MariaDB Cheat Sheet

Console Interface

Description	Command
Log in as root (password is prompted)	`mysql -u root -p`
Log in as root with password `pass123` (**NOTE**: no space allowed between `-p` and the password)	`mysql -u root -ppass123`
Log in as the user `warrior` on the host `skynet`, and use database `annihilation` (password is prompted)	`mysql -h skynet -u warrior -p annihilation`
Setting the root password (after clean install)	`mysqladmin password "my new password"`

Useful SQL Commands

Remember that it is **good practice** to end all SQL statements with a semicolon (;).

Non CRUD

Task	SQL Query
List all databases	`SHOW DATABASES;`
Change active database	`USE database_name;`
Show all tables of the active database	`SHOW TABLES;`
Show table properties	`DESCRIBE table_name;`

CRUD

Legend	Description
`[...]`	words in **square brackets** are **optional**.
`{...}`	words in **curly brackets** are **mandatory**.

Create Database

`CREATE [OR REPLACE] {DATABASE | SCHEMA} [IF NOT EXISTS] db_name`

Drop Database

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

Create Table

```
CREATE [OR REPLACE] [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    ({col_name_1 column_definition_1} [, col_name_2 column_definition_2] ...,
     [constraint_definition], ...)
```

where:

- `OR REPLACE` mean that the existing table is dropped (if it existed) then created after therefore if this option is used, the table **will not exist** anymore if the statement fails. Note that this option **cannot be used** with the `IF EXISTS` option.
- `TEMPORARY` means that the table is only available to the current session. Temporary tables are dropped when the session ends.
- `column_definition` consists of the column's datatype and options such as `NULL`, `AUTO_INCREMENT`, etc. Datatypes such as `CHAR`, `VARCHAR`, `INT`, `DECIMAL`, etc
- `constraint_definition` consists of the constraints for the table such as `PRIMARY KEY`, `FOREIGN KEY` and `CHECK` constraints.

Alter Table

```
ALTER TABLE [IF EXISTS] tbl_name
    alter_specification [, alter_specification] ...
```

where:

- `alter_specification` consists of the table options for modification of columns and/or table/column constraints

Drop Table

```
DROP [TEMPORARY] TABLE [IF EXISTS] [/*COMMENT TO SAVE*/]
    tbl_name [, tbl_name] ...
```

Notes:

- Removes one or more tables. MariaDB will return an error indicating by name which non-existing tables it was **unable** to drop, but it also **drops all of the tables in the list that do exist**.
- The user must have the `DROP` privilege for **each table**.
- All table data and the table definition are removed, **as well as triggers** associated to the table

Insert Data

```
INSERT [INTO] tbl_name [(col,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
/* or */
INSERT [INTO] tbl_name
    SET col_1 = {expr | DEFAULT}, ..., col_n = {expr | DEFAULT}
/* or */
INSERT [INTO] tbl_name [(col_name,...)]
    SELECT ...
```

where:

- The `INSERT ... VALUES` and `INSERT ... SET` forms of the statement insert rows based on explicitly specified values and the `INSERT ... SELECT` form inserts rows selected from another table or tables.
- The columns list is **optional**. It specifies which values are **explicitly inserted**, and **in which order**. If this clause is not specified, all values must be explicitly specified, in the **same order** they are listed in the table definition.
- The list of value/s follow the `VALUES` or `VALUE` keyword (which are interchangeable), is **always wrapped by parenthesis** and must be listed in the **same order** as the columns list.
- `INSERT ... SET` forms of the statement are mainly used for one-row statements. Column order is not required.

Update Data

```
UPDATE table_references
    SET col1={expr1|DEFAULT} [,col2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Delete Data

```
DELETE FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Select Data

```
SELECT
    [ALL | DISTINCT | DISTINCTROW]
    [STRAIGHT_JOIN]
    select_expr [, select_expr ...]
    [ FROM table_references
      [WHERE where_condition]
      [GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH
ROLLUP]]
      [HAVING where_condition]
      [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
      [LIMIT {[offset,] row_count | row_count OFFSET offset}]
      [INTO OUTFILE | DUMPFILE 'file_name']
      [INTO var_name [, var_name]] ]
```

```
13
14 /* special select statement for returning ALL records of the table */
15 SELECT * FROM tbl_name;
```

where:

- `select_expr` can be the name of a column/s or some function.
- `INTO` clause is used to specify that the query results should be written to a file or variable

Joins

Simplistic SQL syntax:
`table_reference [, table_factor | join_table] ...`

where:

- `table_reference` consist of a `table_factor` or `join_table`. They are also part of any query part of the `SELECT`, `DELETE` and `UPDATE` statement
- `table_factor` is the table name or subquery or `table_reference`
- `join_table` is a `table_reference` with a particular type of join. The supporting joins are:

```
1     table_reference [INNER | CROSS] JOIN table_factor [join_condition]
2   | table_reference STRAIGHT_JOIN table_factor
3   | table_reference STRAIGHT_JOIN table_factor ON conditional_expr
4   | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference
  join_condition
5   | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor
```

Relational Algebra & Set Operations

Union

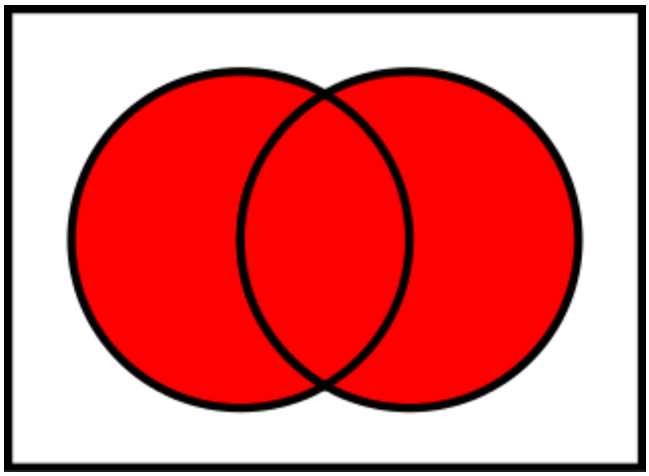

Figure 1: Venn Diagram for Union.

Relational algebra syntax: $R \cup S$

SQL syntax:

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
[ORDER BY [column [, column ...]]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Note:

- that `UNION`, `EXCEPT` and `INTERSECT` SQL commands can be used in combination.
- `UNION` queries **cannot be used with aggregate functions**.

Intersection

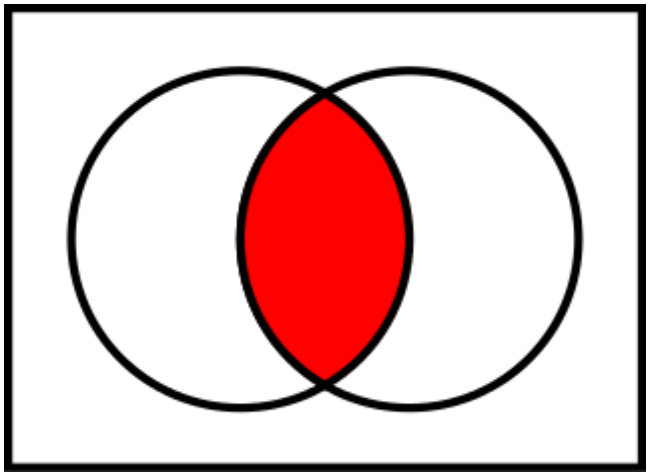

Figure 2: Venn Diagram for Intersection.

Relational algebra syntax: $R \cap S$

SQL syntax:

```
SELECT ...
(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL |
DISTINCT]) SELECT ...
[(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL |
DISTINCT]) SELECT ...]
[ORDER BY [column [, column ...]]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Note:

- that `UNION`, `EXCEPT` and `INTERSECT` SQL commands can be used in combination.
- `INTERSECT` implicitly supposes a `DISTINCT` operation.
- The **results** of an `INTERSECT` is the intersection of right and left `SELECT` results, in other words, only records that are present in **both result sets** will be included in the result of the operation.

Set Difference

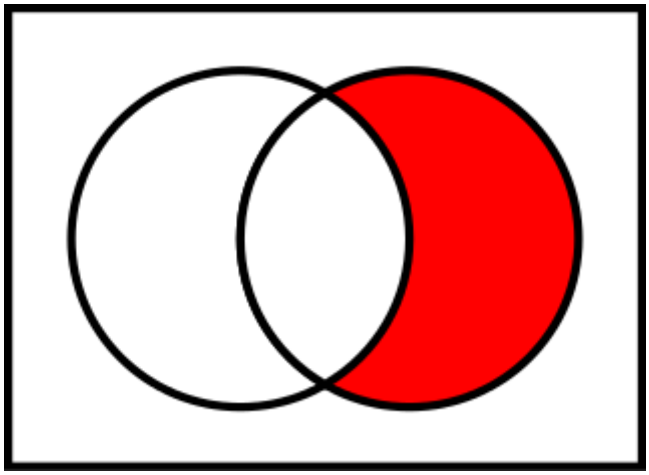

Figure 3: Venn Diagram for Difference.

Relational algebra syntax: $R - S$

SQL syntax:

```
SELECT ...
(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL |
DISTINCT]) SELECT ...
[(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL |
DISTINCT]) SELECT ...]
[ORDER BY [column [, column ...]]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Note:

- that `UNION`, `EXCEPT` and `INTERSECT` SQL commands can be used in combination.
- `EXCEPT` implicitly supposes a `DISTINCT` operation.
- The **results** of `EXCEPT` is all records of the left `SELECT` result except records which are in right `SELECT` result set, in other words, it is the subtraction of the right result set from the left result set.

Cartesian Product

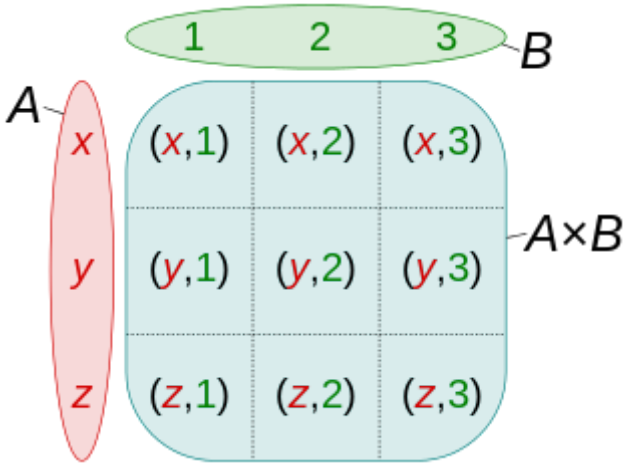

Figure 4: Cartesian Product.

Relational algebra syntax: $R \times S$

Selection

Relational algebra syntax: $\sigma_{\theta}(R)$

where θ is the selection condition.

Projection

Relational algebra syntax: $\pi_{A_1,A_2,...,A_n}(R)$

where $A_1, A_2, \ldots, A_n$ is the attributes/column names of the relation/table.

Rename

Relational algebra syntax: $\rho_{R_{new}}(R_{old})$

where R is the relation the requires the name change.

Natural Join

Relational algebra syntax: $R \bowtie S$

where the relations R and S are joined via the cartesian product of the attributes with the same name of the 2 relations. One copy of the duplicate attributes are removed.

Theta Join

Relational algebra syntax: $R \bowtie_{\theta} S$

where the relations R and S are joined via the cartesian product and a selection of the 2 relations. This is also the basic join that most DBMS implements.

Division

Relational algebra syntax: $R \div S$

where the relations R and S must follow these properties:

- Attributes of S is proper subset of Attributes of R.
- The relation returned by division operator will have attributes = (All attributes of R – All Attributes of S)
- The relation returned by division operator will return those tuples from relation R which are associated to every S's tuple.

Stored Function & Stored Procedures

Remember to switch the delimiter to another symbol like `//` using the `DELIMITER` command placed before and after the stored function or stored procedure. Delimiter symbols **must have** a space between the `DELIMITER` command and the symbol itself.

Stored Functions

SQL syntax:

```
CREATE FUNCTION [IF NOT EXISTS] func_name ([func_parameter[,...]])
    RETURNS type
    RETURN func_body
```

where:

- `type` is any valid MariaDB data type
- `func_parameter` are the parameters of the function
- `func_body` contains the procedures that the function is to do but most of the time, it is replaced with compound statements that has the `BEGIN...END` statements.
- invoked using the `SELECT` statement

Stored Procedures

SQL syntax:

```
CREATE PROCEDURE proc_name ([ IN | OUT | INOUT ] param_name datatype, ...)
    routine_body
```

where:

- does not explicitly return a value.
- `param_name` is the parameter name of the procedure. It must have should have the parameter option `IN` or `OUT` or `INOUT` preceding it to determine if the parameter is an input or an output or both input & output parameter. Parameter's datatype must also be declared.
- `routine_body` contains the procedures that the procedure is to do but most of the time, it is replaced with compound statements that has the `BEGIN...END` statements.
- invoked using the `CALL` statement

Triggers

Remember to switch the delimiter to another symbol like `//` using the `DELIMITER` command placed before and after the stored function or stored procedure. Delimiter symbols **must have** a space between the `DELIMITER` command and the symbol itself.

SQL syntax:

```
CREATE TRIGGER trigger_name
    {BEFORE | AFTER} {INSERT | UPDATE | DELETE }
    ON table_name FOR EACH ROW
    [{ FOLLOWS | PRECEDES } other_trigger_name ]
    trigger_body;
```

There are 3 trigger types:

- `INSERT` trigger - automatically executed when an `INSERT` statement adds a new row to a table.
- `UPDATE` trigger - automatically fired when an `UPDATE` statement modifies the data on a table.
- `DELETE` trigger - automatically invoked when a `DELETE` statement removes one or more rows from a table.

Each trigger is further classified into their trigger time:

- `BEFORE` - fired right before an event occurs. For example, before the `INSERT` event occurs, you can validate the values that are being inserted. Typically, you use a `BEFORE` trigger for data cleansing and modification.
- `AFTER` - invoked after an event occurs. Generally, you use an `AFTER` trigger to keep audit trails.

SQL Errors

SQL syntax:

```
SIGNAL SQLSTATE [VALUE]
    [SET error_property
    [, error_property] ...]
```

where:

- `error_property` is one or more properties to set for the SQL exception
- value of `45000` is recommended for customized errors as it is not part of the defined error codes.

Transaction

SQL syntax:

```
START TRANSACTION [transaction_property [, transaction_property] ...] | BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

where the transaction property has the following options:

- WITH CONSISTENT SNAPSHOT
- READ WRITE
- READ ONLY

Note that although `BEGIN` can be used for transactions, it is always better to use the `START TRANSACTION` command instead.