UPPER(name),
 surname,
 UPPER(surname)
from employees;
```

UPPER(str) - returns the string str with all characters in uppercase.

name	upper	surname	upper
BRAD	BRAD	POTTER	POTTER
SERGIO	SERGIO	SÁNCHEZ	SÁNCHEZ
MARTA	MARTA	ARROYO	ARROYO
REBECA	REBECA	SALA	SALA
JUAN	JUAN	JIMÉNEZ	JIMÉNEZ
MONICA	MONICA	MARTÍN	MARTÍN
BARTOLOME	BARTOLOME	GOMIS	GOMIS
MARIA	MARIA	CEREZO	CEREZO
JESUS	JESUS	GILBERTO	GILBERTO
LUIS	LUIS	TOVAR	TOVAR
FERNANDO	FERNANDO	ALONSO	ALONSO
LAURA	LAURA	ALONSO	ALONSO
XAVIER	XAVIER	GIMENO	GIMENO
ANA	ANA	FERNÁNDEZ	FERNÁNDEZ
ANTONIA	ANTONIA	MUÑOZ	MUÑOZ
ANTONIO	ANTONIO	RUIZ	RUIZ
FERNANDA	FERNANDA	RUIZ	RUIZ

(17 rows)

name	UPPER(name)	surname	UPPER(surname)
BRAD	BRAD	POTTER	POTTER
SERGIO	SERGIO	SÁNCHEZ	SÁNCHEZ
MARTA	MARTA	ARROYO	ARROYO
REBECA	REBECA	SALA	SALA
JUAN	JUAN	JIMÉNEZ	JIMÉNEZ
MONICA	MONICA	MARTÍN	MARTÍN
BARTOLOME	BARTOLOME	GOMIS	GOMIS
MARIA	MARIA	CEREZO	CEREZO
JESUS	JESUS	GILBERTO	GILBERTO
LUIS	LUIS	TOVAR	TOVAR
FERNANDO	FERNANDO	ALONSO	ALONSO
LAURA	LAURA	ALONSO	ALONSO
XAVIER	XAVIER	GIMENO	GIMENO
ANA	ANA	FERNÁNDEZ	FERNÁNDEZ
ANTONIA	ANTONIA	MUÑOZ	MUÑOZ
ANTONIO	ANTONIO	RUIZ	RUIZ
FERNANDA	FERNANDA	RUIZ	RUIZ

17 rows in set (0,000 sec)

7.5. String functions and operators.

```
select name,
 LOWER(name),
 surname,
 LOWER(surname)
from employees;
```

LOWER(str) - returns the string str with all characters in lowercase.

name	lower	surname	lower
BRAD	brad	POTTER	potter
SERGIO	sergio	SÁNCHEZ	sánchez
MARTA	marta	ARROYO	arroyo
REBECA	rebeca	SALA	sala
JUAN	juan	JIMÉNEZ	jiménez
MONICA	monica	MARTÍN	martín
BARTOLOME	bartolome	GOMIS	gomis
MARIA	maria	CEREZO	cerezo
JESUS	jesus	GILBERTO	gilberto
LUIS	luis	TOVAR	tovar
FERNANDO	fernando	ALONSO	alonso
LAURA	laura	ALONSO	alonso
XAVIER	xavier	GIMENO	gimeno
ANA	ana	FERNÁNDEZ	fernández
ANTONIA	antonia	MUÑOZ	muñoz
ANTONIO	antonio	RUIZ	ruiz
FERNANDA	fernanda	RUIZ	ruiz

(17 rows)

name	LOWER(name)	surname	LOWER(surname)
BRAD	brad	POTTER	potter
SERGIO	sergio	SÁNCHEZ	sánchez
MARTA	marta	ARROYO	arroyo
REBECA	rebeca	SALA	sala
JUAN	juan	JIMÉNEZ	jiménez
MONICA	monica	MARTÍN	martín
BARTOLOME	bartolome	GOMIS	gomis
MARIA	maria	CEREZO	cerezo
JESUS	jesus	GILBERTO	gilberto
LUIS	luis	TOVAR	tovar
FERNANDO	fernando	ALONSO	alonso
LAURA	laura	ALONSO	alonso
XAVIER	xavier	GIMENO	gimeno
ANA	ana	FERNÁNDEZ	fernández
ANTONIA	antonia	MUÑOZ	muñoz
ANTONIO	antonio	RUIZ	ruiz
FERNANDA	fernanda	RUIZ	ruiz

17 rows in set (0,000 sec)

7.5. String functions and operators.

```
select name,
 LEFT(name,2),
 surname,
 LEFT(surname,2)
from employees;
```

LEFT(str, len) - returns the leftmost len characters from the string str.

name	left	surname	left
BRAD	BR	POTTER	PO
SERGIO	SE	SÁNCHEZ	SÁ
MARTA	MA	ARROYO	AR
REBECA	RE	SALA	SA
JUAN	JU	JIMÉNEZ	JI
MONICA	MO	MARTÍN	MA
BARTOLOME	BA	GOMIS	GO
MARIA	MA	CEREZO	CE
JESUS	JE	GILBERTO	GI
LUIS	LU	TOVAR	TO
FERNANDO	FE	ALONSO	AL
LAURA	LA	ALONSO	AL
XAVIER	XA	GIMENO	GI
ANA	AN	FERNÁNDEZ	FE
ANTONIA	AN	MUÑOZ	MU
ANTONIO	AN	RUIZ	RU
FERNANDA	FE	RUIZ	RU

(17 rows)

name	LEFT(name,2)	surname	LEFT(surname,2)
BRAD	BR	POTTER	PO
SERGIO	SE	SÁNCHEZ	SÁ
MARTA	MA	ARROYO	AR
REBECA	RE	SALA	SA
JUAN	JU	JIMÉNEZ	JI
MONICA	MO	MARTÍN	MA
BARTOLOME	BA	GOMIS	GO
MARIA	MA	CEREZO	CE
JESUS	JE	GILBERTO	GI
LUIS	LU	TOVAR	TO
FERNANDO	FE	ALONSO	AL
LAURA	LA	ALONSO	AL
XAVIER	XA	GIMENO	GI
ANA	AN	FERNÁNDEZ	FE
ANTONIA	AN	MUÑOZ	MU
ANTONIO	AN	RUIZ	RU
FERNANDA	FE	RUIZ	RU

17 rows in set (0,000 sec)

7.5. String functions and operators.

```
select name,
 RIGHT(name,2),
 surname,
 RIGHT(surname,2)
from employees;
```

RIGHT(str, len) - returns the rightmost len characters from the string str.

name	right	surname	right
BRAD	AD	POTTER	ER
SERGIO	IO	SÁNCHEZ	EZ
MARTA	TA	ARROYO	YO
REBECA	CA	SALA	LA
JUAN	AN	JIMÉNEZ	EZ
MONICA	CA	MARTÍN	ÍN
BARTOLOME	ME	GOMIS	IS
MARIA	IA	CEREZO	ZO
JESUS	US	GILBERTO	TO
LUIS	IS	TOVAR	AR
FERNANDO	DO	ALONSO	SO
LAURA	RA	ALONSO	SO
XAVIER	ER	GIMENO	NO
ANA	NA	FERNÁNDEZ	EZ
ANTONIA	IA	MUÑOZ	OZ
ANTONIO	IO	RUIZ	IZ
FERNANDA	DA	RUIZ	IZ

(17 rows)

name	RIGHT(name,2)	surname	RIGHT(surname,2)
BRAD	AD	POTTER	ER
SERGIO	IO	SÁNCHEZ	EZ
MARTA	TA	ARROYO	YO
REBECA	CA	SALA	LA
JUAN	AN	JIMÉNEZ	EZ
MONICA	CA	MARTÍN	ÍN
BARTOLOME	ME	GOMIS	IS
MARIA	IA	CEREZO	ZO
JESUS	US	GILBERTO	TO
LUIS	IS	TOVAR	AR
FERNANDO	DO	ALONSO	SO
LAURA	RA	ALONSO	SO
XAVIER	ER	GIMENO	NO
ANA	NA	FERNÁNDEZ	EZ
ANTONIA	IA	MUÑOZ	OZ
ANTONIO	IO	RUIZ	IZ
FERNANDA	DA	RUIZ	IZ

17 rows in set (0,000 sec)

7.5. String functions and operators.

```
select name,
 SUBSTR(name,2,4),
 surname,
 SUBSTR(surname,2,4)
from employees;
```

SUBSTRING(str, start, len) - returns a substring of length len from the string str, starting at position start.

name	substr	surname	substr
BRAD	RAD	POTTER	OTTE
SERGIO	ERGI	SÁNCHEZ	ÁNCH
MARTA	ARTA	ARROYO	RROY
REBECA	EBEC	SALA	ALA
JUAN	UAN	JIMÉNEZ	IMÉN
MONICA	ONIC	MARTÍN	ARTÍ
BARTOLOME	ARTO	GOMIS	OMIS
MARIA	ARIA	CEREZO	EREZ
JESUS	ESUS	GILBERTO	ILBE
LUIS	UIS	TOVAR	OVAR
FERNANDO	ERNA	ALONSO	LONS
LAURA	AURA	ALONSO	LONS
XAVIER	AVIE	GIMENO	IMEN
ANA	NA	FERNÁNDEZ	ERNÁ
ANTONIA	NTON	MUÑOZ	UÑOZ
ANTONIO	NTON	RUIZ	UIZ
FERNANDA	ERNA	RUIZ	UIZ

(17 rows)

name	SUBSTR(name,2,4)	surname	SUBSTR(surname,2,4)
BRAD	RAD	POTTER	OTTE
SERGIO	ERGI	SÁNCHEZ	ÁNCH
MARTA	ARTA	ARROYO	RROY
REBECA	EBEC	SALA	ALA
JUAN	UAN	JIMÉNEZ	IMÉN
MONICA	ONIC	MARTÍN	ARTÍ
BARTOLOME	ARTO	GOMIS	OMIS
MARIA	ARIA	CEREZO	EREZ
JESUS	ESUS	GILBERTO	ILBE
LUIS	UIS	TOVAR	OVAR
FERNANDO	ERNA	ALONSO	LONS
LAURA	AURA	ALONSO	LONS
XAVIER	AVIE	GIMENO	IMEN
ANA	NA	FERNÁNDEZ	ERNÁ
ANTONIA	NTON	MUÑOZ	UÑOZ
ANTONIO	NTON	RUIZ	UIZ
FERNANDA	ERNA	RUIZ	UIZ

17 rows in set (0,000 sec)

7.5. String functions and operators.

```
select name,
 REPLACE(name, 'AN', 'ES'),
 surname,
 REPLACE(surname, 'EZ', '')
from employees;
```

REPLACE(str, old, new) - returns the string str with all occurrences of the substring old replaced with the substring new.

name	replace	surname	replace
BRAD	BRAD	POTTER	POTTER
SERGIO	SERGIO	SÁNCHEZ	SÁNCH
MARTA	MARTA	ARROYO	ARROYO
REBECA	REBECA	SALA	SALA
JUAN	JUES	JIMÉNEZ	JIMÉN
MONICA	MONICA	MARTÍN	MARTÍN
BARTOLOME	BARTOLOME	GOMIS	GOMIS
MARIA	MARIA	CEREZO	CERO
JESUS	JESUS	GILBERTO	GILBERTO
LUIS	LUIS	TOVAR	TOVAR
FERNANDO	FERNESDO	ALONSO	ALONSO
LAURA	LAURA	ALONSO	ALONSO
XAVIER	XAVIER	GIMENO	GIMENO
ANA	ESA	FERNÁNDEZ	FERNÁND
ANTONIA	ESTONIA	MUÑOZ	MUÑOZ
ANTONIO	ESTONIO	RUIZ	RUIZ
FERNANDA	FERNESDA	RUIZ	RUIZ

(17 rows)

name	REPLACE(name, 'AN', 'ES')	surname	REPLACE(surname, 'EZ', '')
BRAD	BRAD	POTTER	POTTER
SERGIO	SERGIO	SÁNCHEZ	SÁNCH
MARTA	MARTA	ARROYO	ARROYO
REBECA	REBECA	SALA	SALA
JUAN	JUES	JIMÉNEZ	JIMÉN
MONICA	MONICA	MARTÍN	MARTÍN
BARTOLOME	BARTOLOME	GOMIS	GOMIS
MARIA	MARIA	CEREZO	CERO
JESUS	JESUS	GILBERTO	GILBERTO
LUIS	LUIS	TOVAR	TOVAR
FERNANDO	FERNESDO	ALONSO	ALONSO
LAURA	LAURA	ALONSO	ALONSO
XAVIER	XAVIER	GIMENO	GIMENO
ANA	ESA	FERNÁNDEZ	FERNÁND
ANTONIA	ESTONIA	MUÑOZ	MUÑOZ
ANTONIO	ESTONIO	RUIZ	RUIZ
FERNANDA	FERNESDA	RUIZ	RUIZ

17 rows in set (0,000 sec)

7.5. String functions and operators.

```
select CONCAT(num, ' ', name, ' ',
surname)
from employees;
```

CONCAT(str1, str2) - returns the concatenation of the two strings str1 and str2.

```
          concat
----------------------
 1000 BRAD POTTER
 7369 SERGIO SÁNCHEZ
 7499 MARTA ARROYO
 7521 REBECA SALA
 7566 JUAN JIMÉNEZ
 7654 MONICA MARTÍN
 7698 BARTOLOME GOMIS
 7782 MARIA CEREZO
 7788 JESUS GILBERTO
 7844 LUIS TOVAR
 7876 FERNANDO ALONSO
 7877 LAURA ALONSO
 7900 XAVIER GIMENO
 7902 ANA FERNÁNDEZ
 7934 ANTONIA MUÑOZ
 8000 ANTONIO RUIZ
 8001 FERNANDA RUIZ
(17 rows)
```

```
+--------------------------------------+
| CONCAT(num, ' ', name, ' ', surname) |
+--------------------------------------+
| 1000 BRAD POTTER                     |
| 7369 SERGIO SÁNCHEZ                  |
| 7499 MARTA ARROYO                    |
| 7521 REBECA SALA                     |
| 7566 JUAN JIMÉNEZ                    |
| 7654 MONICA MARTÍN                   |
| 7698 BARTOLOME GOMIS                 |
| 7782 MARIA CEREZO                    |
| 7788 JESUS GILBERTO                  |
| 7844 LUIS TOVAR                      |
| 7876 FERNANDO ALONSO                 |
| 7877 LAURA ALONSO                    |
| 7900 XAVIER GIMENO                   |
| 7902 ANA FERNÁNDEZ                   |
| 7934 ANTONIA MUÑOZ                   |
| 8000 ANTONIO RUIZ                    |
| 8001 FERNANDA RUIZ                   |
+--------------------------------------+
17 rows in set (0,000 sec)
```

7.5. String functions and operators.

```
select CONCAT_WS(' ', num, name, surname)
from employees;
```

CONCAT_WS(separator, str1, str2, ...) - returns the concatenation of two or more strings str1, str2, ..., with the specified separator between each string. The separator is only added between non-NULL values.

```
      concat_ws
----------------------
 1000 BRAD POTTER
 7369 SERGIO SÁNCHEZ
 7499 MARTA ARROYO
 7521 REBECA SALA
 7566 JUAN JIMÉNEZ
 7654 MONICA MARTÍN
 7698 BARTOLOME GOMIS
 7782 MARIA CEREZO
 7788 JESUS GILBERTO
 7844 LUIS TOVAR
 7876 FERNANDO ALONSO
 7877 LAURA ALONSO
 7900 XAVIER GIMENO
 7902 ANA FERNÁNDEZ
 7934 ANTONIA MUÑOZ
 8000 ANTONIO RUIZ
 8001 FERNANDA RUIZ
(17 rows)
```

```
+-----------------------------------+
| CONCAT_WS(' ', num, name, surname) |
+-----------------------------------+
| 1000 BRAD POTTER                  |
| 7369 SERGIO SÁNCHEZ               |
| 7499 MARTA ARROYO                 |
| 7521 REBECA SALA                  |
| 7566 JUAN JIMÉNEZ                 |
| 7654 MONICA MARTÍN                |
| 7698 BARTOLOME GOMIS              |
| 7782 MARIA CEREZO                 |
| 7788 JESUS GILBERTO               |
| 7844 LUIS TOVAR                   |
| 7876 FERNANDO ALONSO              |
| 7877 LAURA ALONSO                 |
| 7900 XAVIER GIMENO                |
| 7902 ANA FERNÁNDEZ                |
| 7934 ANTONIA MUÑOZ                |
| 8000 ANTONIO RUIZ                 |
| 8001 FERNANDA RUIZ                |
+-----------------------------------+
17 rows in set (0,000 sec)
```

7.5. String functions and operators.

TRIM(str) - returns the string str with leading and trailing whitespace removed. Also LTRIM and RTRIM.

```
     concat      |  concat   |    concat     |    concat
-----------------+-----------+---------------+--------------
 ***   AAA   *** | ***AAA*** | ***AAA   *** | ***   AAA***
(1 row)
```

```
+--------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| CONCAT('***', '   AAA   ', '***') | CONCAT('***', TRIM('   AAA   '), '***') | CONCAT('***', LTRIM('   AAA   '), '***') | CONCAT('***', RTRIM('   AAA   '), '***') |
+--------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
| ***   AAA   ***                | ***AAA***                              | ***AAA   ***                           | ***   AAA***                           |
+--------------------------------+----------------------------------------+----------------------------------------+----------------------------------------+
1 row in set (0,000 sec)
```

```
select CONCAT('***', '   AAA   ', '***'),
 CONCAT('***', TRIM('   AAA   '), '***'),
 CONCAT('***', LTRIM('   AAA   '), '***'),
 CONCAT('***', RTRIM('   AAA   '), '***');
```

7.5. String functions and operators.

```
select name,
 POSITION('A' IN name)
from employees;
```

POSITION(substr IN str) - returns the position of the substring substr within the string str.

name	position
BRAD	3
SERGIO	0
MARTA	2
REBECA	6
JUAN	3
MONICA	6
BARTOLOME	2
MARIA	2
JESUS	0
LUIS	0
FERNANDO	5
LAURA	2
XAVIER	2
ANA	1
ANTONIA	1
ANTONIO	1
FERNANDA	5

(17 rows)

name	POSITION('A' IN name)
BRAD	3
SERGIO	0
MARTA	2
REBECA	6
JUAN	3
MONICA	6
BARTOLOME	2
MARIA	2
JESUS	0
LUIS	0
FERNANDO	5
LAURA	2
XAVIER	2
ANA	1
ANTONIA	1
ANTONIO	1
FERNANDA	5

17 rows in set (0,000 sec)

7.5. String functions and operators.

```
select name,
 POSITION('A' IN name)
from employees;
```

INITCAP(str) - returns the string str with the first letter of each word capitalized.

NOT available in MariaDB.

Check this link.

Nested functions

```
select CONCAT_WS(
   '',
   CONCAT(
     UPPER(LEFT(E.name, 1)),
     LOWER(SUBSTRING(E.name, 2))
   ),
   CONCAT(
     UPPER(LEFT(E.surname, 1)),
     LOWER(SUBSTRING(E.surname, 2))
   )
  ) AS FullName
from employees E;
```

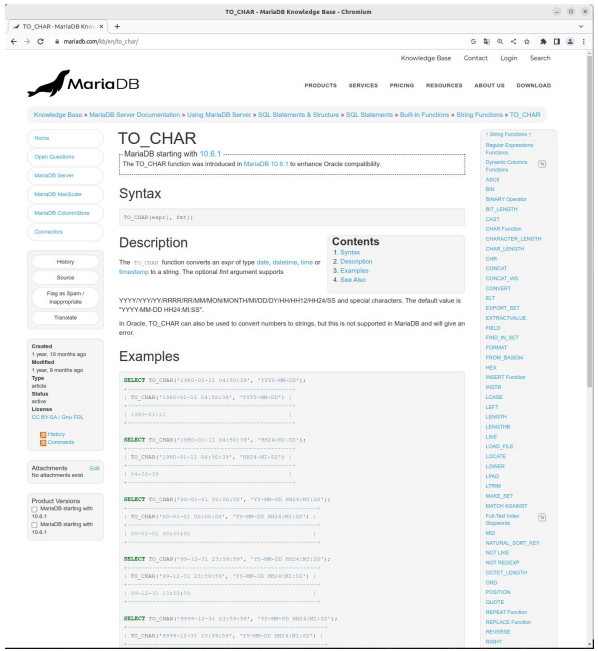
MariaDB

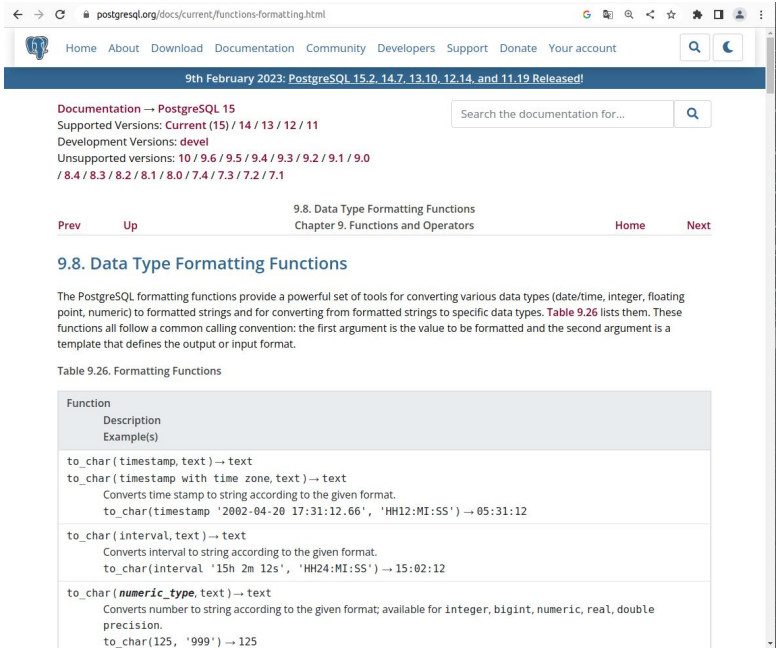
name	initcap	surname	initcap
BRAD	Brad	POTTER	Potter
SERGIO	Sergio	SÁNCHEZ	Sánchez
MARTA	Marta	ARROYO	Arroyo
REBECA	Rebeca	SALA	Sala
JUAN	Juan	JIMÉNEZ	Jiménez
MONICA	Monica	MARTÍN	Martín
BARTOLOME	Bartolome	GOMIS	Gomis
MARIA	Maria	CEREZO	Cerezo
JESUS	Jesus	GILBERTO	Gilberto
LUIS	Luis	TOVAR	Tovar
FERNANDO	Fernando	ALONSO	Alonso
LAURA	Laura	ALONSO	Alonso
XAVIER	Xavier	GIMENO	Gimeno
ANA	Ana	FERNÁNDEZ	Fernández
ANTONIA	Antonia	MUÑOZ	Muñoz
ANTONIO	Antonio	RUIZ	Ruiz
FERNANDA	Fernanda	RUIZ	Ruiz

(17 rows)

7.5. String functions and operators.

TO_CHAR: Find it out how it works!





7.5. String functions and operators.

References:

https://www.postgresql.org/docs/current/functions-string.html

PostgreSQL

https://mariadb.com/kb/en/string-functions/

MariaDB

7.5. String functions and operators.

P07_strings

Data de venciment:

7.6. Date functions and operators.

date — Dates, containing a (4 digit) year, month and date. Example: date '2005-07-27'

time — Time of day, in hours, minutes and seconds. Examples: 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'.

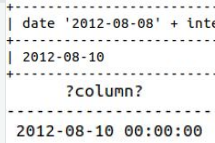
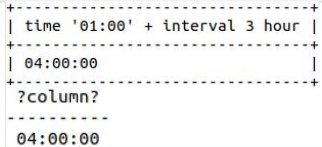
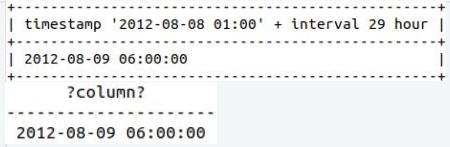
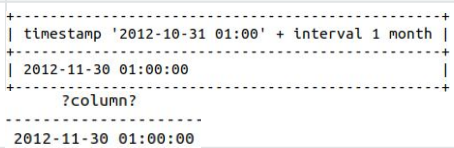
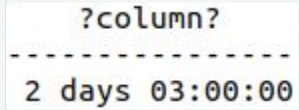
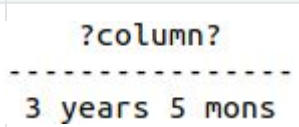
timestamp — Date plus time of day (**inserted automatically in MariaDB, like datetime in Postgresql**). Example: timestamp '2005-7-27 09:00:30.75'

interval — Period of time (year, month, day, hour, minute, second). Example: interval 2 day. Example: interval '2 month'.

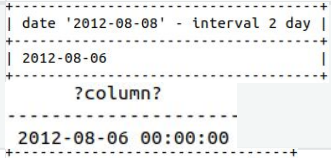
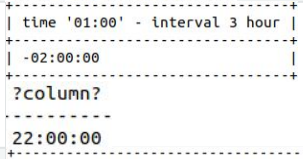
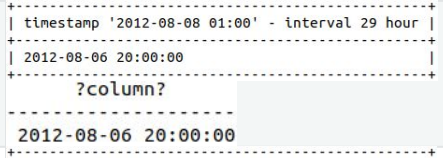
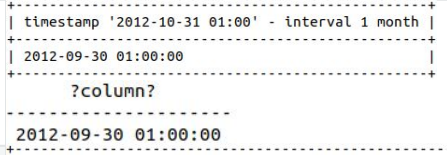
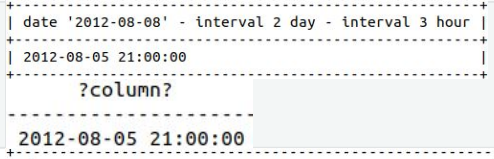
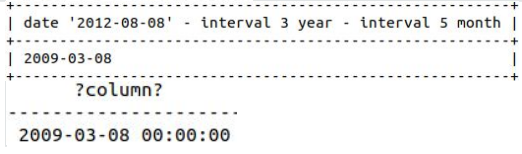
Subtracting a **date/time/timestamp** value from another gives an **interval** value.

Interval values can be added to **date/time/timestamp** values (a date will be returned).

7.6. Date functions and operators.

Operator	Example (MariaDB // PostgreSQL)	Result [source]
+	date '2012-08-08' + interval 2 day // date '2012-08-08' + interval '2 day'	
+	time '01:00' + interval 3 hour // time '01:00' + interval '3 hour'	
+	timestamp '2012-08-08 01:00' + interval 29 hour // timestamp '2012-08-08 01:00' + interval '29 hour'	
+	timestamp '2012-10-31 01:00' + interval 1 month // timestamp '2012-10-31 01:00' + interval '1 month'	
+	~~interval 2 day + interval 3 hour~~ // interval '2 day' + interval '3 hour'	
+	~~interval 3 year + interval 5 month~~ // interval '3 year' + interval '5 month'	

7.6. Date functions and operators.

Operator	Example (MariaDB // PostgreSQL)	Result [source]
-	date '2012-08-08' - interval 2 day // date '2012-08-08' - interval '2 day'	
-	time '01:00' - interval 3 hour // time '01:00' - interval '3 hour'	
-	timestamp '2012-08-08 01:00' - interval 29 hour // timestamp '2012-08-08 01:00' - interval '29 hour'	
-	timestamp '2012-10-31 01:00' - interval 1 month // timestamp '2012-10-31 01:00' - interval '1 month'	
-	date '2012-08-08' - interval 2 day - interval 3 hour // date '2012-08-08' - interval '2 day' - interval '3 hour'	
-	date '2012-08-08' - interval 3 year - interval 5 month // date '2012-08-08' - interval '3 year' - interval '5 month'	

7.6. Date functions and operators.

OVERLAPS operator.

```
postgres=# SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
 overlaps
----------
 t
(1 row)

postgres=# SELECT (DATE '2001-02-16', INTERVAL '100 days') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
 overlaps
----------
 f
(1 row)

postgres=# SELECT (DATE '2001-10-29', DATE '2001-10-30') OVERLAPS
       (DATE '2001-10-30', DATE '2001-10-31');
 overlaps
----------
 f
(1 row)

postgres=# SELECT (DATE '2001-10-30', DATE '2001-10-30') OVERLAPS
       (DATE '2001-10-30', DATE '2001-10-31');
 overlaps
----------
 t
(1 row)
```

PostgreSQL

Source:
https://www.postgresql.org/docs/current/functions-datetime.html

7.6. Date functions and operators.

CASTING to date.

```
samplecompany=# select '2023-02-22'::date;
    date
------------
 2023-02-22
(1 row)

samplecompany=# select CAST('2023-02-22' AS date);
    date
------------
 2023-02-22
(1 row)

samplecompany=# SELECT TO_DATE('2023-02-22', 'YYYY-MM-DD');
  to_date
------------
 2023-02-22
(1 row)
```

PostgreSQL

```
MariaDB [samplecompany]> SELECT STR_TO_DATE('2023-02-22', '%Y-%m-%e');
+---------------------------------------+
| STR_TO_DATE('2023-02-22', '%Y-%m-%e') |
+---------------------------------------+
| 2023-02-22                            |
+---------------------------------------+
1 row in set (0,000 sec)
```

MariaDB

7.6. Date functions and operators.

`age ( timestamp, timestamp ) → interval`

Subtract arguments, producing a "symbolic" result that uses years and months, rather than just days

`age(timestamp '2001-04-10', timestamp '1957-06-13') → 43 years 9 mons 27 days`

`age ( timestamp ) → interval`

Subtract argument from `current_date` (at midnight)

`age(timestamp '1957-06-13') → 62 years 6 mons 10 days`

`clock_timestamp () → timestamp with time zone`

Current date and time (changes during statement execution); see Section 9.9.5

`clock_timestamp() → 2019-12-23 14:39:53.662522-05`

`current_date → date`

Current date; see Section 9.9.5

`current_date → 2019-12-23`

`current_time → time with time zone`

Current time of day; see Section 9.9.5

`current_time → 14:39:53.662522-05`

PostgreSQL

Source:
https://www.postgresql.org/docs/current/functions-datetime.html

7.6. Date functions and operators.

`current_time ( integer ) → time with time zone`

Current time of day, with limited precision; see Section 9.9.5

`current_time(2) → 14:39:53.66-05`

`current_timestamp → timestamp with time zone`

Current date and time (start of current transaction); see Section 9.9.5

`current_timestamp → 2019-12-23 14:39:53.662522-05`

`current_timestamp ( integer ) → timestamp with time zone`

Current date and time (start of current transaction), with limited precision; see Section 9.9.5

`current_timestamp(0) → 2019-12-23 14:39:53-05`

`date_bin ( interval, timestamp, timestamp ) → timestamp`

Bin input into specified interval aligned with specified origin; see Section 9.9.3

`date_bin('15 minutes', timestamp '2001-02-16 20:38:40', timestamp '2001-02-16 20:05:00') → 2001-02-16 20:35:00`

`date_part ( text, timestamp ) → double precision`

Get timestamp subfield (equivalent to `extract`); see Section 9.9.1

`date_part('hour', timestamp '2001-02-16 20:38:40') → 20`

PostgreSQL

Source:
https://www.postgresql.org/docs/current/functions-datetime.html

7.6. Date functions and operators.

`date_part ( text, interval ) → double precision`
Get interval subfield (equivalent to `extract`); see Section 9.9.1
`date_part('month', interval '2 years 3 months') → 3`

`date_trunc ( text, timestamp ) → timestamp`
Truncate to specified precision; see Section 9.9.2
`date_trunc('hour', timestamp '2001-02-16 20:38:40') → 2001-02-16 20:00:00`

`date_trunc ( text, timestamp with time zone, text ) → timestamp with time zone`
Truncate to specified precision in the specified time zone; see Section 9.9.2
`date_trunc('day', timestamptz '2001-02-16 20:38:40+00', 'Australia/Sydney') → 2001-02-16 13:00:00+00`

`date_trunc ( text, interval ) → interval`
Truncate to specified precision; see Section 9.9.2
`date_trunc('hour', interval '2 days 3 hours 40 minutes') → 2 days 03:00:00`

`extract ( field from timestamp ) → numeric`
Get timestamp subfield; see Section 9.9.1
`extract(hour from timestamp '2001-02-16 20:38:40') → 20`

PostgreSQL

Source:
https://www.postgresql.org/docs/current/functions-datetime.html

7.6. Date functions and operators.

Source:
https://www.postgresql.org/docs/current/functions-datetime.html

`extract ( field from interval ) → numeric`

Get interval subfield; see Section 9.9.1

`extract(month from interval '2 years 3 months') → 3`

`isfinite ( date ) → boolean`

Test for finite date (not +/-infinity)

`isfinite(date '2001-02-16') → true`

`isfinite ( timestamp ) → boolean`

Test for finite timestamp (not +/-infinity)

`isfinite(timestamp 'infinity') → false`

`isfinite ( interval ) → boolean`

Test for finite interval (currently always true)

`isfinite(interval '4 hours') → true`

`justify_days ( interval ) → interval`

Adjust interval so 30-day time periods are represented as months

`justify_days(interval '35 days') → 1 mon 5 days`

PostgreSQL

7.6. Date functions and operators.

`justify_hours ( interval ) → interval`

Adjust interval so 24-hour time periods are represented as days

`justify_hours(interval '27 hours') → 1 day 03:00:00`

`justify_interval ( interval ) → interval`

Adjust interval using `justify_days` and `justify_hours`, with additional sign adjustments

`justify_interval(interval '1 mon -1 hour') → 29 days 23:00:00`

`localtime → time`

Current time of day; see Section 9.9.5

`localtime → 14:39:53.662522`

`localtime ( integer ) → time`

Current time of day, with limited precision; see Section 9.9.5

`localtime(0) → 14:39:53`

`localtimestamp → timestamp`

Current date and time (start of current transaction); see Section 9.9.5

`localtimestamp → 2019-12-23 14:39:53.662522`

PostgreSQL

Source:
https://www.postgresql.org/docs/current/functions-datetime.html

7.6. Date functions and operators.

`localtimestamp ( integer ) → timestamp`

Current date and time (start of current transaction), with limited precision; see Section 9.9.5

`localtimestamp(2) → 2019-12-23 14:39:53.66`

`make_date ( year int, month int, day int ) → date`

Create date from year, month and day fields (negative years signify BC)

`make_date(2013, 7, 15) → 2013-07-15`

`make_interval ( [ years int [, months int [, weeks int [, days int [, hours int [, mins int [, secs double precision ]]]]]] ] ) → interval`

Create interval from years, months, weeks, days, hours, minutes and seconds fields, each of which can default to zero

`make_interval(days => 10) → 10 days`

`make_time ( hour int, min int, sec double precision ) → time`

Create time from hour, minute and seconds fields

`make_time(8, 15, 23.5) → 08:15:23.5`

`make_timestamp ( year int, month int, day int, hour int, min int, sec double precision ) → timestamp`

Create timestamp from year, month, day, hour, minute and seconds fields (negative years signify BC)

`make_timestamp(2013, 7, 15, 8, 15, 23.5) → 2013-07-15 08:15:23.5`

PostgreSQL

Source:
https://www.postgresql.org/docs/current/functions-datetime.html

7.6. Date functions and operators.

`make_timestamptz(`*`year`* `int,` *`month`* `int,` *`day`* `int,` *`hour`* `int,` *`min`* `int,` *`sec`* `double precision [,` *`timezone`* `text ]) → timestamp with time zone`

Create timestamp with time zone from year, month, day, hour, minute and seconds fields (negative years signify BC). If *`timezone`* is not specified, the current time zone is used; the examples assume the session time zone is `Europe/London`

```
make_timestamptz(2013, 7, 15, 8, 15, 23.5) → 2013-07-15 08:15:23.5+01
make_timestamptz(2013, 7, 15, 8, 15, 23.5, 'America/New_York') → 2013-07-15 13:15:23.5+01
```

`now() → timestamp with time zone`

Current date and time (start of current transaction); see Section 9.9.5

```
now() → 2019-12-23 14:39:53.662522-05
```

`statement_timestamp() → timestamp with time zone`

Current date and time (start of current statement); see Section 9.9.5

```
statement_timestamp() → 2019-12-23 14:39:53.662522-05
```

`timeofday() → text`

Current date and time (like `clock_timestamp`, but as a `text` string); see Section 9.9.5

```
timeofday() → Mon Dec 23 14:39:53.662522 2019 EST
```

`transaction_timestamp() → timestamp with time zone`

Current date and time (start of current transaction); see Section 9.9.5

```
transaction_timestamp() → 2019-12-23 14:39:53.662522-05
```

`to_timestamp(double precision) → timestamp with time zone`

Convert Unix epoch (seconds since 1970-01-01 00:00:00+00) to timestamp with time zone

```
to_timestamp(1284352323) → 2010-09-13 04:32:03+00
```

PostgreSQL

Source:
https://www.postgresql.org/docs/current/functions-datetime.html

7.6. Date functions and operators.

- **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)

MariaDB

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

- **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 |
+---------------------+--------------------+
```

MariaDB

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

- **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.

MariaDB

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

- **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 |
+-----------------------------+
```

MariaDB

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

- **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 |
+------------------------------------------------+
```

MariaDB

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

- **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 |
+-------------------+
```

MariaDB

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

Source: https://mariadb.com/kb/en/date-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            |
+-----------------+
```

MariaDB

7.6. Date functions and operators.

- **MICROSECOND**: Returns microseconds from a date or datetime.

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

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

```
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                |
+-------------------------+
```

MariaDB

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

Source: https://mariadb.com/kb/en/date-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 |
+---------------------+
```

MariaDB

7.6. Date functions and operators.

- **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                              |
+-----------------------------------------+
```

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

- **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.

MariaDB

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

Source: https://mariadb.com/kb/en/date-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                        |
+-----------------------------------+
```

MariaDB

7.6. Date functions and operators.

Source: https://mariadb.com/kb/en/date-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 |
+-----------------+
```

MariaDB

7.6. Date functions and operators.

- **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 |
+---------------------+-----------------------+
```

MariaDB

Source: https://mariadb.com/kb/en/date-time-functions/

7.6. Date functions and operators.

Source: https://mariadb.com/kb/en/date-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 |
+-------------------------+
```

MariaDB

7.5. String functions and operators.

References:

https://www.postgresql.org/docs/current/datatype-datetime.html
https://www.postgresql.org/docs/current/functions-datetime.html

PostgreSQL

https://mariadb.com/kb/en/date-and-time-data-types/
https://mariadb.com/kb/en/date-time-functions/

MariaDB

7.5. String functions and operators.

P07_dates

Publicat el dia 15:45

7.7. Control Flow Functions.

CASE:

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

```
SELECT * FROM test;

 a
---
 1
 2
 3


SELECT a,
       CASE WHEN a=1 THEN 'one'
            WHEN a=2 THEN 'two'
            ELSE 'other'
       END
    FROM test;

 a | case
---+-------
 1 | one
 2 | two
 3 | other
```

7.7. Control Flow Functions.

```
SELECT tidhome, tidaway,
  goalshome, goalsaway, `date`,
  CASE
    WHEN goalshome > goalsaway THEN CONCAT(tidhome, ' won ', tidaway)
    WHEN goalshome < goalsaway THEN CONCAT(tidaway, ' won ', tidhome)
    ELSE CONCAT('Draw game between ', tidhome, ' and ',tidaway)
  END AS result_commented
FROM matches
LIMIT 5;
```

```
+---------+---------+-----------+-----------+------------+------------------------------+
| tidhome | tidaway | goalshome | goalsaway | date       | result_commented             |
+---------+---------+-----------+-----------+------------+------------------------------+
| ARS     | CHE     |         0 |         0 | 2017-01-21 | Draw game between ARS and CHE |
| ARS     | CHE     |         2 |         0 | 2017-12-03 | ARS won CHE                  |
| ARS     | LIV     |         1 |         3 | 2016-11-20 | LIV won ARS                  |
| ARS     | LIV     |         0 |         0 | 2018-01-17 | Draw game between ARS and LIV |
| ARS     | MAC     |         1 |         2 | 2017-01-08 | MAC won ARS                  |
+---------+---------+-----------+-----------+------------+------------------------------+
5 rows in set (0,001 sec)
```

7.7. Control Flow Functions.

```
+--------------------+--------+------+-------+------+------+------+------+
| team               | points | won  | drawn | lost | gf   | ga   | gd   |
+--------------------+--------+------+-------+------+------+------+------+
| Tottenham Hotspur  |     36 |    9 |     9 |    2 |   35 |   21 |   14 |
| Chelsea            |     33 |    9 |     6 |    5 |   22 |   19 |    3 |
| Manchester City    |     28 |    7 |     7 |    6 |   29 |   30 |   -1 |
| Arsenal            |     24 |    6 |     6 |    8 |   17 |   19 |   -2 |
| Liverpool          |     22 |    5 |     7 |    8 |   26 |   32 |   -6 |
| Manchester United  |     17 |    4 |     5 |   11 |   21 |   29 |   -8 |
+--------------------+--------+------+-------+------+------+------+------+
6 rows in set (0,001 sec)
```

7.7. Control Flow Functions.

COALESCE: Returns the first of its arguments that is not null.

COALESCE(value [, ...])

```
MariaDB [samplecompany]> SELECT COALESCE(commission, 0)
    -> FROM employees;
+--------------------------+
| COALESCE(commission, 0) |
+--------------------------+
|                        0 |
|                        0 |
|                      390 |
|                      650 |
|                        0 |
|                     1020 |
|                        0 |
|                        0 |
|                        0 |
|                        0 |
|                        0 |
|                        0 |
|                        0 |
|                        0 |
|                        0 |
|                        0 |
|                        0 |
+--------------------------+
17 rows in set (0,001 sec)
```

7.7. Control Flow Functions.

NULLIF: Returns a null value if value1 equals value2; otherwise it returns value1.

NULLIF(value1, value2)

commission	NULLIF(commission,0)
NULL	NULL
NULL	NULL
390	390
650	650
NULL	NULL
1020	1020
NULL	NULL
NULL	NULL
NULL	NULL
0	NULL
NULL	NULL
NULL	NULL
NULL	NULL
NULL	NULL
NULL	NULL
NULL	NULL
NULL	NULL

17 rows in set (0,001 sec)

7.7. Control Flow Functions.

GREATEST: Returns the largest value from a list.

GREATEST(value [, ...])

```
MariaDB [samplecompany]> SELECT salary,
    -> commission,
    -> GREATEST(salary, commission),
    -> GREATEST(12, 45, 1, 34),
    -> GREATEST('abs', 'cdf', 'hdz', 'za')
    -> FROM employees;
+--------+------------+-----------------------------+-------------------------+-----------------------------------+
| salary | commission | GREATEST(salary, commission) | GREATEST(12, 45, 1, 34) | GREATEST('abs', 'cdf', 'hdz', 'za') |
+--------+------------+-----------------------------+-------------------------+-----------------------------------+
|   1040 |       NULL |                        NULL |                      45 | za                                |
|   1040 |       NULL |                        NULL |                      45 | za                                |
|   1500 |        390 |                        1500 |                      45 | za                                |
|   1625 |        650 |                        1625 |                      45 | za                                |
|   2900 |       NULL |                        NULL |                      45 | za                                |
|   1600 |       1020 |                        1600 |                      45 | za                                |
|   3005 |       NULL |                        NULL |                      45 | za                                |
|   2885 |       NULL |                        NULL |                      45 | za                                |
|   3000 |       NULL |                        NULL |                      45 | za                                |
|   1350 |          0 |                        1350 |                      45 | za                                |
|   1430 |       NULL |                        NULL |                      45 | za                                |
|   1430 |       NULL |                        NULL |                      45 | za                                |
|   1335 |       NULL |                        NULL |                      45 | za                                |
|   3000 |       NULL |                        NULL |                      45 | za                                |
|   1690 |       NULL |                        NULL |                      45 | za                                |
|   2885 |       NULL |                        NULL |                      45 | za                                |
|   2885 |       NULL |                        NULL |                      45 | za                                |
+--------+------------+-----------------------------+-------------------------+-----------------------------------+
17 rows in set (0,001 sec)
```

7.7. Control Flow Functions.

LEAST: Returns the smallest value from a list.

LEAST(value [, ...])

```
MariaDB [samplecompany]> SELECT salary,
    -> commission,
    -> LEAST(salary, commission),
    -> LEAST(12, 45, 1, 34),
    -> LEAST('abs', 'cdf', 'hdz', 'za')
    -> FROM employees;
+--------+------------+--------------------------+---------------------+---------------------------------+
| salary | commission | LEAST(salary, commission) | LEAST(12, 45, 1, 34) | LEAST('abs', 'cdf', 'hdz', 'za') |
+--------+------------+--------------------------+---------------------+---------------------------------+
|   1040 |       NULL |                     NULL |                   1 | abs                             |
|   1040 |       NULL |                     NULL |                   1 | abs                             |
|   1500 |        390 |                      390 |                   1 | abs                             |
|   1625 |        650 |                      650 |                   1 | abs                             |
|   2900 |       NULL |                     NULL |                   1 | abs                             |
|   1600 |       1020 |                     1020 |                   1 | abs                             |
|   3005 |       NULL |                     NULL |                   1 | abs                             |
|   2885 |       NULL |                     NULL |                   1 | abs                             |
|   3000 |       NULL |                     NULL |                   1 | abs                             |
|   1350 |          0 |                        0 |                   1 | abs                             |
|   1430 |       NULL |                     NULL |                   1 | abs                             |
|   1430 |       NULL |                     NULL |                   1 | abs                             |
|   1335 |       NULL |                     NULL |                   1 | abs                             |
|   3000 |       NULL |                     NULL |                   1 | abs                             |
|   1690 |       NULL |                     NULL |                   1 | abs                             |
|   2885 |       NULL |                     NULL |                   1 | abs                             |
|   2885 |       NULL |                     NULL |                   1 | abs                             |
+--------+------------+--------------------------+---------------------+---------------------------------+
17 rows in set (0,001 sec)
```

7.7. Control Flow Functions.

Only MariaDB

IF: If expr1 is TRUE (expr1 <> 0 and expr1 <> NULL) then IF() returns expr2; otherwise it returns expr3.

IF(expr1,expr2,expr3)

MariaDB

```
MariaDB [samplecompany]> select name,
    -> IF(name='FERNANDO', 'FERRAN', name)
    -> from employees;
+-----------+--------------------------------------+
| name      | IF(name='FERNANDO', 'FERRAN', name) |
+-----------+--------------------------------------+
| BRAD      | BRAD                                 |
| SERGIO    | SERGIO                               |
| MARTA     | MARTA                                |
| REBECA    | REBECA                               |
| JUAN      | JUAN                                 |
| MONICA    | MONICA                               |
| BARTOLOME | BARTOLOME                            |
| MARIA     | MARIA                                |
| JESUS     | JESUS                                |
| LUIS      | LUIS                                 |
| FERNANDO  | FERRAN                               |
| LAURA     | LAURA                                |
| XAVIER    | XAVIER                               |
| ANA       | ANA                                  |
| ANTONIA   | ANTONIA                              |
| ANTONIO   | ANTONIO                              |
| FERNANDA  | FERNANDA                             |
+-----------+--------------------------------------+
17 rows in set (0,001 sec)
```

7.8. User-Defined Types.

Create type construct in SQL creates user-defined type.

```
CREATE TYPE employee_type AS (
    employee_id integer,
    employee_name varchar(50),
    employee_salary numeric(10,2)
);
```

```
CREATE TABLE employees (
    id serial primary key,
    employee_info employee_type
);
INSERT INTO employees (employee_info)
VALUES ((1, 'Sergi', 1000));
```

```
test=# INSERT INTO
  employees (employee_info)
VALUES
  ((1, 'Sergi', 1000));
INSERT 0 1
test=# select * from employees;
 id |    employee_info
----+--------------------
  1 | (1,Sergi,1000.00)
(1 row)
```

```
CREATE FUNCTION get_employee_info(employee employee_type) RETURNS text AS $$
BEGIN
    RETURN employee.employee_name || ' makes ' || employee.employee_salary;
END;
$$ LANGUAGE plpgsql;
```

7.8. User-Defined Types.

Create type also supports the creation of user-defined enumerated types (also known as "**enums**").

```
CREATE TYPE day_of_week AS ENUM (
    'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'
);
```

```
CREATE TABLE meetings (
    id serial primary key,
    day day_of_week,
    start_time time,
    end_time time,
    topic varchar(100)
);
INSERT INTO meetings (day, start_time, end_time, topic)
VALUES ('Monday', '10:00', '11:00', 'Weekly Staff Meeting');
```

7.8. User-Defined Types.

```
test=# \dT
         List of data types
 Schema |     Name      | Description
--------+---------------+-------------
 public | day_of_week   |
 public | employee_type |
(2 rows)

test=# \dT+
                                    List of data types
 Schema |     Name      | Internal name | Size  | Elements  |  Owner   | Access privileges | Description
--------+---------------+---------------+-------+-----------+----------+-------------------+-------------
 public | day_of_week   | day_of_week   | 4     | Monday   +| postgres |                   |
        |               |               |       | Tuesday  +|          |                   |
        |               |               |       | Wednesday+|          |                   |
        |               |               |       | Thursday +|          |                   |
        |               |               |       | Friday   +|          |                   |
        |               |               |       | Saturday +|          |                   |
        |               |               |       | Sunday    |          |                   |
 public | employee_type | employee_type | tuple |           | postgres |                   |
(2 rows)
```

7.8. User-Defined Types.

ENUM types in MariaDB:

Doc: https://mariadb.com/kb/en/enum/

```
ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]
```

```
CREATE TABLE fruits (
  id INT NOT NULL auto_increment PRIMARY KEY,
  fruit ENUM('apple','orange','pear'),
  bushels INT);
```

7.9. Domains.

Domains in PostgreSQL:

Video
Creació de dominis en PostgreSQL (Miquel Boada Artigas).
Doc:
https://www.postgresql.org/docs/current/sql-createdomain.html

Domains are used to specify valid values for a column tables. The difference between a domain and an enum is that with domains we can check using a range of values, a regular expression…

```
CREATE DOMAIN name [ AS ] data_type
    [ COLLATE collation ]
    [ DEFAULT expression ]
    [ constraint [ ... ] ]

where constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK ( expression) }
```

```
CREATE DOMAIN postal_code_IB AS TEXT
CHECK(
   VALUE ~ '^07\d{3}$'
);
```

7.9. Domains.

Domains in PostgreSQL:

```
CREATE DOMAIN postal_code_IB AS TEXT
CHECK(
   VALUE ~ '^07\d{3}$'
);
```

```
create table customers (
  ID serial primary key,
  fullname varchar(150),
  address varchar(250),
  postal_code postal_code_IB
);
```

```
postgres=# insert into customers (fullname, address, postal_code) values
('Sergi González', 'Caracas Street, 6', '07006');
INSERT 0 1
postgres=# insert into customers (fullname, address, postal_code) values
('Sergi González', 'Caracas Street, 6', '08006');
ERROR:  value for domain postal_code_ib violates check constraint "postal_code_ib_chec
```

7.9. Domains.

```
CREATE DOMAIN email_domain AS TEXT
CHECK(
   VALUE ~ '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'
);

CREATE TABLE employees (
    id serial PRIMARY KEY,
    name VARCHAR(50),
    email email_domain
);
```

Source of the regular expression:
https://stackoverflow.com/questions/10164758/postgresql-regex-to-validate-email-addresses

7.9. Domains.

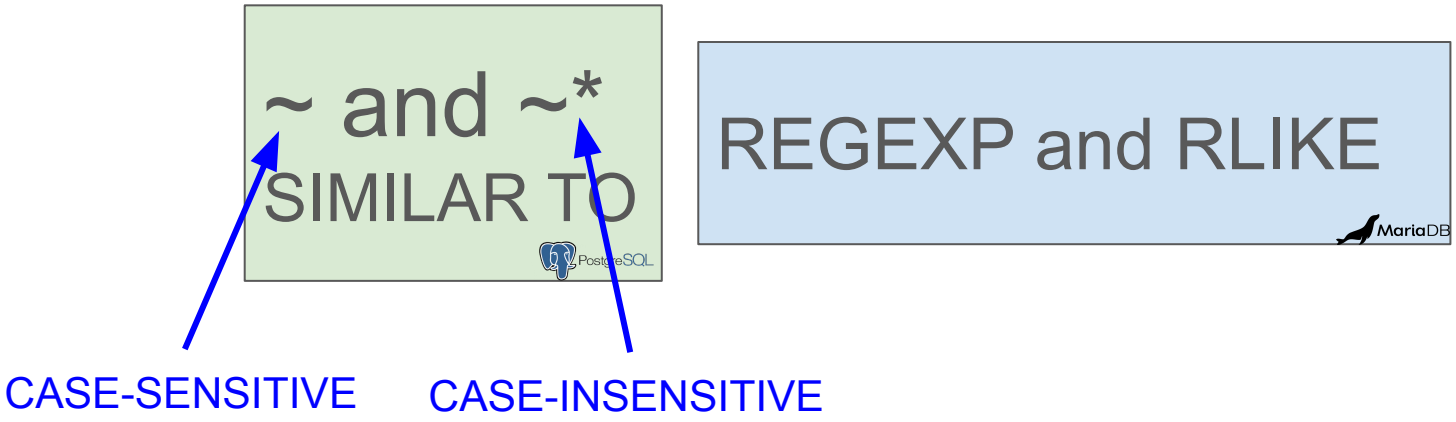
To change or remove a domain, you use the **ALTER DOMAIN** or **DROP DOMAIN** respectively.

To view all domains in the current database, you use the \dD command as follows:

```
postgres=# \dD
                                                 List of domains
 Schema |      Name      | Type | Collation | Nullable | Default |                                  Check
--------+----------------+------+-----------+----------+---------+-------------------------------------------------------------------------
 public | email_domain   | text |           |          |         | CHECK (VALUE ~ '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'::text)
 public | postal_code_ib | text |           |          |         | CHECK (VALUE ~ '^07\d{3}$'::text)
(2 rows)
```

7.10. Regular expressions.

Regular expressions are a powerful tool for **pattern matching** and text search in databases.



7.10. Regular expressions.

```
CREATE TABLE test (
  id integer PRIMARY KEY,
  name text
);

INSERT INTO
  test
VALUES
  (1, 'ABC'),
  (2, 'abc'),
  (3, 'ABc'),
  (4, '111');
```

```
MariaDB [test]> SELECT
    ->   *
    -> FROM
    ->   test
    -> WHERE
    ->   name REGEXP '^[A-Za-z]+$';
+----+------+
| id | name |
+----+------+
|  1 | ABC  |
|  2 | abc  |
|  3 | ABc  |
+----+------+
3 rows in set (0,000 sec)

MariaDB [test]> SELECT
    ->   *
    -> FROM
    ->   test
    -> WHERE
    ->   name REGEXP '^[0-9]+$';
+----+------+
| id | name |
+----+------+
|  4 | 111  |
+----+------+
1 row in set (0,000 sec)
```

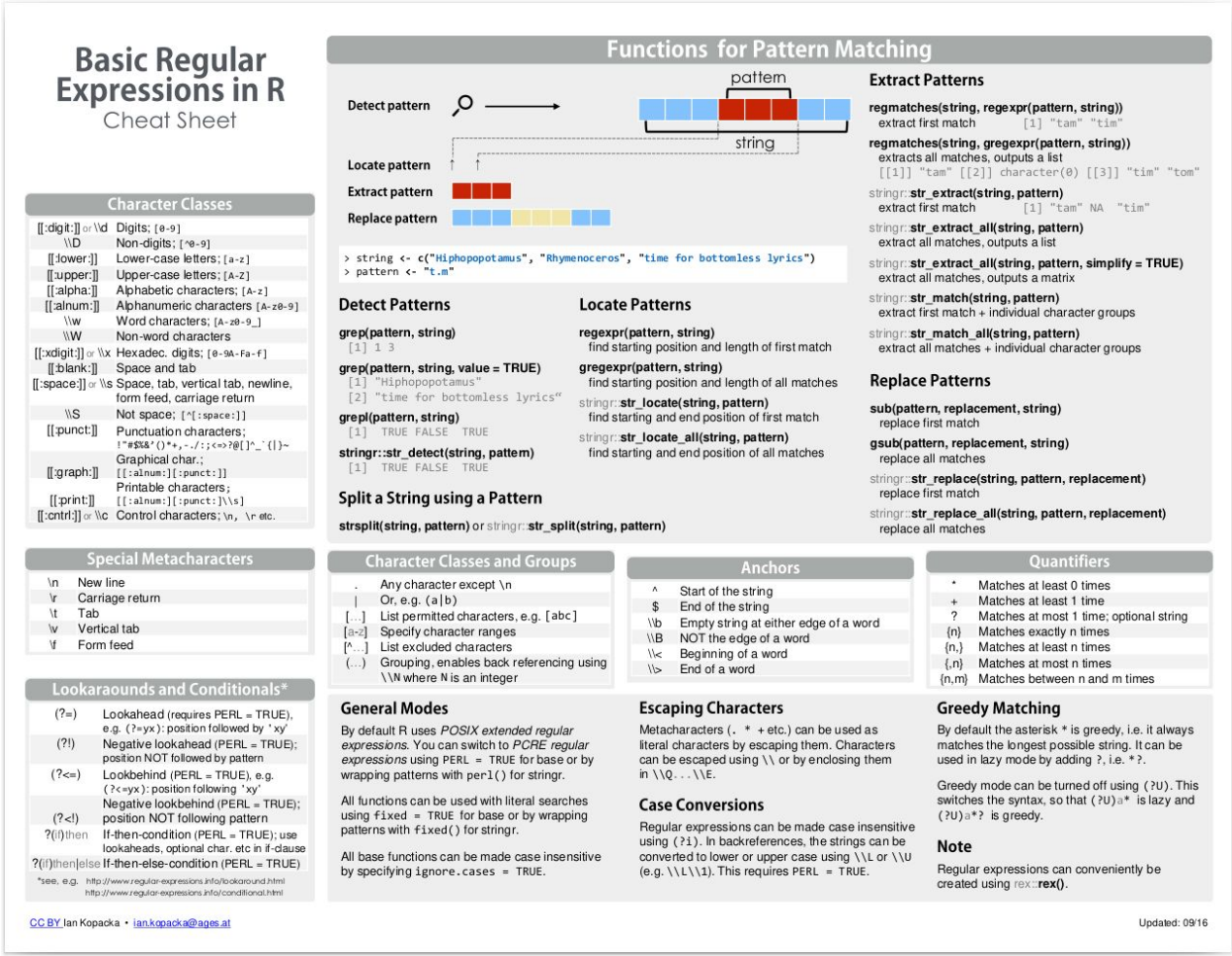
MariaDB

```
test=# SELECT
  *
FROM
  test
WHERE
  name ~ '^[A-Za-z]+$';
 id | name
----+------
  1 | ABC
  2 | abc
  3 | ABc
(3 rows)

test=# SELECT
  *
FROM
  test
WHERE
  name ~ '^[0-9]+$';
 id | name
----+------
  4 | 111
(1 row)
```

PostgreSQL

7.10. Regular expressions.



Basic Regular Expressions in R
Cheat Sheet

Character Classes

[[:digit:]] or \\d	Digits; `[0-9]`
\\D	Non-digits; `[^0-9]`
[[:lower:]]	Lower-case letters; `[a-z]`
[[:upper:]]	Upper-case letters; `[A-Z]`
[[:alpha:]]	Alphabetic characters; `[A-z]`
[[:alnum:]]	Alphanumeric characters `[A-z0-9]`
\\w	Word characters; `[A-z0-9_]`
\\W	Non-word characters
[[:xdigit:]] or \\x	Hexadec. digits; `[0-9A-Fa-f]`
[[:blank:]]	Space and tab
[[:space:]] or \\s	Space, tab, vertical tab, newline, form feed, carriage return
\\S	Not space; `[^[:space:]]`
[[:punct:]]	Punctuation characters; `!"#$%&'()*+,-./:;<=>?@[]^_`{|}~`
[[:graph:]]	Graphical char.; `[[:alnum:][:punct:]]`
[[:print:]]	Printable characters; `[[:alnum:][:punct:]\\s]`
[[:cntrl:]] or \\c	Control characters; `\n`, `\r` etc.

Special Metacharacters

\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed

Lookaraounds and Conditionals*

(?=)	Lookahead (requires PERL = TRUE), e.g. `(?=yx)`: position followed by 'xy'
(?!)	Negative lookahead (PERL = TRUE); position NOT followed by pattern
(?<=)	Lookbehind (PERL = TRUE), e.g. `(?<=yx)`: position following 'xy'
(?<!)	Negative lookbehind (PERL = TRUE); position NOT following pattern
?(if)then	If-then-condition (PERL = TRUE); use lookaheads, optional char. etc in if-clause
?(if)then|else	If-then-else-condition (PERL = TRUE)

*see, e.g. http://www.regular-expressions.info/lookaround.html
http://www.regular-expressions.info/conditional.html

Functions for Pattern Matching


```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

Detect Patterns

grep(pattern, string)
`[1] 1 3`

grep(pattern, string, value = TRUE)
`[1] "Hiphopopotamus"`
`[2] "time for bottomless lyrics"`

grepl(pattern, string)
`[1] TRUE FALSE TRUE`

stringr::str_detect(string, pattern)
`[1] TRUE FALSE TRUE`

Locate Patterns

regexpr(pattern, string)
find starting position and length of first match

gregexpr(pattern, string)
find starting position and length of all matches

stringr::**str_locate(string, pattern)**
find starting and end position of first match

stringr::**str_locate_all(string, pattern)**
find starting and end position of all matches

Split a String using a Pattern

strsplit(string, pattern) or stringr::**str_split(string, pattern)**

Extract Patterns

regmatches(string, regexpr(pattern, string))
extract first match `[1] "tam" "tim"`

regmatches(string, gregexpr(pattern, string))
extracts all matches, outputs a list
`[[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"`

stringr::**str_extract(string, pattern)**
extract first match `[1] "tam" NA "tim"`

stringr::**str_extract_all(string, pattern)**
extract all matches, outputs a list

stringr::**str_extract_all(string, pattern, simplify = TRUE)**
extract all matches, outputs a matrix

stringr::**str_match(string, pattern)**
extract first match + individual character groups

stringr::**str_match_all(string, pattern)**
extract all matches + individual character groups

Replace Patterns

sub(pattern, replacement, string)
replace first match

gsub(pattern, replacement, string)
replace all matches

stringr::**str_replace(string, pattern, replacement)**
replace first match

stringr::**str_replace_all(string, pattern, replacement)**
replace all matches

Character Classes and Groups

.	Any character except \n
|	Or, e.g. (a|b)
[...]	List permitted characters, e.g. `[abc]`
[a-z]	Specify character ranges
[^...]	List excluded characters
(...)	Grouping, enables back referencing using \\N where N is an integer

Anchors

^	Start of the string
$	End of the string
\\b	Empty string at either edge of a word
\\B	NOT the edge of a word
\\<	Beginning of a word
\\>	End of a word

Quantifiers

*	Matches at least 0 times
+	Matches at least 1 time
?	Matches at most 1 time; optional string
{n}	Matches exactly n times
{n,}	Matches at least n times
{,n}	Matches at most n times
{n,m}	Matches between n and m times

General Modes

By default R uses *POSIX extended regular expressions*. You can switch to *PCRE regular expressions* using `PERL = TRUE` for base or by wrapping patterns with `perl()` for stringr.

All functions can be used with literal searches using `fixed = TRUE` for base or by wrapping patterns with `fixed()` for stringr.

All base functions can be made case insensitive by specifying `ignore.cases = TRUE`.

Escaping Characters

Metacharacters (`. * +` etc.) can be used as literal characters by escaping them. Characters can be escaped using \\ or by enclosing them in \\Q...\\E.

Case Conversions

Regular expressions can be made case insensitive using `(?i)`. In backreferences, the strings can be converted to lower or upper case using \\L or \\U (e.g. \\L\\1). This requires `PERL = TRUE`.

Greedy Matching

By default the asterisk * is greedy, i.e. it always matches the longest possible string. It can be used in lazy mode by adding ?, i.e. `*?`.

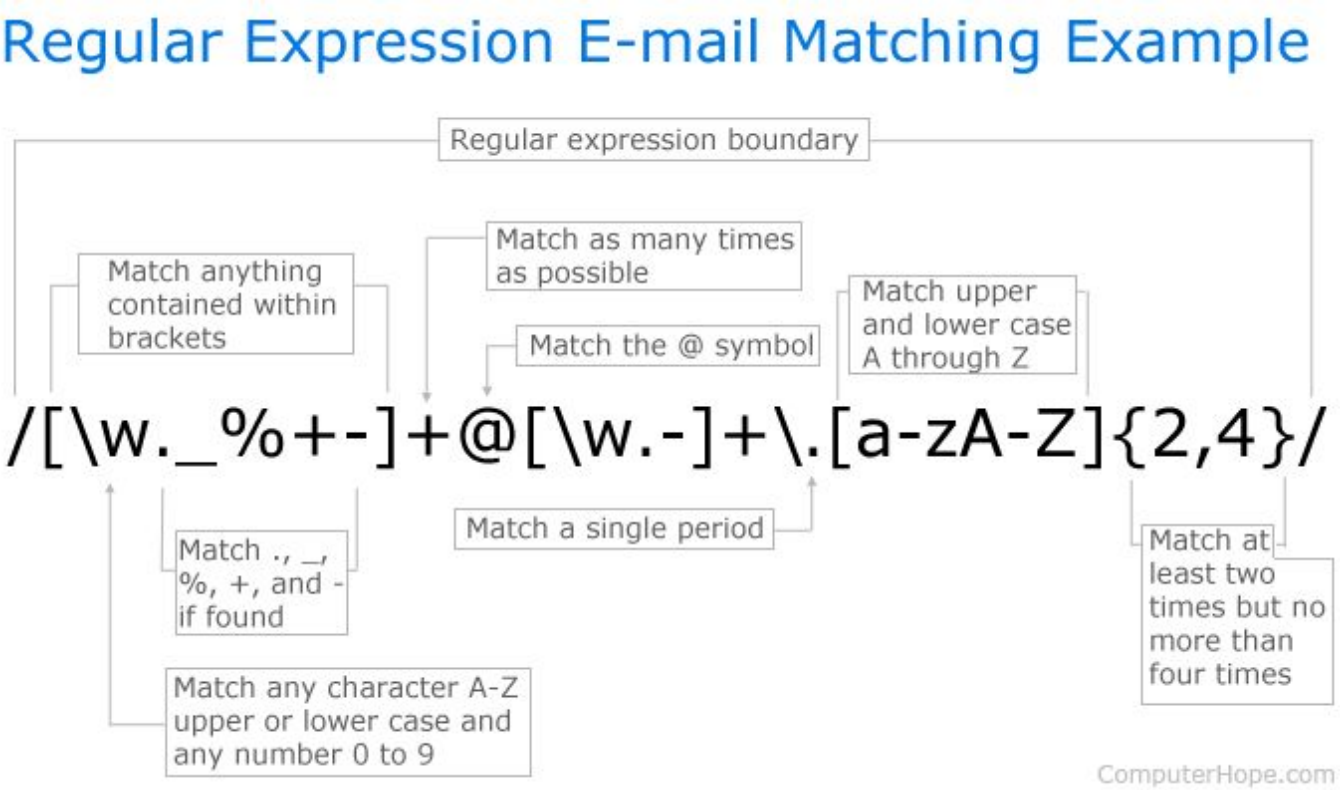
Greedy mode can be turned off using `(?U)`. This switches the syntax, so that `(?U)a*` is lazy and `(?U)a*?` is greedy.

Note

Regular expressions can conveniently be created using rex::**rex()**.

Updated: 09/16

7.10. Regular expressions.



7.10. Regular expressions.

References:

https://www.postgresql.org/docs/current/functions-matching.html

https://mariadb.com/kb/en/regular-expressions-overview/

7.11. Large-Object Types.

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are:

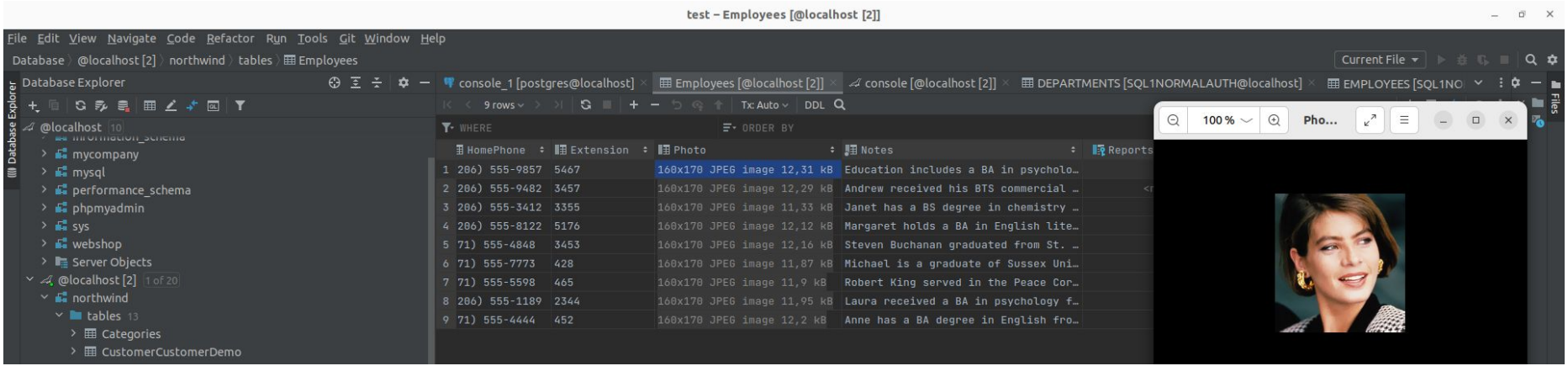
TINYBLOB	BLOB	MEDIUMBLOB	LONGBLOB

MariaDB

The table for storing **BLOB** data in PostgreSQL is called as Large Object table and the **data type** is **bytea**.

PostgreSQL

7.11. Large-Object Types.



7.12. Indexes.

Database:

TENNIS_PPKK_FFKK.sql

TENNIS_NOPK_NOFK.sql

AND

Query:

TENNIS_QUERY.sql

JOINS ARE VERY SLOW WITHOUT INDEXING.

TENNIS WITH **NO** PPKK and FFKK

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 (3 min 10,941 sec)

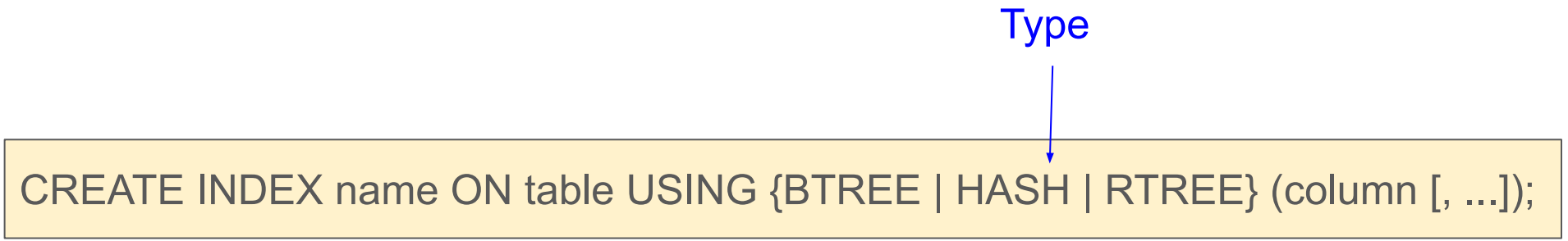
TENNIS WITH PPKK and FFKK

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 (0,076 sec)

7.12. Indexes.

Indexes are data structures used to **speed up** database **queries** by allowing faster data retrieval.



CREATE UNIQUE INDEX name ON table (column [, ...]);

7.12. Indexes.

There are different types of indexes in MariaDB and in Postgresql:

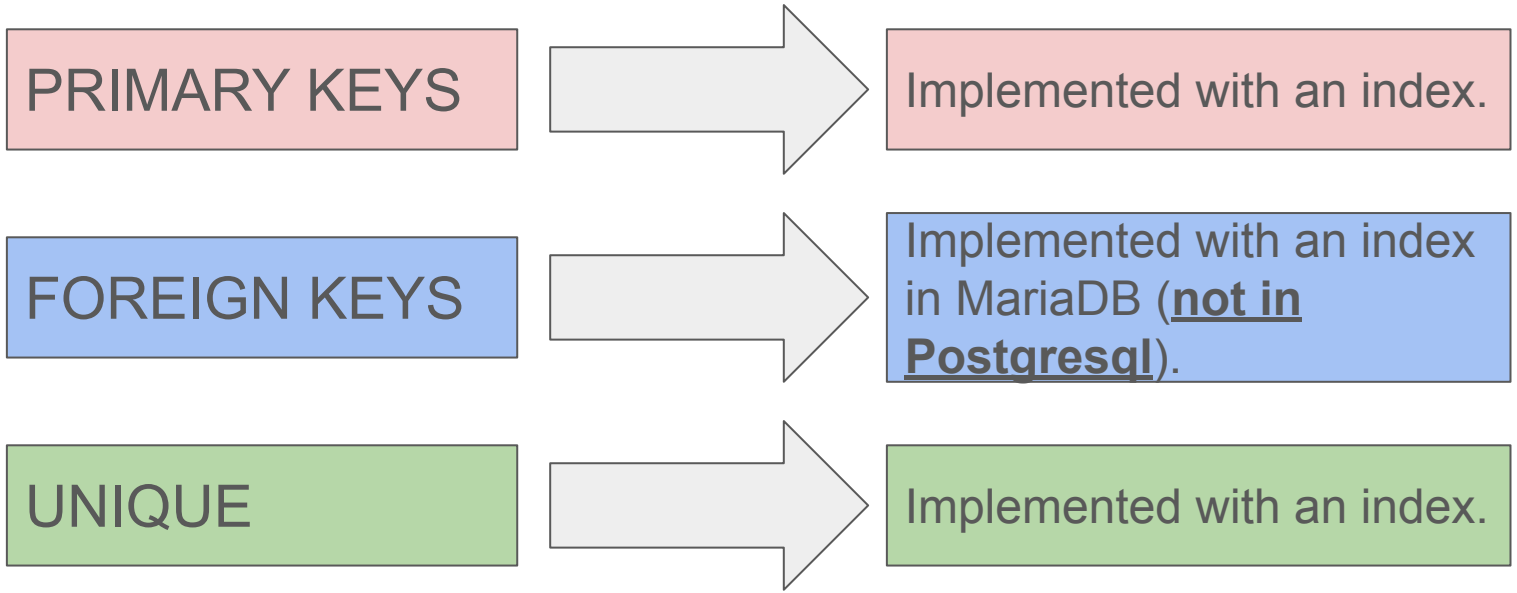
https://www.postgresql.org/docs/current/indexes-types.html

https://mariadb.com/kb/en/getting-started-with-indexes/

7.12. Indexes.

Index Type	Description	Suitable for…
B-tree	Stores data in a tree-like structure where each node contains a range of values.	…queries such as "SELECT * FROM table WHERE column BETWEEN x AND y".
Hash	Hashes the indexed column values to create a direct link to the data.	…exact-match queries such as "SELECT * FROM table WHERE column = x".
GiST	Creates a tree-like structure of the indexed data that can be searched using a variety of algorithms.	…complex data types such as geometric shapes and full-text search.
GIN	Creates an inverted index of the data, which allows for fast searches of individual words or array elements.	…searching large amounts of text data, array data types
Full-Text	Breaks text down into individual words and creates an index of those words.	…searching large amounts of text data.

7.12. Indexes.


PRIMARY KEYS
Implemented with an index.
FOREIGN KEYS
Implemented with an index in MariaDB (**not in Postgresql**).
UNIQUE
Implemented with an index.

7.12. Indexes.

In general, you should only add **indexes to match the queries your application uses** (for instance, a JOIn between table or a text column where you search frequently!).

Any extra index will waste resources. Indexing may have a performance impact on write operations, as the database have to maintain this data structure.

Using the **EXPLAIN** statement on your queries can help you decide which columns need indexing.

- https://mariadb.com/kb/en/library/explain/
- https://www.postgresql.org/docs/current/sql-explain.html

7.12. Indexes.

We said that if a query contains something like LIKE '%word%', without an index you are using a **full table scan every time**, which is very slow. More:

- https://www.postgresql.org/docs/current/textsearch.html
- https://mariadb.com/kb/en/getting-started-with-indexes/#full-text-indexes

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.

If an index is rarely used (or not used at all) then remove it to increase INSERT, and UPDATE performance.

7.12. Indexes.

If an index is rarely used (or not used at all) then remove it to increase INSERT, and UPDATE performance.

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

PostgreSQL

```
DROP INDEX [IF EXISTS] index_name ON tbl_name
    [WAIT n |NOWAIT]
```

MariaDB

7.12. Indexes.

```
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 (0,002 sec)
```

```
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 (0,001 sec)
```

7.12. Indexes.

```
applyingtocollege=# explain select S.id, S.surname, S.name, count(distinct A.college)
from students S,  applies A
where S.id = A.stid
group by S.id, S.surname, S.name
order by S.surname, S.name, S.id;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Sort  (cost=102.34..103.64 rows=520 width=128)
   Sort Key: s.surname, s.name, s.id
   ->  GroupAggregate  (cost=68.96..78.88 rows=520 width=128)
         Group Key: s.id
         ->  Sort  (cost=68.96..70.53 rows=630 width=158)
               Sort Key: s.id
               ->  Hash Join  (cost=21.70..39.66 rows=630 width=158)
                     Hash Cond: (a.stid = s.id)
                     ->  Seq Scan on applies a  (cost=0.00..16.30 rows=630 width=42)
                     ->  Hash  (cost=15.20..15.20 rows=520 width=120)
                           ->  Seq Scan on students s  (cost=0.00..15.20 rows=520 width=120)
(11 rows)
```

7.12. Indexes.

To show indexes:

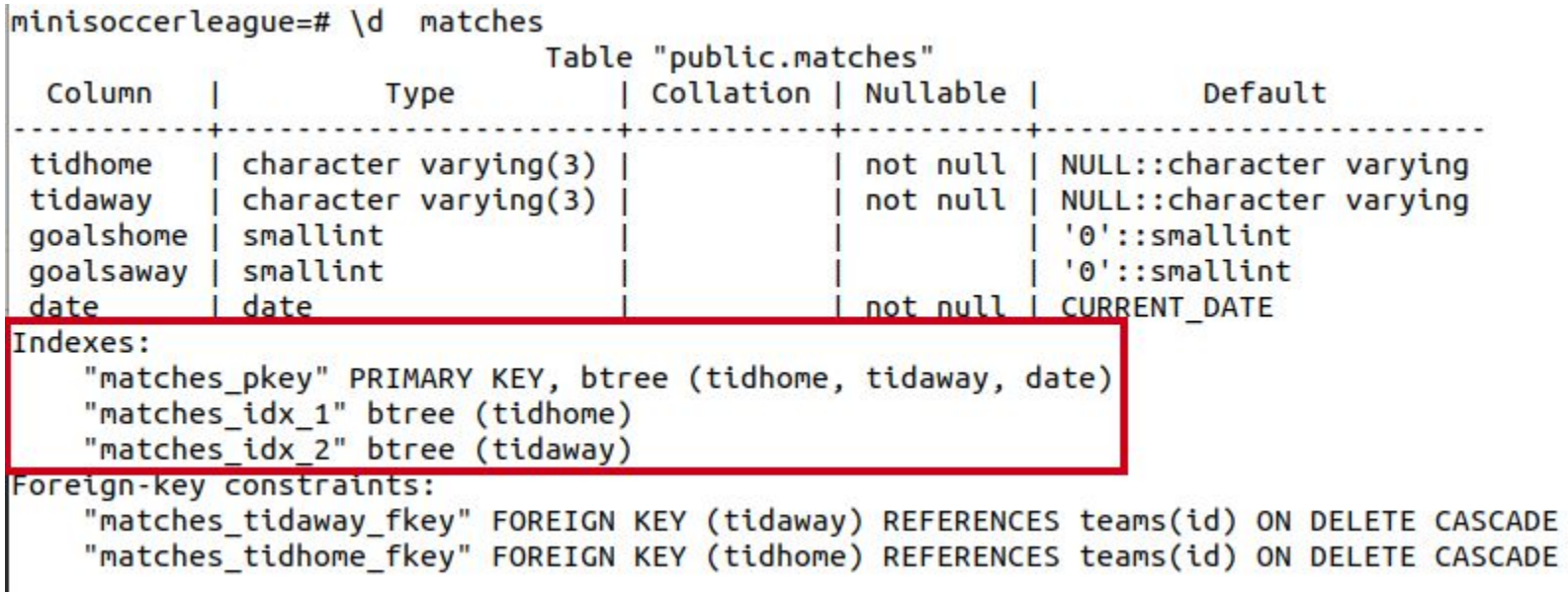
- SHOW INDEX
- SHOW CREATE TABLE

```
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 | Ignored |
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| PLAYERS |          0 | PRIMARY  |            1 | p_id        | A         |         309 |     NULL | NULL   |      | BTREE      |         |               | NO      |
| PLAYERS |          1 | p_c_id   |            1 | p_c_id      | A         |         103 |     NULL | NULL   | YES  | BTREE      |         |               | NO      |
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
2 rows in set (0,004 sec)

MariaDB [TENNIS]> show create table PLAYERS;
+---------+--------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| Table   | Create Table
                                                                                             |
+---------+--------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
| PLAYERS | CREATE TABLE `PLAYERS` (
  `p_id` int(11) NOT NULL,
  `p_name` varchar(30) NOT NULL,
  `p_surname` varchar(30) NOT NULL,
  `p_gender` char(1) NOT NULL,
  `p_c_id` varchar(3) DEFAULT NULL,
  PRIMARY KEY (`p_id`),
  KEY `p_c_id` (`p_c_id`),
  CONSTRAINT `PLAYERS_ibfk_1` FOREIGN KEY (`p_c_id`) REFERENCES `COUNTRIES` (`c_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci |
+---------+--------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------+
1 row in set (0,000 sec)
```

7.12. Indexes.

\d tablename



```
minisoccerleague=# \d  matches
                              Table "public.matches"
  Column   |         Type          | Collation | Nullable |         Default
-----------+-----------------------+-----------+----------+-------------------------
 tidhome   | character varying(3) |           | not null | NULL::character varying
 tidaway   | character varying(3) |           | not null | NULL::character varying
 goalshome | smallint              |           |          | '0'::smallint
 goalsaway | smallint              |           |          | '0'::smallint
 date      | date                  |           | not null | CURRENT_DATE
Indexes:
    "matches_pkey" PRIMARY KEY, btree (tidhome, tidaway, date)
    "matches_idx_1" btree (tidhome)
    "matches_idx_2" btree (tidaway)
Foreign-key constraints:
    "matches_tidaway_fkey" FOREIGN KEY (tidaway) REFERENCES teams(id) ON DELETE CASCADE
    "matches_tidhome_fkey" FOREIGN KEY (tidhome) REFERENCES teams(id) ON DELETE CASCADE
```

7.12. Indexes.

A JOIN WITHOUT INDEXES IN BOTH COLUMNS IS VERY SLOW!!
Thus, define PRIMARY KEYS and FOREIGN KEYS (=CREATE INDEX IN POSTGRESQL)!

7.13. Transactions.

Transaction processing is used to **maintain database integrity** by ensuring that **batches of SQL operations execute completely or not at all**.

The essential point of a transaction is that it bundles **multiple steps into** a single **all-or-nothing operation**.

The **intermediate states** between the steps are **not visible to other concurrent transactions**, and if some failure occurs that prevents the transaction from completing, then none of the steps affect the database at all.

7.13. Transactions.

Use cases:

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., etc., etc.

7.13. Transactions.

In MariaDB use storage engine **INNODB**. Other engine like MYISAM does not implement referential integrity and transactions (but it is specially good for read-intensive (select) tables.

MariaDB

7.13. Transactions.

For example, consider a **bank database** that contains **balances** for various customer accounts, as well as total deposit balances for branches. Suppose that we want to record a **payment of $100.00 from Alice**'s account **to Bob**'s account. Simplifying outrageously, the SQL commands for this might look like:

```
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
UPDATE branches SET balance = balance - 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Alice');
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
UPDATE branches SET balance = balance + 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Bob');
```

7.13. Transactions.

There are **several separate updates** involved **to accomplish** this rather simple **operation**.

All these updates happen, or none of them happen. It would certainly not do for a system failure to result in Bob receiving $100.00 that was not debited from Alice. Nor would Alice long remain a happy customer if she was debited without Bob being credited.

Grouping the updates into a transaction gives us this guarantee. A **transaction is said to be atomic**: from the point of view of other transactions, it either happens completely or not at all.

7.13. Transactions.

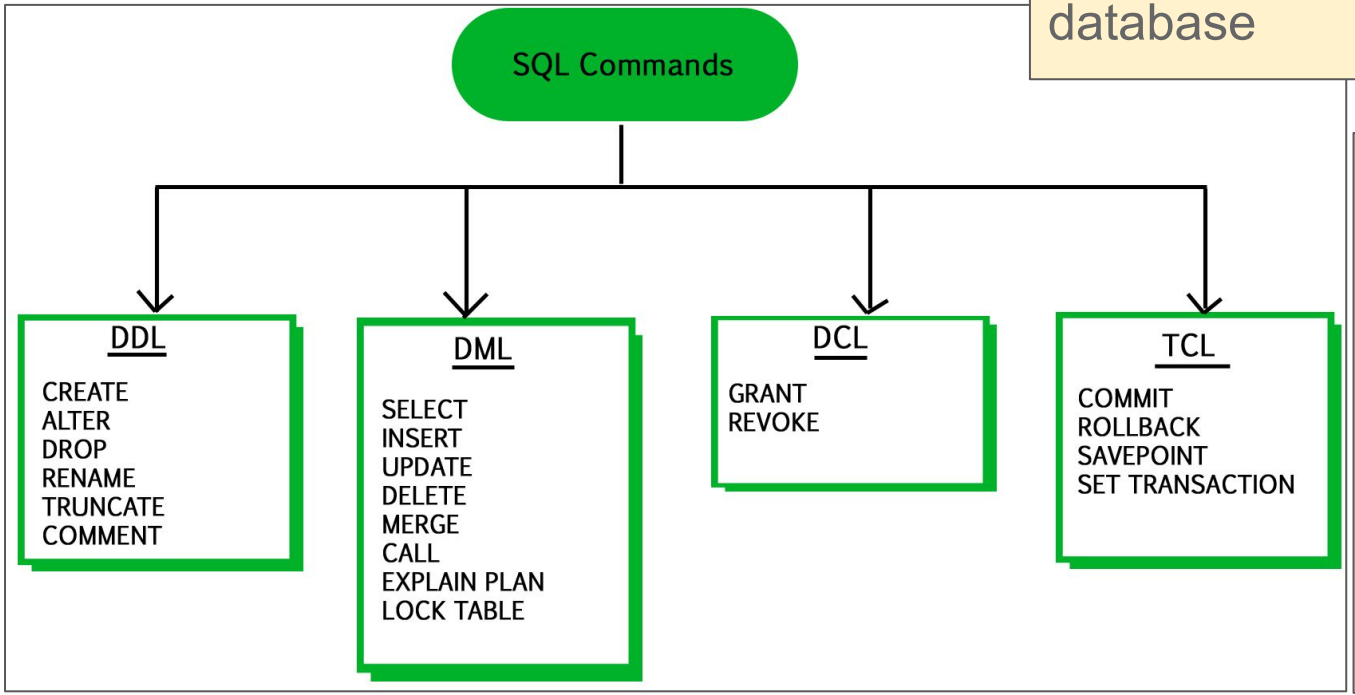
We also want a guarantee that once a transaction is completed and acknowledged by the database system, it has indeed been **permanently recorded** and won't be lost even if a crash ensues shortly thereafter. A transactional database guarantees that **all the updates made by a transaction are logged in permanent storage** (i.e., on disk) before the transaction is reported complete.

Another important property of transactional databases is closely related to the notion of **atomic updates**: when **multiple transactions are running concurrently**, each one should not be able to see the incomplete changes made by others.

An open transaction are invisible to other transactions until the transaction completes, whereupon all the updates become visible simultaneously.

7.13. Transactions.

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.

7.13. Transactions.

Transaction: A block of SQL statements executed completely or not at all.

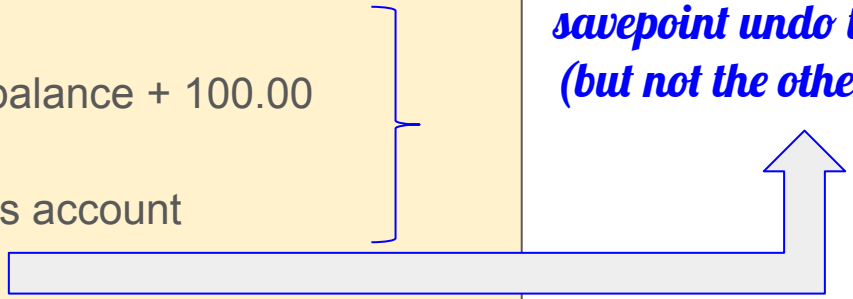
Rollback: The process of undoing specified SQL statements.

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).

7.13. Transactions.

```
BEGIN TRANSACTION; --MariaDB: START TRANSACTION
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT; -- or ROLLBACK
```



The rollback to this savepoint undo this update (but not the other ones).

MariaDB

PostgreSQL

7.13. Transactions.

Both MariaDB and PostgreSQL have the ability to **lock tables** to **prevent other sessions from modifying them while a transaction is in progress**.

```
LOCK TABLE table_name IN SHARE MODE;
UNLOCK TABLE table_name;
```

PostgreSQL

```
LOCK TABLES table_name WRITE;
UNLOCK TABLES;
```

MariaDB

7.13. Transactions.

But you'll see this next year…

More queries:

P07_BORJA_PHONE

Publicat el dia 23:08

P07_TENNIS

Publicat el dia 23:09

Sources.

- https://www.postgresql.org/docs/current/sql-dropindex.html
- https://mariadb.com/kb/en/