# MariaDB Cheat Sheet

A comprehensive cheat sheet for MariaDB, covering essential commands, data types, SQL syntax, and administrative tasks.

## Basic SQL Commands

### Data Definition Language (DDL)

| | |
|---|---|
| `CREATE DATABASE database_name;` | Creates a new database. |
| `DROP DATABASE database_name;` | Deletes an existing database. |
| `CREATE TABLE table_name (column_name datatype, ...);` | Creates a new table. |
| `ALTER TABLE table_name ADD column_name datatype;` | Adds a new column to a table. |
| `ALTER TABLE table_name DROP COLUMN column_name;` | Deletes a column from a table. |
| `DROP TABLE table_name;` | Deletes an existing table. |

### Data Manipulation Language (DML)

| | |
|---|---|
| `INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);` | Inserts new data into a table. |
| `UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;` | Updates existing data in a table. |
| `DELETE FROM table_name WHERE condition;` | Deletes data from a table. |
| `SELECT column1, column2, ... FROM table_name WHERE condition;` | Retrieves data from a table. |
| `REPLACE INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);` | Replaces data in a table if a row with the same primary key or unique key exists. |
| `TRUNCATE TABLE table_name;` | Removes all rows from a table. It's faster than DELETE because it deallocates the table's data pages. |

### Data Control Language (DCL)

| | |
|---|---|
| `GRANT permission ON database_name.table_name TO 'user'@'host';` | Grants privileges to a user. |
| `REVOKE permission ON database_name.table_name FROM 'user'@'host';` | Revokes privileges from a user. |
| `SET PASSWORD FOR 'user'@'host' = PASSWORD('new_password');` | Sets a new password for a user. |
| `FLUSH PRIVILEGES;` | Reloads the grant tables. |

## Advanced SQL Queries

### Joins

`INNER JOIN`: Returns rows when there is a match in both tables.

```
SELECT * FROM table1 INNER JOIN table2 ON table1.column = table2.column;
```

`LEFT JOIN`: Returns all rows from the left table, and the matched rows from the right table. If there is no match, the result is NULL on the right side.

```
SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column;
```

`RIGHT JOIN`: Returns all rows from the right table, and the matched rows from the left table. If there is no match, the result is NULL on the left side.

```
SELECT * FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;
```

`FULL OUTER JOIN`: Returns all rows when there is a match in one of the tables. Note: MariaDB doesn't directly support `FULL OUTER JOIN`, but it can be emulated using `UNION`.

```
SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column
UNION
SELECT * FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;
```

`CROSS JOIN`: Returns the Cartesian product of the tables.

```
SELECT * FROM table1 CROSS JOIN table2;
```

### Subqueries

A subquery is a query nested inside another query.

```
SELECT column1 FROM table1 WHERE column2 IN (SELECT column2 FROM table2);
```

`EXISTS`: Checks for the existence of rows in a subquery.

```
SELECT column1 FROM table1 WHERE EXISTS (SELECT 1 FROM table2 WHERE table1.column = table2.column);
```

`ANY` or `SOME`: Compare a value to each value returned by a subquery.

```
SELECT column1 FROM table1 WHERE column2 > ANY (SELECT column2 FROM table2);
```

`ALL`: Compare a value to all values returned by a subquery.

```
SELECT column1 FROM table1 WHERE column2 > ALL (SELECT column2 FROM table2);
```

Subqueries in the `FROM` clause: Can be used to treat the result of a query as a table.

```
SELECT column1, AVG(column2) FROM (SELECT column1, column2 FROM table1 WHERE condition) AS subquery GROUP BY column1;
```

### Aggregate Functions

| | |
|---|---|
| `COUNT(column)` | Counts the number of rows. |
| `SUM(column)` | Calculates the sum of values in a column. |
| `AVG(column)` | Calculates the average of values in a column. |
| `MIN(column)` | Finds the minimum value in a column. |
| `MAX(column)` | Finds the maximum value in a column. |
| `GROUP_CONCAT(column)` | Concatenates values from a column into a single string. |

## MariaDB Specific Features

### Storage Engines

MariaDB supports multiple storage engines, each with its own characteristics.

- `InnoDB`: Default storage engine, supports transactions, row-level locking, and foreign keys.
- `MyISAM`: Older storage engine, faster for read-heavy workloads but lacks transaction support.
- `Aria`: Designed for temporary tables and internal use, supports page caching for faster performance.
- `Memory`: Stores data in memory, very fast but data is lost when the server restarts.
- `Spider`: Allows partitioning tables across multiple MariaDB servers.

To specify a storage engine when creating a table:

```
CREATE TABLE table_name (column_name datatype) ENGINE=InnoDB;
```

To change the storage engine of an existing table:

```
ALTER TABLE table_name ENGINE=MyISAM;
```

### Dynamic Columns

Dynamic columns allow storing multiple values in a single column, similar to JSON or key-value stores.

- `COLUMN_CREATE(name, type, value)`: Creates a dynamic column.
- `COLUMN_GET(column, path)`: Retrieves a value from a dynamic column.
- `COLUMN_ADD(column, name, type, value)`: Adds a new value to a dynamic column.
- `COLUMN_DELETE(column, name)`: Deletes a value from a dynamic column.
- `COLUMN_LIST(column)`: Lists all names in a dynamic column.
- `COLUMN_EXISTS(column, name)`: Checks if a name exists in a dynamic column.

Example:

```
CREATE TABLE dynamic_table (id INT, data BLOB);
INSERT INTO dynamic_table VALUES (1, COLUMN_CREATE('name', 'string', 'John', 'age', 'int', 30));
SELECT COLUMN_GET(data, 'name' AS CHAR) FROM dynamic_table WHERE id = 1;
```

### JSON Functions

MariaDB provides functions for working with JSON data.

- `JSON_TYPE(json_doc)`: Returns the type of the JSON document.
- `JSON_EXTRACT(json_doc, path)`: Extracts a value from a JSON document.
- `JSON_INSERT(json_doc, path, val)`: Inserts a new value into a JSON document.
- `JSON_REPLACE(json_doc, path, val)`: Replaces an existing value in a JSON document.
- `JSON_REMOVE(json_doc, path)`: Removes a value from a JSON document.
- `JSON_OBJECT(key, value, ...)`: Creates a JSON object.
- `JSON_ARRAY(val, ...)`: Creates a JSON array.

Example:

```
CREATE TABLE json_table (id INT, data JSON);
INSERT INTO json_table VALUES (1, '{"name": "John", "age": 30}');
SELECT JSON_EXTRACT(data, '$.name') FROM json_table WHERE id = 1;
```

## Administrative Tasks

## User Management

| | |
|---|---|
| `CREATE USER 'user'@'host' IDENTIFIED BY 'password';` | Creates a new user account. |
| `RENAME USER 'old_user'@'host' TO 'new_user'@'host';` | Renames an existing user account. |
| `DROP USER 'user'@'host';` | Deletes a user account. |
| `SHOW GRANTS FOR 'user'@'host';` | Displays the privileges granted to a user. |
| `SET PASSWORD FOR 'user'@'host' = PASSWORD('new_password');` | Sets a new password for a user. |
| `ALTER USER 'user'@'host' IDENTIFIED BY 'new_password';` | Changes the password for a user. |

## Backup and Restore

Backup:

```
mysqldump -u user -p database_name > backup.sql
```

Restore:

```
mysql -u user -p database_name < backup.sql
```

Using `mariabackup` (for InnoDB/Aria):

Backup:

```
mariabackup --backup --target-dir=/path/to/backup
```

Prepare:

```
mariabackup --prepare --target-dir=/path/to/backup
```

Restore:

```
mariabackup --copy-back --target-dir=/path/to/backup
chown -R mysql:mysql /var/lib/mysql
```

Consider using logical (mysqldump) for smaller databases and physical (mariabackup) for larger databases

## Server Management

| | |
|---|---|
| `SHOW GLOBAL STATUS;` | Displays the current status of the MariaDB server. |
| `SHOW GLOBAL VARIABLES;` | Displays the global variables of the MariaDB server. |
| `SHOW PROCESSLIST;` | Displays a list of currently running threads. |
| `KILL thread_id;` | Kills a specific thread. |
| `FLUSH TABLES;` | Closes all open tables, forcing them to be written to disk. |
| `RESET QUERY CACHE;` | Removes all query results from the query cache. |