# SQL Databases

Databases are means that facilitate data persistence, querying, and manipulations.

There are many types of databases:

- Relational DBs (MySQL, PostgreSQL, MSSQL)
- Document DBs (MongoDB, Cosmos DB, CouchDB)
- Graph DBs (Neo4j, ArangoDB)
- Key-Value DBS (Redis, Memcached)
- Wide Column DBs (Cassandra, ScyllaDB)

We will concentrate our attention on relational databases which are to most spread DBs.
For Our examples we will use MySQL and discuss the building blocks of SQL: tables, columns, primary keys, and foreign keys.

sourcemind

# Tables

The database consists of tables. Tables have a spreadsheet-like structure. They consist of rows and columns.

- Each row represents an entity, for example, a car, user, product, or order, e.g.
- Each column represents a field in an entity and has a type. For example name of the user with the string type, and the price of the order with the number type.

| id | name | age | mail | created-at |
|---|---|---|---|---|
| 1 | John | 22 | John@mail.com | 2020-11-11 |
| 2 | Alice | 33 | Alice@mail.com | 2021-10-03 |
| 3 | Ben | 44 | Ben@mail.com | 2019-04-07 |

id – INT
name – VARCHAR(50)
age – INT
mail – VARCHAR(70)
created-at - DATE

# Keys

There are many types of keys in SQL, but the widely used ones are Primary and Foreign keys.

- Primary Key

  The PRIMARY KEY constraint uniquely identifies each record in a table.

  Primary keys must contain UNIQUE values, and cannot contain NULL values.

  Table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

- Foreign Key

  The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

  A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

  The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

| id | brand | year | model | owner-id |
|---|---|---|---|---|
| 1 | Mercedes | 2010 | C | 1 |
| 2 | BMW | 2015 | 3 | 2 |

Here we have the cars table which has a foreign key reference to its owner via its owner-id column

# Create Table

The CREATE TABLE statement is used to create a new table in a database.

```
CREATE TABLE table_name (
    column1 <type> <options>,
    column2 <type> <options>,
    column3 <type> <options>,
    ....

    PRIMARY KEY (<col_name>),
    FOREIGN KEY (<key_col_name>) REFERENCES <parent_table>(<parent_col_name>)
);
```

Let's create the users table that we defined previously

```
CREATE TABLE users(
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(50),
    age INT,
    mail VARCHAR(70),
    create_at DATE,

    PRIMARY KEY (id)
)
```

sourcemind

# Alter Table and Drop Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
The ALTER TABLE statement is also used to add and drop various constraints on an existing table.
Let's create a new basic table with only one column for our examples.

```
CREATE TABLE tasks(id INT)
```

Let's add a few columns into tasks.

```
ALTER TABLE tasks
ADD name VARCHAR(250),
ADD age INT
```

Let's delete age column.

```
ALTER TABLE tasks
DROP age;
```

Let's change the size of name column

```
ALTER TABLE tasks
MODIFY COLUMN name VARCHAR(50);
```

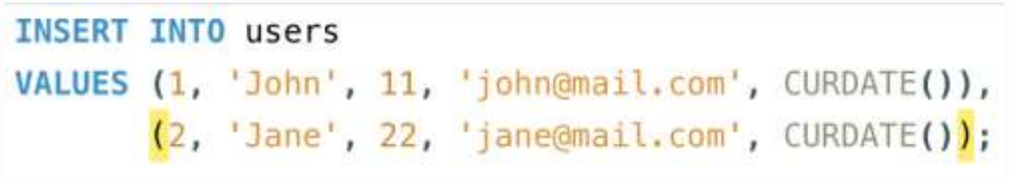
To delete table use Drop Table statement.

```
DROP TABLE tasks
```

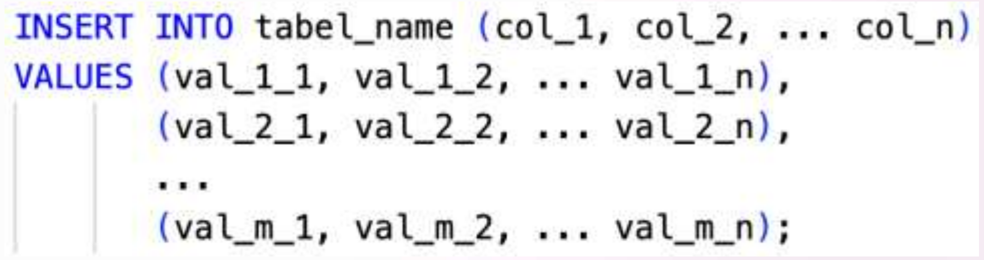
sourcemind

# Insert, Update, Delete

Use INSERT INTO statement to add data:
Let's add a few record into users table.

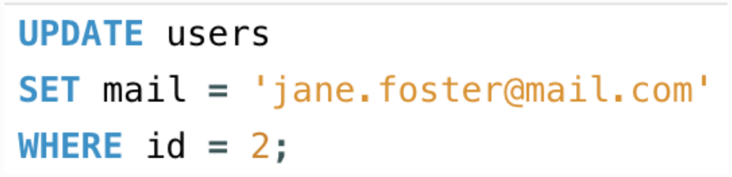


```
INSERT INTO users
VALUES (1, 'John', 11, 'john@mail.com', CURDATE()),
       (2, 'Jane', 22, 'jane@mail.com', CURDATE());
```

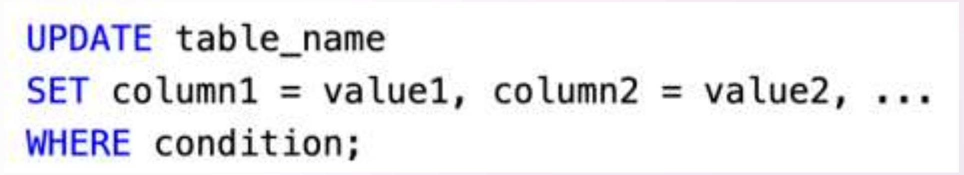


```
INSERT INTO tabel_name (col_1, col_2, ... col_n)
VALUES (val_1_1, val_1_2, ... val_1_n),
       (val_2_1, val_2_2, ... val_2_n),
       ...
       (val_m_1, val_m_2, ... val_m_n);
```

Use UPDATE statement to update records:
Let's update user Jane's email.

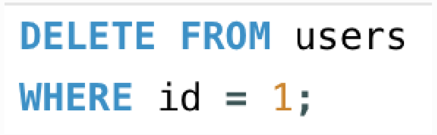


```
UPDATE users
SET mail = 'jane.foster@mail.com'
WHERE id = 2;
```

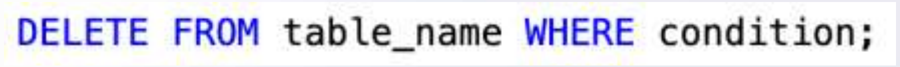


```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Use DELETE FROM statement to update records:
Let's delete first user.



```
DELETE FROM users
WHERE id = 1;
```



```
DELETE FROM table_name WHERE condition;
```

# Table Queries

The simplified version of MySql queries have this syntacs:

```
SELECT col_1, col_2, ... col_n
FROM table_name
WHERE conditions
ORDER BY col_name [asc | desc];
```

For example to get all adult users name and age we can use this query:

```
SELECT name, age FROM test.users
WHERE age > 18;
```

We can use asterisks '*' to select all columns:

```
SELECT * FROM test.users;
```

We can use complex condition by using boolean operators:

```
WHERE age > 18 AND created_at < '2020-01-01' OR mail = 'test@test.com';
```

To order records we use ORDER BY and order direc
by using ASC or DESC

```
SELECT * FROM test.users
ORDER BY name ASC;
```

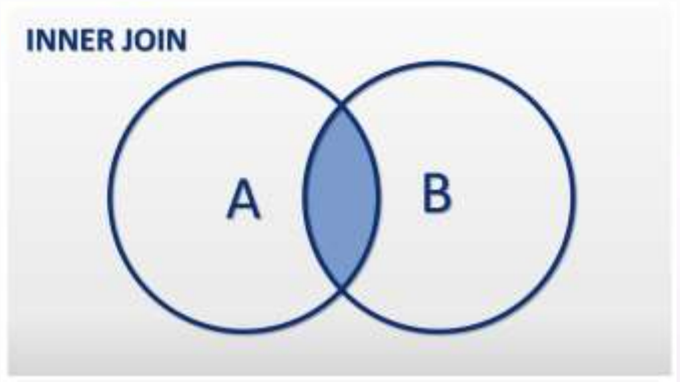
We can use LIMIT and OFFSET for pagination.
For example let's assume we want to skip first 20 record
and take next 10:

```
SELECT * FROM test.users
LIMIT 10
OFFSET 20;
```

sourcemind

# Inner Join

The INNER JOIN keyword selects records that have matching values in both tables.



INNER JOIN syntax looks like this:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

# INNER JOIN example

| id | name | age | mail | created-at |
|---|---|---|---|---|
| 1 | John | 22 | John@mail.com | 2020-11-11 |
| 2 | Alice | 33 | Alice@mail.com | 2021-10-03 |
| 3 | Ben | 44 | Ben@mail.com | 2019-04-07 |

| id | brand | year | model | owner-id |
|---|---|---|---|---|
| 1 | Mercedes | 2010 | C | 1 |
| 2 | BMW | 2015 | 3 | 2 |

```
SELECT users.id as userID, users.name, cars.id as carID, cars.brand, cars.model
FROM users
INNER JOIN cars
ON users.id = cars.owner_id;
```

| userID | name | cardID | brand | model |
|---|---|---|---|---|
| 1 | John | 1 | Mercedes | C |
| 2 | Alice | 2 | BMW | 3 |

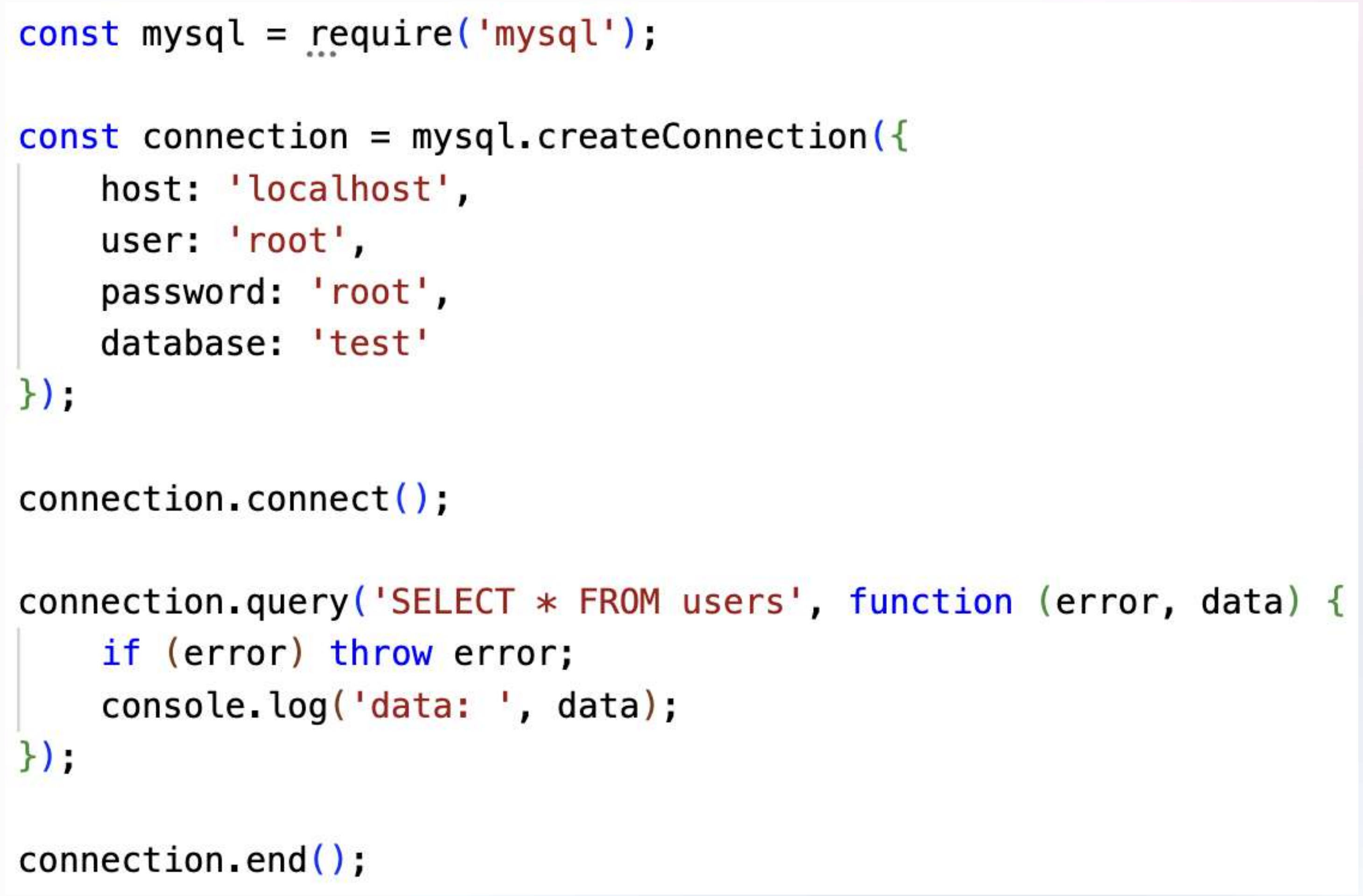
sourcemind

## mysql packag e

To connect to mysql db from node app we will use mysql package.

```
npm i mysql
```

After installing we can connect to DB and run our queries by using a connection.query function as we did in MySql Workbench. In the end, we must close the connection by calling the end function.



```
const mysql = require('mysql');

const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'root',
    database: 'test'
});

connection.connect();

connection.query('SELECT * FROM users', function (error, data) {
    if (error) throw error;
    console.log('data: ', data);
});

connection.end();
```