Ilkka Heliö 2020

Relational databases

(check wikipedia and mariadb documentation for details)

SQL

Relational databases are used with the `S`tructured `Q`uery `L`anguage, `SQL`.

(check wikipedia and mariadb documentation for details)

Using a MariaDb prompt

You can access the Mariadb/Mysql databases from a mariadb/mysql prompt.The prompt is opened by giving either the command `mariadb` or `mysql` to the terminal window (win: cmd, powerShell, linux: terminal).

When you use mariadb prompt for the first time, login as root.

```
mysql -u root
```

In Linux you might need to start mariadb as sudo

```
sudo mariadb
```

or

```
sudo mysql
```

The option -u is for the user and -p asks for the password. Here the user zeke logs in.

```
mysql -u zeke -p
Enter password:
```

Login as root. This time we use the command `mariadb`

```
> mariadb -u root -p
Enter password:

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.3.22-MariaDB MariaDB Server

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

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

MariaDB [(none)]>
```

You can write sql commands to the prompt. For example the command `show databases;` lists all databases. Remember to always end the command with a semicolon `;`.

```
MariaDB [(none)]> show databases;
+-----------------------+
| Database              |
+-----------------------+
| cardb                 |
| information_schema    |
| mysql                 |
+-----------------------+
3 rows in set (0.283 sec)

MariaDB [(none)]>
```

With the command `use` you change the active database. For example, after the command `use cardb` all operations are done to that database.

```
MariaDB [(none)]> use cardb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [cardb]> select * from car;
+--------+--------+-----------+-----------+
| carId  | model  | licence   | imagepath |
+--------+--------+-----------+-----------+
|      1 | Hoppa  | ABC-1     | hoppa.png |
|      2 | Kaara  | XYZ-123   | kaara.png |
|     23 | Vaunux | HÄK-122   | hoppa.png |
+--------+--------+-----------+-----------+
3 rows in set (0.001 sec)

MariaDB [cardb]>
```

You can close the prompt and return to the operation system with the command `exit`.

```
MariaDB [cardb]> exit
Bye
>
```

You can also redirect commands from a file. For example to create a database you can write all sql statements into a text file and redirect the input from that file to mariadb when you login to the mariadb prompt.

createStatements.sql

```
DROP DATABASE IF EXISTS employeedb;
CREATE DATABASE employeedb;
USE employeedb;
CREATE USER IF NOT EXISTS 'zeke'@'localhost' IDENTIFIED BY 'secret';
CREATE TABLE employee(
employeeId INTEGER NOT NULL PRIMARY KEY,
firstname VARCHAR(20) NOT NULL,
lastname VARCHAR(30) NOT NULL,
department VARCHAR(15),
salary DECIMAL(6,2)
);
GRANT ALL PRIVILEGES ON employeedb.* TO 'zeke'@'localhost';
INSERT INTO employee VALUES(1,'Leila','Hökki','ict',3000);
INSERT INTO employee VALUES(2,'Matt','River','admin',7000);
```

```
> mysql -u root -p < createStatements.sql
Enter password:
```

If no errors are shown, all statements were executed succesfully. If there are errors in the statements, the execution stops at the first error and prints out an error message.

```
> mysql -u root -p < createStatements.sql
Enter password:
ERROR 1064 (42000) at line 3: 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 'US employeedb' at line 1
>
```

Here we have misspelled the `USE` command. The command is missing the letter `E` at the end.

You can also paste statements to the mariadb prompt. You might need to press the enter key at the last line.

```
MariaDB [(none)]> DROP DATABASE IF EXISTS employeedb;
Query OK, 0 rows affected, 2 warnings (0.001 sec)

MariaDB [(none)]> CREATE DATABASE employeedb;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> USE employeedb;
Database changed
MariaDB [employeedb]> CREATE USER IF NOT EXISTS 'zeke'@'localhost'
IDENTIFIED BY 'secret';
Query OK, 0 rows affected, 1 warning (0.000 sec)

MariaDB [employeedb]> CREATE TABLE employee(
    -> employeeId INTEGER NOT NULL PRIMARY KEY,
    -> firstname VARCHAR(20) NOT NULL,
    -> lastname VARCHAR(30) NOT NULL,
    -> department VARCHAR(15),
    -> salary DECIMAL(6,2)
    -> );
Query OK, 0 rows affected (0.236 sec)
MariaDB [employeedb]> GRANT ALL PRIVILEGES ON employeedb.* TO
'zeke'@'localhost';
Query OK, 0 rows affected (0.000 sec)

MariaDB [employeedb]> INSERT INTO employee
VALUES(1,'Leila','Hökki','ict',3000);
MariaDB [employeedb]> INSERT INTO employee
VALUES(2,'Matt','River','admin',7000);
Query OK, 1 row affected (0.070 sec)
MariaDB [employeedb]>
```

You can test the `user` and `password` by loggin in with those. Here the user was `zeke` and the password was `secret`. If you are greeted with the prompt, then the user and password works.

```
> mysql -u zeke -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 18
Server version: 10.3.22-MariaDB MariaDB Server

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

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

MariaDB [(none)]>
```

Check the databases to which user has access rights. The `show databases` command shows only those databases which the user can access. The newly created `employeedb` should be on the list.

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| employeedb         |
| information_schema |
| test               |
+--------------------+
3 rows in set (0.001 sec)
MariaDB [(none)]>
```

If you give a wrong password, you get an error message.

```
> mysql -u saku -p
Enter password:
ERROR 1045 (28000): Access denied for user 'zeke'@'localhost' (using
password: YES)
```

To remove the user, use the command `drop user`.

```
MariaDB [enployeedb]> drop user 'zeke'@'localhost';
Query OK, 0 rows affected (0.060 sec)

MariaDB [enployeedb]>
```

A few commonly used sql commands

show databases

The command show databases lists all databases to which the user has access.

```
show databases;
```

create database

This command creates a new database. You need to be root or a user with create permissions.

Here we create an employeedb database and make it as an active database.

```
create database employeedb;
use employeedb;
```

You can define access rights to the database with the command `grant` as we see later.

drop database

The database is removed with the command `drop database`. For example to remove `testdb` use command

```
drop database testdb;
```

create user

This command creates a new user. You need to be root or a user with create permissions.

When the user is created, a password can be assigned to the user with the `identified by` option.

```
CREATE USER 'zeke'@'localhost' IDENTIFIED BY 'secret';
```

Alternatively the password may be set later with the command `set password`.

```
CREATE USER 'matt'@'localhost';
SET PASSWORD FOR 'matt'@'localhost' = PASSWORD('secr');
```

User information can be queried from the `mysql.user` table with the command `select`.

```
SELECT user,host,password FROM mysql.user;
```

The resultset includes the password in encoded form

```
select user,host,password from mysql.user;
+------------------+------------------------+-----------+
| user             | host                   | password  |
+------------------+------------------------+-----------+
...
| zeke             | localhost              | *14E...67 |
...
```

drop user

This command removes a user. You need to be root or a user with remove permissions.

```
drop user 'matt'@'localhost';
```

grant

A user needs access rights to the database. These rights are given with the command `grant`. The privileges can be narrowed down to smaller parts of the database, for example to a single table.

```
GRANT ALL PRIVILEGES ON employeedb.* TO 'zeke'@'localhost';
```

The asterisk * denotes all tables (present and those to that will be created later) in the employeedb.

The following statement gives only `select` and `insert` rights to only one table in the employeedb, namely the employee table.

```
GRANT SELECT, INSERT ON employeedb.employee TO 'matt'@'localhost';
```

The clause `with grant option` gives the user rights to give access rights to other users.

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'adminpass';
GRANT ALL PRIVILEGES ON *.* TO 'admin'@'localhost' WITH GRANT OPTION;
```

Here the `*.*` denotes all databases and all tables in those databases (present and future)

show grants for

This command shows the users privileges.

```
SHOW GRANTS FOR 'matt'@'localhost';
```

outputs

Grants for matt@localhost
GRANT USAGE ON . TO 'matt'@'localhost' IDENTIFIED BY PASSWORD '*D12D4CE6E60E502E5C260524CA919A1F7AC03A39'
GRANT SELECT, INSERT ON `employeedb`.`employee` TO 'matt'@'localhost'

revoke...on

Access rights can be revoked with the command `revoke`. The rights to revoke is listed as a comma separated list after `revoke` and before the word `on`.

```
REVOKE INSERT ON employeedb.employee FROM 'matt'@'localhost';
```

create table

To create a table to the database use the command `create table`.

```
CREATE TABLE employee(
    employeeId INTEGER NOT NULL PRIMARY KEY,
    firstname VARCHAR(20) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    department VARCHAR(15),
    salary DECIMAL(6,2)
);
```

This table has five columns. The first column is defined as `primary key`, which means that the values in it must be unique. This column also must have a value, it can't be null. That is defined with the part `not null`. The type of value is integer.

The columns `firstname` and `lastname` are also obligatory (`not null`). Their values are variable length strings (varchar). The maximum length of the data is in parenthesis.

The last two columns are voluntary (there in no `not null` definition). The department column is also varchar as were firstname and lastname.

The salary column is of type decimal. The parameters in the parenthesis (6,2) tells us that salary has at maximum 6 numbers of which 2 denote the decimal part. So the biggest number that can be stored in that column (field) is 9999.99.

show tables

This command shows all tables in the active database.

Example

```
MariaDB [(none)]> use employeedb;
Database changed
MariaDB [employeedb]> show tables;
+----------------------+
| Tables_in_employeedb |
+----------------------+
| employee             |
+----------------------+
1 row in set (0.001 sec)

MariaDB [employeedb]>
```

describe

This command shows the stucture of the table.

The format of the command is:

```
describe tableName;
```

Example

```
 MariaDB [employeedb]> describe employee;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| employeeId | int(11)      | NO   | PRI | NULL    |       |
| firstname  | varchar(20)  | NO   |     | NULL    |       |
| lastname   | varchar(30)  | NO   |     | NULL    |       |
| department | varchar(15)  | YES  |     | NULL    |       |
| salary     | decimal(6,2) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
5 rows in set (0.276 sec)

MariaDB [employeedb]>
```

drop table

This command removes a table. Note that this also removes all data from that table. You need to be root or a user with the necessary permissions.

```
drop table employee;
```

insert

With this command you insert data to the table. The command takes a list of column names to which you insert data and an another list which consists of the values to be inserted to those columns in the same order as the column names.

If the column is marked with `not null` then the column must be given a value. You need not give a value to columns that are not marked as `not null` or have a default value.

The format of the insert statement is:

```
insert into table (listOfColumns) values (listOfValues)
```

Both the `listOfColumns` and `listOfValues` are comma separated lists.

If the `listOfColumns` is left out, values to all columns must be given in the order of which the columns are defined in the table definition.

Examples

```
insert into employee (employeeId, firstname, lastname, department,salary)
values (10,'Pirkko','Puro','hr',3000);

insert into employee (department, lastname,firstname, employeeId,salary)
values ('hr','River','Leyla',11,5000);

insert into employee values (9,'Will','Vahti','security',100.99);

insert into employee (employeeId, firstname, lastname)
values (6,'Meri','Myrskylä');

insert into employee (employeeId, firstname, lastname,department)
values (8,'Kari','Kyylä','secr');

insert into employee (employeeId, firstname, lastname, salary)
values (7,'Jesse','Meri',8000);

insert into employee values (3,'Jesse','Virtanen','ict',7000);
insert into employee values (4,'Alli','Hökki','marketing',3000);
insert into employee values (5,'Peter','Purse','finance',3000);
```

After insertions our employee table looks like this:

```
MariaDB [employeedb]> select * from employee;
+------------+-----------+----------+------------+---------+
| employeeId | firstname | lastname | department | salary  |
+------------+-----------+----------+------------+---------+
|          1 | Leila     | Hökki    | ict        | 3000.00 |
|          2 | Matt      | River    | admin      | 7000.00 |
|          3 | Jesse     | Virtanen | ict        | 7000.00 |
|          4 | Alli      | Hökki    | marketing  | 3000.00 |
|          5 | Peter     | Purse    | finance    | 3000.00 |
|          6 | Meri      | Myrskylä | NULL       |    NULL |
|          7 | Jesse     | Meri     | NULL       | 8000.00 |
|          8 | Kari      | Kyylä    | secr       |    NULL |
|          9 | Will      | Vahti    | security   |  100.99 |
|         10 | Pirkko    | Puro     | hr         | 3000.00 |
|         11 | Leyla     | River    | hr         | 5000.00 |
+------------+-----------+----------+------------+---------+
11 rows in set (0.001 sec)
```

Values may also be inserted with a subquery. Note that the values clause is missing. You can also omit the parenthesis around the subquery as is done in screenshot below.

```
create table salaryMissing(
    firstname varchar(20) not null,
    lastname varchar(30) not null
);

insert into salaryMissing (firstname,lastname) (select firstname, lastname
from employee where salary is null);
```

Screenshot:

```
MariaDB [employeedb]> create table salaryMissing(
    -> firstname varchar(20) not null,
    -> lastname varchar(30) not null
    -> );
Query OK, 0 rows affected (0.367 sec)

MariaDB [employeedb]> insert into salaryMissing (firstname,lastname) select
firstname, lastname
    -> from employee where salary is null;
Query OK, 2 rows affected (0.101 sec)
Records: 2  Duplicates: 0  Warnings: 0

MariaDB [employeedb]> select * from salaryMissing;
```

```
+-----------+-----------+
| firstname | lastname  |
+-----------+-----------+
| Meri      | Myrskylä  |
| Kari      | Kyylä     |
+-----------+-----------+
2 rows in set (0.000 sec)
```

delete

This command removes rows from the table based on the criteria in the `where` clause.

```
delete from someTable where criteria;
```

Always remember to add `where` and `criteria` to the delete statement. **Otherwise you delete all rows from the table!**

The following command removes all rows from the employee table

```
delete from employee;
```

Examples

```
delete from employee where salary is null;

delete from employee where lastname='Hökki';

delete from employee where department='admin';

delete from employee where department<>'secr' OR department is null;
```

update

The data can be changed with the command `update`.
The format of the update statement is:

```
update table set column1=value1,…,columnN=valueN where criteria
```

After the word `set` there is a comma separated list of column and value pairs. a new value is inserted to the column by insertion `columnName=value`.

In the examples the function `round` (MySql, MariaDb) will round a decimal number by normal rounding rules.

Update examples

Give a 10% raise to all salaries:

```
update employee set salary=ROUND(salary*1.1);
```

All ict workers get a raise of 5%:

```
update employee set salary=ROUND(salary*1.05) where department='ict';
```

Give a 10% raise to all ict workers. If the salary was null change it to 3000:

```
update employee set salary=(select coalesce(ROUND(salary*1.1),3000))
where department='ict';
```

Promote all workers with the `lastname` Myrskylä to admin and give them a salary raise of 50%:

```
update employee set department='admin', salary=ROUND(salary*1.5)
where lastname='Myrskylä' and firstname='Meri';
```

Change the salary to 2000 for workers in the secr department:

```
update employee set salary=2000 where department='secr';
```

select

The `select` statement fetches data from one or multiple tables. The result is also a table, usually called a resultset. This resultset can also be used as input to another sql statement

The format of a `select` statement:

```
select columnList from tableList where searchCriteria
group by columnOrComputedValue having filteringCriteria
order by sortingCriteria
```

A select statement has at minimum `select` and `from` parts if tables are involved; `where`, `group by`, `having` and `order by` are voluntary.

Between the words `Select` and `from` there must be the column names you want to be in the resultset. The column names are separated with a comma. An aterisk `*` means that you want all the columns from table/tables to be in the resultset. If you use an asterisk, the columns in the resultset are in the same order as the columns in the create statement of the table.

```
select firstname, lastname from employee;

select DISTINCT firstname from employee;
```

All tables that participate in the query are listed after the word `from`. Tables are separated by a comma.

```
select firstname, lastname `FROM` employee
```

The where defines the criteria to be met. Only those rows that conform to the criteria are added to the resultset. Each criterion in the criteria solves to a boolean value either true or false. Multiple criterion may be combined with the boolean operators `and` or `or`. The value of the `where` clause is the combined boolean value. In each criterion you can use relational operators `<`, `>`, `<=`, `>=`, `<>` or `!=`, `=`. When you compare null values use either `is null` or `is not null`.

```
select firstname, lastname from employeer WHERE lastname='Puro';
```

```
...where salary is null

...where salary is not null
```

The resultset can be grouped with the clause `group by`. When used with `group by` the function `count(*)` counts how many times different firstnames exists in the table.

```
select firstname, count(*) as amount from employee GROUP BY firstname;
```

The following statement counts how many null values there are in the salary column

```
select count(*)-count(salary) as SalaryMissing from ...
```

The clause `having` filters the groups to be included in the resultset. For example, the next statement counts how many employees have the same name and returns only the groups which have the `firstname` Matt or Jesse.

```
select firstname, count(*) as amount from employee
GROUP BY firstname HAVING firstname='Matt' or firstname='Jesse'
```

The resultset is:

```
MariaDB [employeedb]> select firstname, count(*) as amount from employee
    -> GROUP BY firstname HAVING firstname='Matt' or firstname='Jesse';
+-----------+--------+
| firstname | amount |
+-----------+--------+
| Jesse     |      2 |
| Matt      |      1 |
+-----------+--------+
2 rows in set (0.001 sec)
```

The rows in the resultset can be sorted with the clause `Order by`. The resultset may be sorted either in an ascending (asc) or a descending (desc) order. If the direction is omitted, the default sorting order is ascending.

The format of the `order by` clause is:

```
... order by columnList
```

The `columnList` is a comma separated list of column names and sorting directions. The resultset is sorted by the columns in the `columnList`. The sorting order is prioritised by the order of the columns in the `columnList`. The sorting is done first with the first column, then by the second and so on.

These examples sort the resultset first by `lastname` and persons that have the same last name are sorted after that by `firstname`.

This uses the default sorting order (asc) for both columns

```
select * from employee order by lastname, firstname;
```

This sorts both columns in an ascending order

```
select * from employee order by lastname asc, firstname asc;
```

This sorts first by `lastname` in an ascending order and then by `firstname` also in an ascending order

```
select lastname,firstname from employee where lastname='River' order by
lastname asc, firstname asc;
```

```
+----------+-----------+
| lastname | firstname |
+----------+-----------+
| River    | Leyla     |
| River    | Matt      |
+----------+-----------+
```

This sorts first by `lastname` in an ascending order and then by `firstname` in a descending order

```
select lastname,firstname from employee where lastname='River' ORDER BY
lastname asc, firstname desc;
```

```
+----------+-----------+
| lastname | firstname |
+----------+-----------+
| River    | Matt      |
| River    | Leyla     |
+----------+-----------+
```

Functions count, min, max, sum ja avg

You can query a minimum value by the function `min(column)` and a maximum value by the function `max(column)`. The sum of the values in the column can be counted with the function `sum(column)` and the avarage of the values in the column with the function `avg(column)`. The function `count(column)` counts the number of rows in the given column excluding null values. If a column name is replaced with an asterisk * the function `count(*)` returns the count of all rows including null values.

The total rowcount:

```
select count(*) as numberOfEmployees from employee;
```

```
MariaDB [employeedb]> select count(*) as numberOfEmployees from employee;
+-------------------+
| numberOfEmployees |
+-------------------+
|                11 |
+-------------------+
1 row in set (0.001 sec)
```

Count only the rows that have a salary;

```
select count(salary) as salaryOk from employee;
```

```
MariaDB [employeedb]> select count(salary) as salaryOk from employee;
+----------+
| salaryOk |
+----------+
|        9 |
+----------+
1 row in set (0.001 sec)
```

Count the rows that don't have a salary

```
select count(*) as "salary missing" from employee where salary is null;
```

```
MariaDB [employeedb]> select count(*) as "salary missing" from employee
where salary is null;
+----------------+
| salary missing |
+----------------+
|              2 |
+----------------+
1 row in set (0.001 sec)
```

Other examples

```
select min(salary) as lowest, max(salary) as highest from employee;

select sum(salary) as total, avg(salary) as average from employee;

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

in, between and like

in

The operator `in` returns true if the value is in the given set, otherwise it returns false.

For example here we select the rows that have lastname either 'River' or 'Puro'. Note: strings are case sensitive, so 'River' is not the same as 'river'.

```
select firstname, lastname from employee where lastname IN
('River','Puro');
```

Select all except 'River' and 'Puro':

```
select firstname, lastname from employee where lastname NOT IN
('River','Puro');
```

between ... and

`columnName` BETWEEN `lowerbound` AND `upperbound`

The clause is the same as `columnName>=lowerbound and columnName<=upperbound`

```
select lastname,salary from employee where salary BETWEEN 1000 AND 5000
order by salary;
```

outputs

```
+----------+-----------+---------+
| lastname | firstname | salary  |
+----------+-----------+---------+
| Hökki    | Leila     | 3000.00 |
| Hökki    | Alli      | 3000.00 |
| Purse    | Peter     | 3000.00 |
| Puro     | Pirkko    | 3000.00 |
| River    | Leyla     | 5000.00 |
+----------+-----------+---------+
5 rows in set (0.001 sec)
```

like

The `like` operator matches strings or substrings.

- % sign means an arbitrary string, can be an empty string
- underscore _ means an arbitrary character

All last names that begin with an uppercase V:

```
select lastname from employee where lastname LIKE 'V%';
```

outputs

```
+----------+
| lastname |
+----------+
| Virtanen |
| Vahti    |
+----------+
2 rows in set (0.001 sec)
```

All first names that contain e

```
select firstname from employee where firstname LIKE '%e%';
```

```
+-----------+
| firstname |
+-----------+
| Leila     |
| Jesse     |
| Peter     |
| Meri      |
| Jesse     |
| Leyla     |
+-----------+
6 rows in set (0.001 sec)
```

All last names where the first letter is something and the second letter is u and after that comes an arbitrary string:

```
select lastname from employee where lastname LIKE '_u%';
```

```
+----------+
| lastname |
+----------+
| Purse    |
| Puro     |
+----------+
2 rows in set (0.001 sec)
```

subquery

Subquerys can be used in the select statement. The resultset of the subquery is used in the main query. The subquery is in the parenthesis. In the following example the subquery fetches the minimun `salary`. This result is then used in the comparition to select `firstname` and `lastname` of the employees whose `salary` is the same as the minimum `salary` found by the subquery.

```
select firstname, lastname from employee where salary=(select min(salary)
from employee);
```

```
MariaDB [employeedb]> select firstname, lastname from employee where
salary=(select min(salary) from employee);
+-----------+----------+
| firstname | lastname |
+-----------+----------+
| Will      | Vahti    |
+-----------+----------+
1 row in set (0.056 sec)

MariaDB [employeedb]>
```

CASE

In the column list of the select query, there can be a case clause as a column.

```
 select firstname,lastname,salary, CASE WHEN salary>6000
    THEN 'top worker'
    ELSE '-' END as notes
from employee order by salary desc,lastname, firstname;
```

```
MariaDB [employeedb]> select firstname,lastname,salary, CASE WHEN
salary>6000 THEN 'top worker' ELSE '-' END as notes from employee order by
salary desc,lastname, firstname;
+-----------+-----------+---------+------------+
| firstname | lastname  | salary  | notes      |
+-----------+-----------+---------+------------+
| Jesse     | Meri      | 8000.00 | top worker |
| Matt      | River     | 7000.00 | top worker |
| Jesse     | Virtanen  | 7000.00 | top worker |
| Leyla     | River     | 5000.00 | -          |
| Alli      | Hökki     | 3000.00 | -          |
| Leila     | Hökki     | 3000.00 | -          |
| Pirkko    | Puro      | 3000.00 | -          |
| Peter     | Purse     | 3000.00 | -          |
| Will      | Vahti     |  100.99 | -          |
| Kari      | Kyylä     |    NULL | -          |
| Meri      | Myrskylä  |    NULL | -          |
+-----------+-----------+---------+------------+
11 rows in set (0.001 sec)
```

In the case clause you can have multiple When clauses:

```
select lastname,firstname,salary, case
        when salary=(select min(salary) from employee) then 'min'
        when salary=(select max(salary) from employee) then 'max'
        when salary is null then 'missing'
        else ' ' end as minmax
from employee order by lastname;
```

```
MariaDB [employeedb]> select lastname,firstname,salary, case
    -> when salary=(select min(salary) from employee) then 'min'
    -> when salary=(select max(salary) from employee) then 'max'
    -> when salary is null then 'missing'
    -> else ' ' end as minmax
    -> from employee order by lastname;

+-----------+-----------+---------+---------+
| lastname  | firstname | salary  | minmax  |
+-----------+-----------+---------+---------+
| Hökki     | Leila     | 3000.00 |         |
| Hökki     | Alli      | 3000.00 |         |
| Kyylä     | Kari      |    NULL | missing |
| Meri      | Jesse     | 8000.00 | max     |
| Myrskylä  | Meri      |    NULL | missing |
| Puro      | Pirkko    | 3000.00 |         |
| Purse     | Peter     | 3000.00 |         |
| River     | Matt      | 7000.00 |         |
| River     | Leyla     | 5000.00 |         |
| Vahti     | Will      |  100.99 | min     |
| Virtanen  | Jesse     | 7000.00 |         |
+-----------+-----------+---------+---------+
11 rows in set (0.001 sec)
```

Selecting from multiple tables

Usually databases consists of many tables. With `select` you can query also from multiple tables. Tables are conneted in the query with `foreign key`-`primary key` pairs.

Primary key

A primary key makes a row unique in the table. A primary key cannot be null. The primary key can be formed combining multiple columns. If the key consists of only one column, as it usually does, you can include the primary key clause in the column definition as we did earlier.

```
create table employee(
    employeeId integer not null PRIMARY KEY, ...
```

If the primary key consists of multiple columns, the definition must be made separately. Here the primary key is made up from columns `houseId` and `ownerId`. Together they make the row unique.

Foreign key

```
CREATE TABLE houseOwner(
    houseId INTEGER NOT NULL,
    ownerId INTEGER NOT NULL,
    PRIMARY KEY (houseId,ownerId),
    FOREIGN KEY (houseId) REFERENCES house(houseNumber),
    FOREIGN KEY (ownerId) REFERENCES owner(ownerNumber)
);
```

In this example there are also two foreign keys which link the table `houseOwner` to table `house` and table `owner`. The column `houseId` references the `house` tables' primary key `houseNumber`, and `ownerId` refers to `owner` tables' primary key `ownerNumber`.

Here are also those tables

```
CREATE TABLE house(
    houseNumber INTEGER NOT NULL PRIMARY KEY,
    address varchar(50) NOT NULL
);

CREATE TABLE owner(
    ownerNumber INTEGER NOT NULL PRIMARY KEY,
    firstname VARCHAR(20) NOT NULL,
    lastname VARCHAR(30) NOT NULL
);
```

For the next examples we create the following database:

coursedbCreateStatements.sql

```sql
drop database if exists coursedb;
create database coursedb;
drop user if exists 'coursemaster'@'localhost';
create user if not exists 'coursemaster'@'localhost' IDENTIFIED BY
'secret';
create table coursedb.course(
    courseID integer not null primary key,
    courseName varchar(45) not null,
    price integer not null,
    startDate date,
    endDate date
);

create table coursedb.student(
    studentID integer not null primary key,
    firstname VARCHAR(20) NOT NULL,
    lastname VARCHAR(30) NOT NULL
);

create table coursedb.courseParticipant (
    courseNumber integer not null,
    studentNumber integer not null,
    PRIMARY KEY (courseNumber,studentNumber),
    FOREIGN KEY (courseNumber) REFERENCES coursedb.course(courseID),
    FOREIGN KEY (studentNumber) REFERENCES coursedb.student(studentID)
);

grant ALL PRIVILEGES on coursedb.* to 'coursemaster'@'localhost';
```

```
insert into coursedb.course values(1,'Javascript',100,'2020-8-15','2020-10-
15');
insert into coursedb.course values(2,'Javascript',100,'2020-9-15','2020-11-
15');
insert into coursedb.course values(3,'Java',200,'2020-8-15','2020-10-15');
insert into coursedb.course values(4,'Python 101',50,'2020-8-25','2020-12-
19');

insert into coursedb.student values(1,'Matt','River');
insert into coursedb.student values(2,'Leila','Hökki');
insert into coursedb.student values(3,'Jesse','River');

insert into coursedb.courseParticipant (courseNumber,studentNumber)
values(1,1);
insert into coursedb.courseParticipant (courseNumber,studentNumber)
values(1,2);
insert into coursedb.courseParticipant (courseNumber,studentNumber)
values(1,3);
insert into coursedb.courseParticipant (courseNumber,studentNumber)
values(4,1);
insert into coursedb.courseParticipant (courseNumber,studentNumber)
values(4,3);
insert into coursedb.courseParticipant (courseNumber,studentNumber)
values(2,2);
```

After running the create statements, login as `coursemaster` and check the `coursedb`.

```
mysql -u coursemaster -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 11
Server version: 10.4.13-MariaDB MariaDB Server

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

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

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| coursedb           |
| information_schema |
| test               |
+--------------------+
3 rows in set (0.001 sec)

MariaDB [(none)]> use coursedb;
Reading table information for completion of table and column names
```

```
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [coursedb]> select * from course;

+----------+------------+-------+------------+------------+
| courseID | courseName | price | startDate  | endDate    |
+----------+------------+-------+------------+------------+
|        1 | Javascript |   100 | 2020-08-15 | 2020-10-15 |
|        2 | Javascript |   100 | 2020-09-15 | 2020-11-15 |
|        3 | Java       |   200 | 2020-08-15 | 2020-10-15 |
|        4 | Python 101 |    50 | 2020-08-25 | 2020-12-19 |
+----------+------------+-------+------------+------------+

4 rows in set (0.000 sec)

MariaDB [coursedb]> select * from student;
+-----------+-----------+----------+
| studentID | firstname | lastname |
+-----------+-----------+----------+
|         1 | Matt      | River    |
|         2 | Leila     | Hökki    |
|         3 | Jesse     | River    |
+-----------+-----------+----------+
3 rows in set (0.001 sec)

MariaDB [coursedb]> select * from courseParticipant;
+--------------+---------------+
| courseNumber | studentNumber |
+--------------+---------------+
|            1 |             1 |
|            1 |             2 |
|            1 |             3 |
|            2 |             2 |
|            4 |             1 |
|            4 |             3 |
+--------------+---------------+
6 rows in set (0.000 sec)
```

Linking tables in the select query

A classic example where you list all the tables participating in the query after `from`. The joining criteria is in the `where` clause along the restricting criteria.

```
SELECT firstname, lastname, courseName  FROM student, courseParticipant,
course WHERE student.studentID=courseParticipant.studentNumber AND
course.courseID=courseParticipant.courseNumber AND
student.lastname='River';
```

```
MariaDB [coursedb]> SELECT firstname, lastname, courseName  FROM student,
courseParticipant, course WHERE
student.studentID=courseParticipant.studentNumber AND
course.courseID=courseParticipant.courseNumber AND
student.lastname='River';
+-----------+----------+------------+
| firstname | lastname | courseName |
+-----------+----------+------------+
| Matt      | River    | Javascript |
| Jesse     | River    | Javascript |
| Matt      | River    | Python 101 |
| Jesse     | River    | Python 101 |
+-----------+----------+------------+
4 rows in set (0.001 sec)
```

joins

(check details and more joins from the MariaDbs documentation)

inner join or join

Links tables together. The format of join is

`inner join` tableName `on` conditionalExpression

which you write for all joins in that query. Here is the same query implemented with joins as in the previous example:

```
select firstname, lastname, courseName from student
inner join courseParticipant on
student.studentID=courseParticipant.studentNumber
inner join course on course.courseID=courseParticipant.courseNumber
where student.lastname='River';
```

```
MariaDB [coursedb]> select firstname, lastname, courseName from student
    -> inner join courseParticipant on
student.studentID=courseParticipant.studentNumber
    -> inner join course on course.courseID=courseParticipant.courseNumber
    -> where student.lastname='River';
+-----------+----------+------------+
| firstname | lastname | courseName |
+-----------+----------+------------+
| Matt      | River    | Javascript |
| Jesse     | River    | Javascript |
| Matt      | River    | Python 101 |
| Jesse     | River    | Python 101 |
+-----------+----------+------------+
4 rows in set (0.053 sec)
```

```
select courseName, studentNumber from course
inner join courseParticipant on
course.courseID=courseParticipant.courseNumber
order by courseName desc;
```

```
MariaDB [coursedb]> select courseName, studentNumber from course
    -> inner join courseParticipant on
course.courseID=courseParticipant.courseNumber
    -> order by courseName desc;
+------------+---------------+
| courseName | studentNumber |
+------------+---------------+
| Python 101 |             1 |
| Python 101 |             3 |
| Javascript |             1 |
| Javascript |             2 |
| Javascript |             3 |
| Javascript |             2 |
+------------+---------------+
6 rows in set (0.001 sec)
```

```
select courseName, startDate from course where startDate>current_date
order by startDate asc;
```

```
MariaDB [coursedb]> select courseName, startDate from course where
startDate>current_date order by startDate asc;
+------------+------------+
| courseName | startDate  |
+------------+------------+
| Javascript | 2020-08-15 |
| Java       | 2020-08-15 |
| Python 101 | 2020-08-25 |
| Javascript | 2020-09-15 |
+------------+------------+
4 rows in set (0.001 sec)
```

```
select c.courseName, count(p.studentNumber) as numberOfRegitered
from course as c
inner join courseParticipant as p on c.courseID=p.courseNumber
group by c.courseName
order by numberOfRegitered desc, c.courseName
```

```
MariaDB [coursedb]> select c.courseName, count(p.studentNumber) as
numberOfRegitered
    -> from course as c
    -> inner join courseParticipant as p on c.courseID=p.courseNumber
    -> group by c.courseName
    -> order by numberOfRegitered desc, c.courseName;
+------------+-------------------+
| courseName | numberOfRegitered |
+------------+-------------------+
| Javascript |                 4 |
| Python 101 |                 2 |
+------------+-------------------+
2 rows in set (0.001 sec)
```

left outer join, left join

Left join takes also the rows from the first table that have no counterpart in the second table. Here courses with no participants are also in the resultset. The table names are shortened with aliases: `c` for the table `course` and `p` for the table `courseParticipant`.

```
select c.courseName, count(p.studentNumber) as numberOfRegitered
from course as c
left join courseParticipant as p on c.courseID=p.courseNumber
group by c.courseName
order by numberOfRegitered desc, c.courseName
```

```
MariaDB [coursedb]> select c.courseName, count(p.studentNumber) as
numberOfRegitered
    -> from course as c
    -> left join courseParticipant as p on c.courseID=p.courseNumber
    -> group by c.courseName
    -> order by numberOfRegitered desc, c.courseName;
+------------+-------------------+
| courseName | numberOfRegitered |
+------------+-------------------+
| Javascript |                 4 |
| Python 101 |                 2 |
| Java       |                 0 |
+------------+-------------------+
3 rows in set (0.001 sec)
```

```
select courseName, coalesce(char(courseNumber),'-') as registered from
course
left join courseParticipant on courseID=courseNumber
order by courseName asc
```

Date and time

- year(date) year part of the date,
- month(date) year part of the date,
- day(date) year part of the date,
- current_date
- current_time
- current_timestamp

Dateformat is yyyy-mm-dd in quotes, for example '2010-04-23' Timeformat is hh:mm:ss for example '15:00' or '15:12:34'

To get only the day when the course begins use

```
select courseName, day(startDate) as day from course;
```

```
select year(current_date) - yearOfBirth as age from...
```

```
MariaDB [coursedb]> select courseName, day(startDate) as day from course;
+------------+------+
| courseName | day  |
+------------+------+
| Javascript |   15 |
| Javascript |   15 |
| Java       |   15 |
| Python 101 |   25 |
+------------+------+
4 rows in set (0.001 sec)
```

```
select day(startDate) as d, month(startDate) as m, year(startDate) as y
from course;
```

```
MariaDB [coursedb]> select day(startDate) as d, month(startDate) as m,
year(startDate) as y from course;
+------+------+------+
| d    | m    | y    |
+------+------+------+
|   15 |    8 | 2020 |
|   15 |    9 | 2020 |
|   15 |    8 | 2020 |
|   25 |    8 | 2020 |
+------+------+------+
4 rows in set (0.001 sec)
```

```
MariaDB [coursedb]> select day(current_date) as d, month(current_date) as
m, year(current_date) as y;
+------+------+------+
| d    | m    | y    |
+------+------+------+
|    8 |    8 | 2020 |
+------+------+------+
1 row in set (0.000 sec)
```

Coalesce

If the value of column is null it is replaced in theresultset with the given value.

```
select coalesce(max(courseID),0)+1 from course;
```

if the `course` table is empty, this returns 1, otherwise it returns the maximum `courseID`+1.

empty table returns resultset

```
MariaDB [coursedb]> select coalesce(max(courseID),0)+1 from course;
+-----------------------------+
| coalesce(max(courseID),0)+1 |
+-----------------------------+
|                           1 |
+-----------------------------+
1 row in set (0.001 sec)
```

table after creation

```
MariaDB [coursedb]> select coalesce(max(courseID),0)+1 from course;
+-----------------------------+
| coalesce(max(courseID),0)+1 |
+-----------------------------+
|                           5 |
+-----------------------------+
1 row in set (0.001 sec)
```