Purwadhika
Digital Technology School

Full Stack AI Software Development

# Basic Database Design and Development

Job Connector Program

## Outline

### Introduction Database

Learn the basics of databases, how data is stored, and how servers handle structured information.

### SQL Script and Basic CRUD

Understand SQL commands to create, read, update, and delete data effectively in a database.

### Integration SQL with API

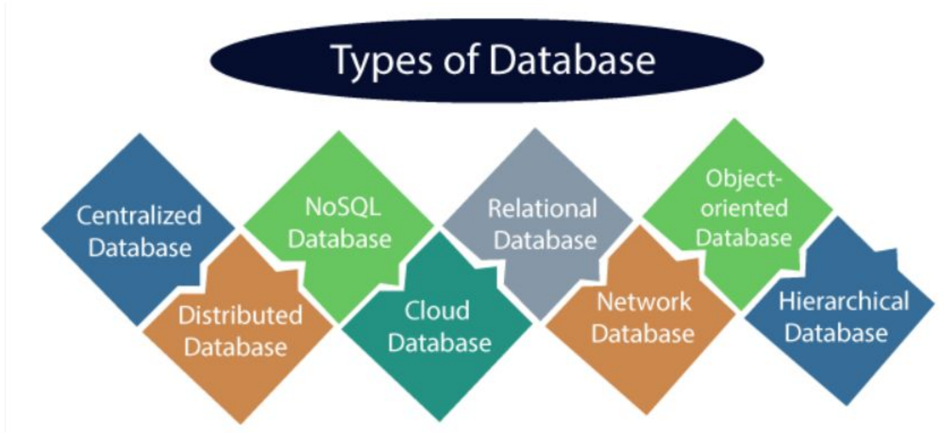
Discover how to connect databases with RESTful APIs using Express and TypeScript.

# Introduction to Database

- A database is an organized collection of data.
- The main purpose of database is to operate large amount of information by storing, retrieving and managing.
- There are many dynamic websites on the world wide web nowadays which are handled through databases. For example, an app to checks the availability of rooms in a hotel. It is an example of dynamic website that uses database.

## Common Types of Database

- Centralised database
- Distribution database
- Relational database
- NoSQL database
- Object-oriented database
- Etc ...



## Relational Database Management System (RDBMS)

- **How it works** - A relational database is the most commonly used database. It contains several tables, and each table has its primary key. Due to a collection of an organized set of tables, data can be accessed easily in RDBMS.
- **What is table with relation** - Everything in a relational database is stored in the form of relations. The RDBMS database uses tables to store data. A table is a collection of related data entries and contains rows and columns to store data. Each table represents some real-world objects such as person, place, or event about which information is collected. The organized collection of data into a relational table is known as the logical view of the database.

All modern database management systems like SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL, and Microsoft Access are based on **RDBMS**.

## Relational Database Management System (RDBMS)

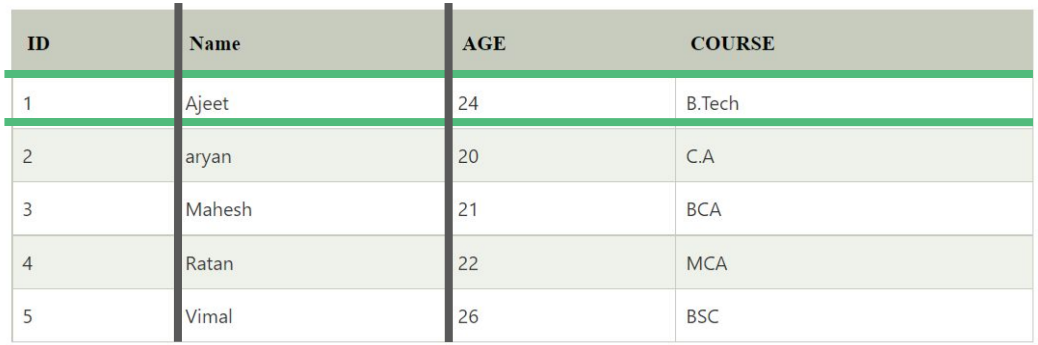
**Properties of a Relation:**

- Each relation has a unique name by which it is identified in the database.
- Relation does not contain duplicate tuples.
- The tuples of a relation have no specific order.

# Relational Database Management System (RDBMS)

A table is a collection of related data entries and contains rows and columns to store data,

- **Row or record -** A row of a table is also called a record or tuple. It contains the specific information of each entry in the table. It is a horizontal entity in the table.
- **Column or attribute** - A column is a vertical entity in the table which contains all information associated with a specific field in a table.



| ID | Name | AGE | COURSE |
|---|---|---|---|
| 1 | Ajeet | 24 | B.Tech |
| 2 | aryan | 20 | C.A |
| 3 | Mahesh | 21 | BCA |
| 4 | Ratan | 22 | MCA |
| 5 | Vimal | 26 | BSC |

## Structured Query Language

- SQL (Structured Query Language) is used to communicate with a database. It’s the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database.
- Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.
- The standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

## Structured Query Language

**SQL uses:**

- **Data definition**:  It is used to define the structure and organization of the stored data and relationships among the stored data items.
- **Data retrieval:** SQL can also be used for data retrieval.
- **Data manipulation**: If the user wants to add new data, remove data, or modifying in existing data then SQL provides this facility also.
- **Access control**: SQL can be used to restrict a user's ability to retrieve, add, and modify data, protecting stored data against unauthorized access.
- **Data sharing**: SQL is used to coordinate data sharing by concurrent users, ensuring that changes made by one user do not inadvertently wipe out changes made at nearly the same time by another user.

## PostgreSQL

PostgreSQL is a powerful, open-source object-relational database system with over 30 years of active development. Known for its advanced features, extensibility, and standards compliance, PostgreSQL is widely used in web applications and data-centric environments. It supports complex queries, large data volumes, and both relational and non-relational data. High-profile companies like Instagram, Reddit, and Spotify rely on PostgreSQL for its reliability, scalability, and performance, making it a top choice for modern applications that require robust and flexible database solutions.

PostgreSQL®

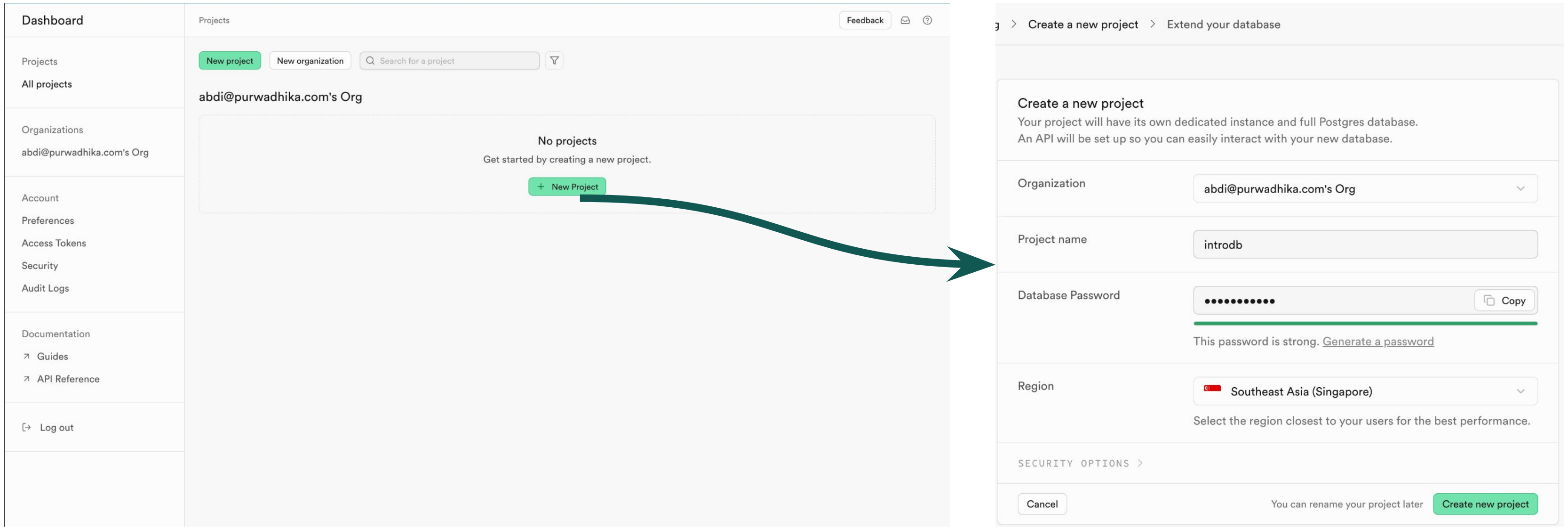
# PostgreSQL

To use postgresql we have some option such as :

- PostgreSQL Desktop
  - Go to: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads
  - Installation guideline : https://www.postgresqltutorial.com/postgresql-getting-started/install-postgresql/
  - You can install PostgreSQL Server only or with PgAdmin (UI editor PostgreSQL)
- Supabase with DBEAVER
  - Supabase for PostgreSQL Server Online
  - DBEAVER for UI editor
    - Download link : https://dbeaver.io/download/

# Setup PostgreSQL Database Server Supabase

Create new project in Supabase :



Dashboard

Projects
All projects

Organizations
abdi@purwadhika.com's Org

Account
Preferences
Access Tokens
Security
Audit Logs

Documentation
Guides
API Reference

Log out

Projects — Feedback

New project | New organization | Search for a project

abdi@purwadhika.com's Org

No projects
Get started by creating a new project.
+ New Project

> Create a new project > Extend your database

Create a new project
Your project will have its own dedicated instance and full Postgres database.
An API will be set up so you can easily interact with your new database.

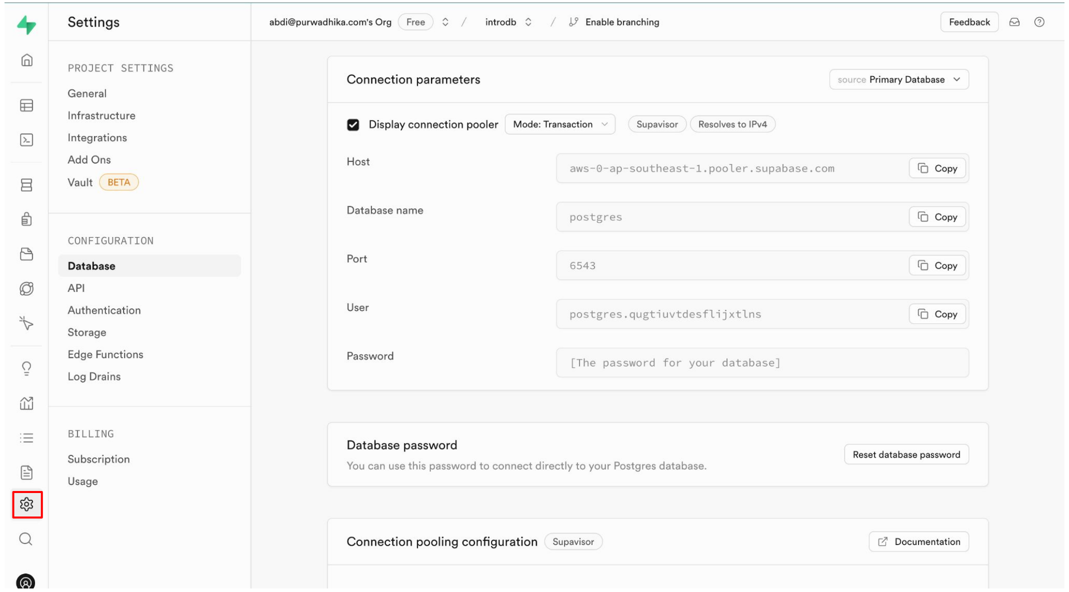
| | |
|---|---|
| Organization | abdi@purwadhika.com's Org |
| Project name | introdb |
| Database Password | ●●●●●●●●●●● (Copy) This password is strong. Generate a password |
| Region | Southeast Asia (Singapore) Select the region closest to your users for the best performance. |

SECURITY OPTIONS >

Cancel — You can rename your project later — Create new project

# Setup PostgreSQL Database Server Supabase

- Waiting setting up project until ready and move to project setting.
- You can view connection parameters and config in DBEAVER



Connection parameters

source Primary Database

Display connection pooler [✓] Mode: Transaction Supavisor Resolves to IPv4

| | |
|---|---|
| Host | aws-0-ap-southeast-1.pooler.supabase.com |
| Database name | postgres |
| Port | 6543 |
| User | postgres.qugtiuvtdesflijxtlns |
| Password | [The password for your database] |

Database password

You can use this password to connect directly to your Postgres database.

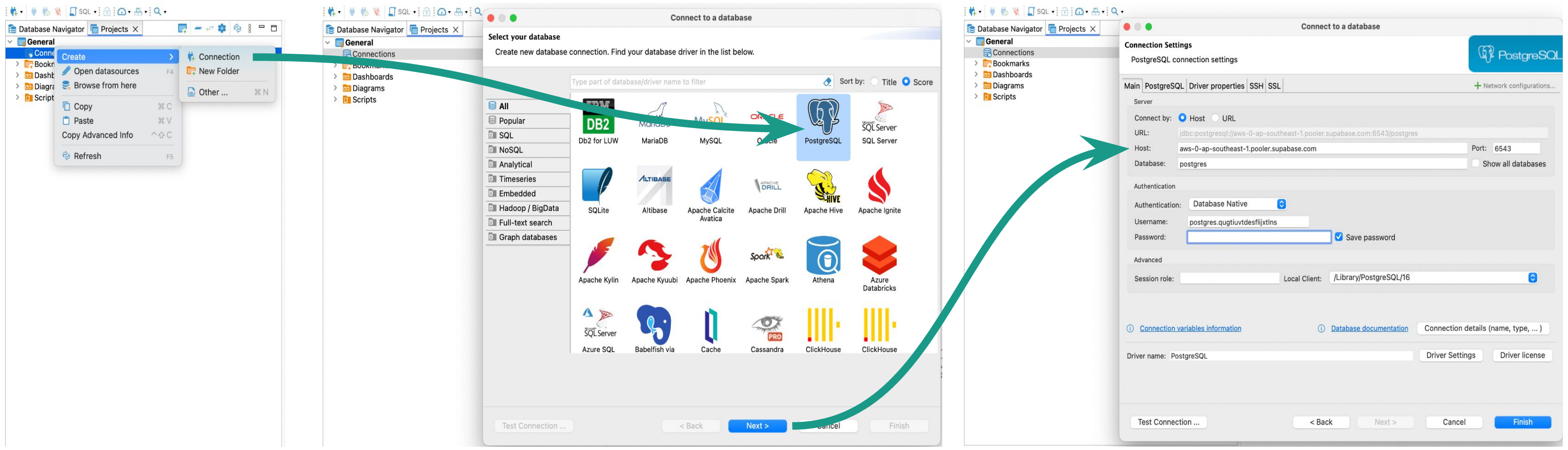
Reset database password

Connection pooling configuration Supavisor

Documentation

# Setup DBEAVER to Connect PostgreSQL Supabase

- Open DBEAVER and create new connection



Connection Settings
PostgreSQL connection settings

Main | PostgreSQL | Driver properties | SSH | SSL

Server
Connect by: [✓ Host] URL
URL: jdbc:postgresql://aws-0-ap-southeast-1.pooler.supabase.com:6543/postgres
Host: aws-0-ap-southeast-1.pooler.supabase.com
Port: 6543
Database: postgres
Show all databases

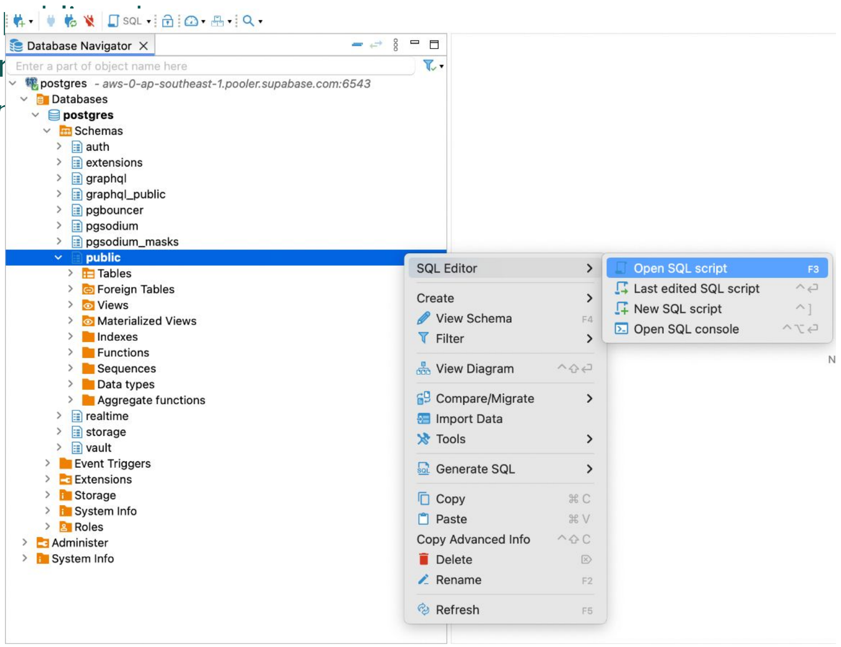
Authentication
Authentication: Database Native
Username: postgres.qugtiuvtdesflijxtlns
Password:
[✓] Save password

Advanced
Session role:
Local Client: /Library/PostgreSQL/16

Driver name: PostgreSQL

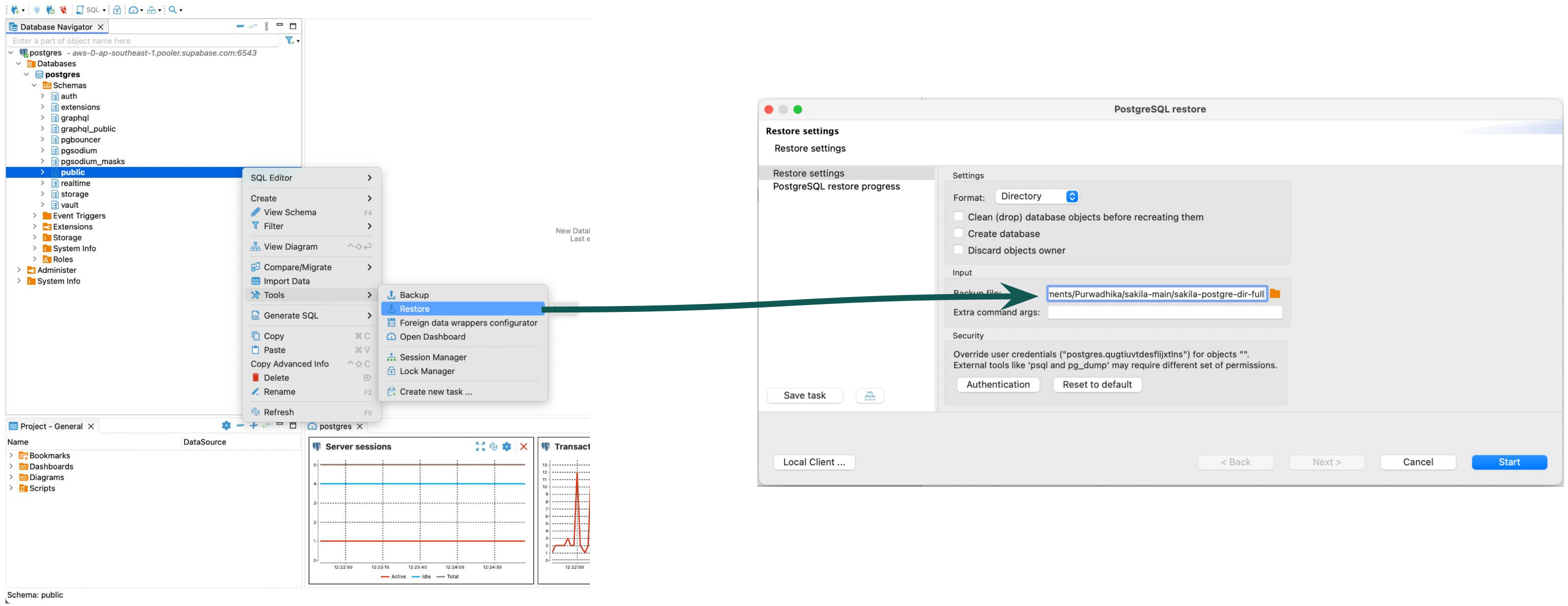
# Setup DBEAVER to Connect PostgreSQL Supabase

- If success to connect, try to access database and
- After that we can try to run SQL script start from cr
    - *After create table from script don’t forget to re-conn*



# Try to Import Data

- Download Sakila backup data : [https://drive.google.com/file/d/1f_GW-fk8-AFC5OXJ2fxmCabydOFkSrBk/view?usp=sharing](https://drive.google.com/file/d/1f_GW-fk8-AFC5OXJ2fxmCabydOFkSrBk/view?usp=sharing)



Settings

Format: Directory

- Clean (drop) database objects before recreating them
- Create database
- Discard objects owner

Input

Backup file: ments/Purwadhika/sakila-main/sakila-postgre-dir-full

Extra command args:

Security

Override user credentials ("postgres.qugtiuvtdesflijxtlns") for objects "".
External tools like 'psql and pg_dump' may require different set of permissions.

# PostgreSQL Data Types

**The data type** of a column defines the kind of value it can hold, such as integers, characters, dates, times, and more.

An SQL developer must choose the data type for each column when creating a table. The data type helps PostgreSQL understand the expected type of data and how to interact with it. In PostgreSQL, the main data types include: character, numeric, date and time, boolean, as well as specialized types like JSON and arrays.

.

## PostgreSQL - Character Types

| **Data type** | **Description** |
|---|---|
| CHAR(size) | A FIXED length string (can contain letters, numbers, and special characters). |
| VARCHAR(size) | A VARIABLE length string (can contain letters, numbers, and special characters). |
| TEXT | Variable-length character string with no maximum length. Suitable for large text data. |

## PostgreSQL - Numeric Data Types

| **Data type** | **Description** |
|---|---|
| SMALLINT | A small integer (2-byte). Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255) |
| INTEGER | A integer (4-byte). Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255) |
| BIGINT | A large integer (8-byte). Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255) |

## PostgreSQL - Numeric Data Types

| **Data type** | **Description** |
|---|---|
| NUMERIC(*p*, *s*) | Exact numeric with a specified precision `p` and scale `s`. Suitable for financial calculations where exactness is crucial. |
| DECIMAL(*p*, *s*) | Similar to `NUMERIC`, with specified precision `p` and scale `s` |
| REAL | Single-precision floating-point number, approximately 1.18E-38 to 3.4E+38. |
| DOUBLE PRECISION | Double-precision floating-point number, approximately 2.23E-308 to 1.79E+308. |

## PostgreSQL - Date and Time Data Types

| **Data type** | **Description** |
|---|---|
| DATE | A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31' |
| TIME [ WITHOUT TIME ZONE ] | Time of day (hours, minutes, seconds) with optional time zone. |
| TIMESTAMP [ WITHOUT TIME ZONE ] | Date and time (year, month, day, hours, minutes, seconds) with optional time zone. |
| TIMESTAMPTZ | Date and time with time zone, adjusting for time zone differences. |
| INTERVAL | Time span, representing a period of time. |

## PostgreSQL - Other Data Types

| Data type | Description |
|---|---|
| BOOLEAN | Stores `TRUE`, `FALSE`, or `NULL`. |
| BYTEA | Stores binary data, such as images or files. |
| ARRAY | Allows storage of multiple values of a specified data type in a single column. For example, `INTEGER[]` stores an array of integers. |
| JSON | Stores JSON (JavaScript Object Notation) data. Useful for semi-structured data. |
| JSONB | Stores JSON data in a binary format. Provides efficient access and manipulation of JSON data. |

## PostgreSQL - Other Data Types

| **Data type** | **Description** |
|---|---|
| POINT | Represents a geometric point in a 2D plane. |
| LINE | Represents an infinite line. |
| LSEG | Represents a line segment. |
| BOX | Represents a rectangular box. |
| CIRCLE | Represents a circle. |
| CIDR | Stores IP network addresses. |
| INET | Stores individual IP addresses with optional subnet. |
| MACADDR | Stores MAC (Media Access Control) addresses. |

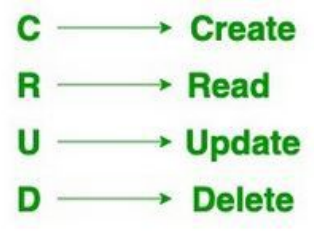
# PostgreSQL - Other Data Types

| Data type | Description |
|---|---|
| UUID | Stores universally unique identifiers, typically used for identifying records uniquely across different systems. |
| XML | Stores XML (eXtensible Markup Language) data, allowing for storage and querying of XML documents. |
| `INT4RANGE`, `INT8RANGE`, `NUMRANGE`, `TSRANGE`, `TSTZRANGE` | Represent ranges of integers, numeric values, timestamps, etc. |
| COMPOSITE | User-defined types that group multiple values into a single unit. Useful for creating custom data structures. |
| ENUM | Stores a predefined set of values. Useful for fields that have a limited set of possible values, like status codes. |
| TYPE | Allows the creation of custom data types based on user-defined requirements. |

## CRUD Database

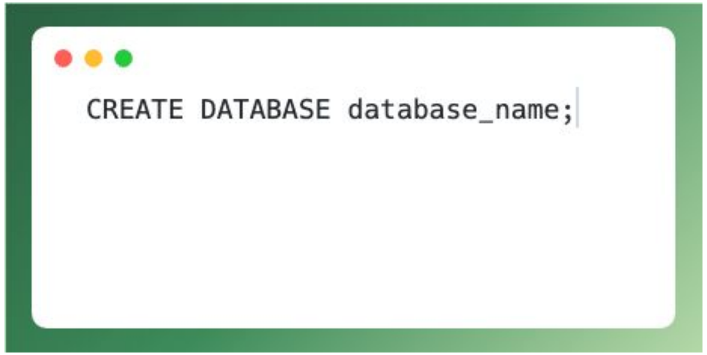
As we know that we can use PostgreSQL to use Structured Query Language to store the data in the form of RDBMS. SQL is the most popular language for adding, accessing, and managing content in a database. It is most noted for its quick processing, proven reliability, ease, and flexibility of use. The application is used for a wide range of purposes, including data warehousing, e-commerce, and logging applications. The most common use for PostgreSQL, however, is for the purpose of a web database.

PostgreSQL provides a set of some basic but most essential operations that will help you to easily interact with the PostgreSQL database and these operations are known as CRUD operations.



## Create Database

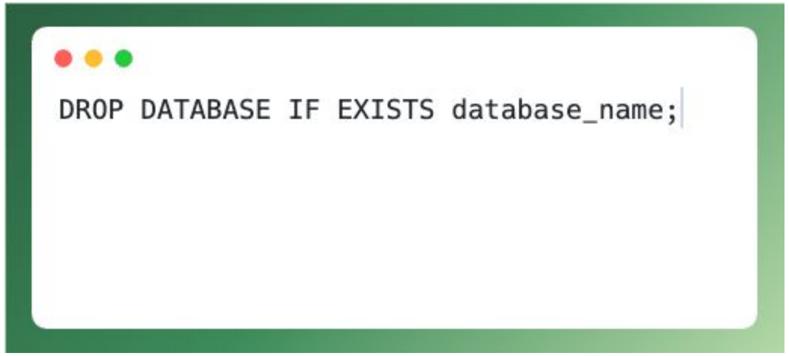
- **database_name** - This command creates a new database named `database_name`. Each database is a separate container for data, tables, schemas, and other objects. Creating a database is typically the first step when starting a new project or application.



```sql
CREATE DATABASE database_name;
```

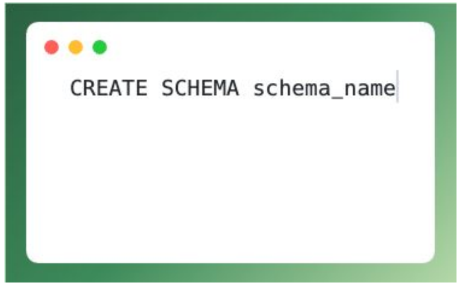
## Drop Database

This command removes the database named `database_name` along with all of its contents, including tables, data, and schemas. The `IF EXISTS` clause prevents an error if the database does not exist.



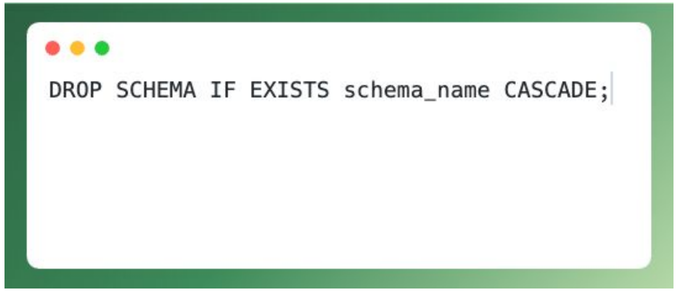
```
DROP DATABASE IF EXISTS database_name;
```

## Schema Management



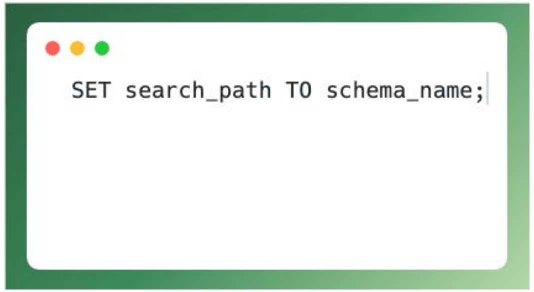
```
CREATE SCHEMA schema_name
```

A schema is a namespace that contains named database objects such as tables, views, and functions. Creating a schema helps organize these objects and manage permissions.



```
DROP SCHEMA IF EXISTS schema_name CASCADE;
```

This command removes the schema named `schema_name` and all objects contained within it. The `CASCADE` option is used to automatically remove all dependent objects such as tables and functions.

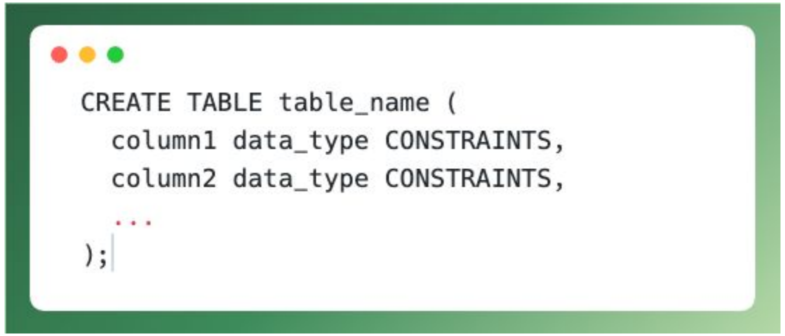


```
SET search_path TO schema_name;
```

This command sets the default schema that PostgreSQL uses when searching for tables and other objects. It helps in managing which schema is used for unqualified object names.

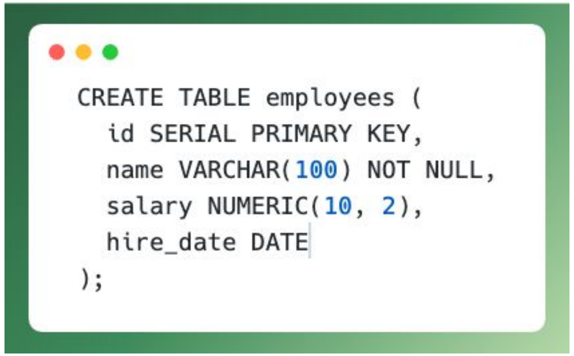
## Create Table

This command creates a new table named `table_name` with specified columns and their data types. Constraints (such as `NOT NULL`, `UNIQUE`, or `PRIMARY KEY`) define rules for data integrity.



```
CREATE TABLE table_name (
  column1 data_type CONSTRAINTS,
  column2 data_type CONSTRAINTS,
  ...
);
```

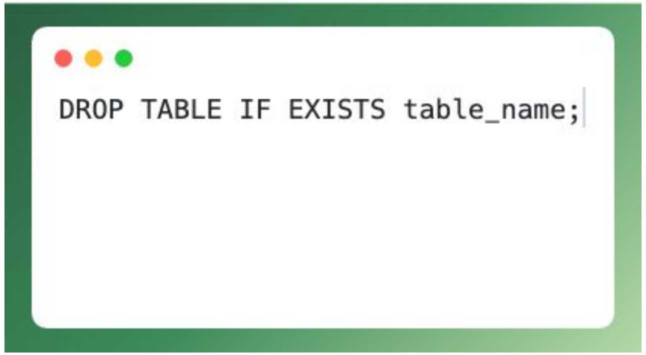
Example →



```
CREATE TABLE employees (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  salary NUMERIC(10, 2),
  hire_date DATE
);
```

## Drop Table

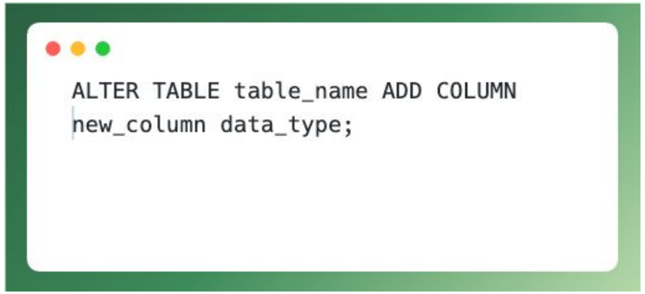
This command removes the table named `table_name` and all of its data. The `IF EXISTS` clause prevents an error if the table does not exist.



```
DROP TABLE IF EXISTS table_name;
```

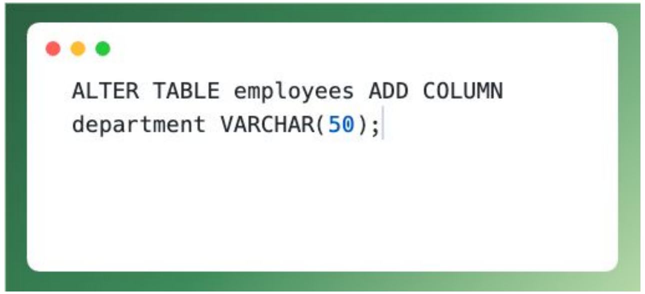
## Alter Table

This command adds a new column to the table. You can also use `ALTER TABLE` to modify or delete columns, rename columns, or add constraints.



```
ALTER TABLE table_name ADD COLUMN
new_column data_type;
```

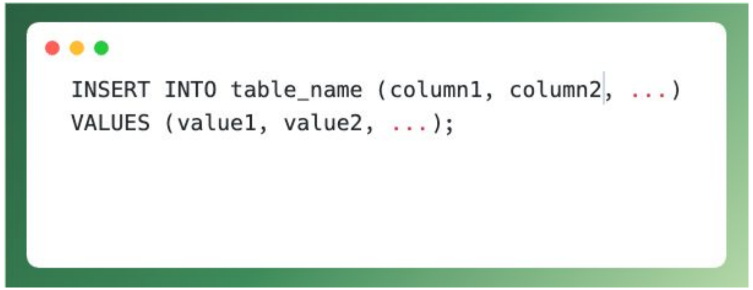
Example →



```
ALTER TABLE employees ADD COLUMN
department VARCHAR(50);
```

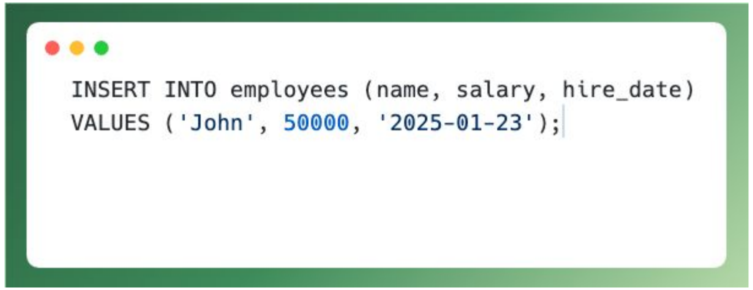
## Create (Insert Data)

This command adds a new row to the table with specified values for each column.



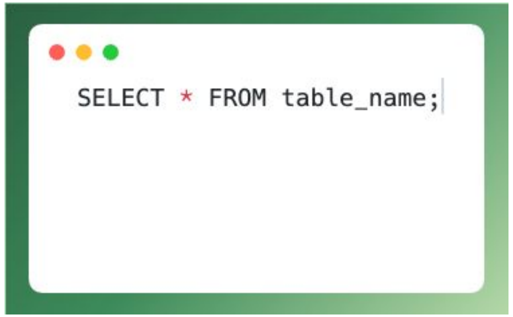
```sql
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Example →



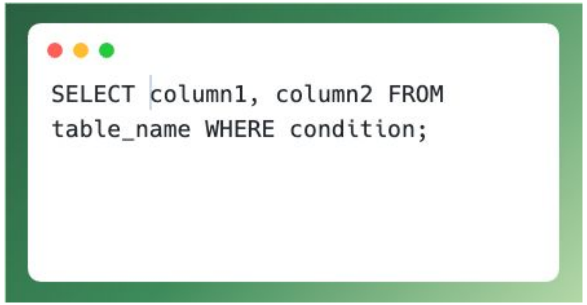
```sql
INSERT INTO employees (name, salary, hire_date)
VALUES ('John', 50000, '2025-01-23');
```

## Read (Retrieve Data)



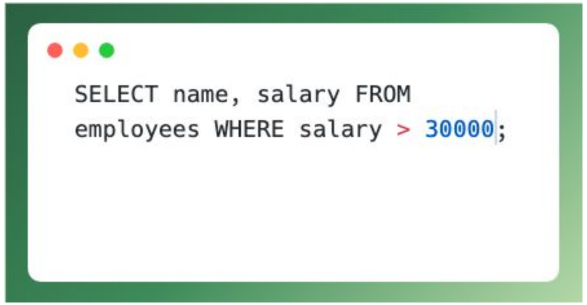
```
SELECT * FROM table_name;
```

This command to get all rows data.



```
SELECT column1, column2 FROM
table_name WHERE condition;
```

This command to get specific columns and filter data.

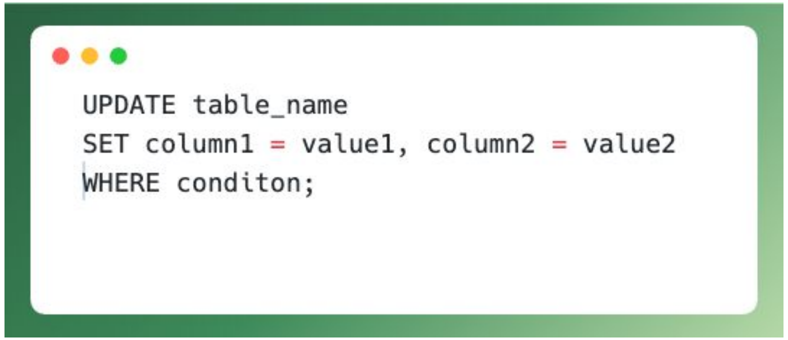


```
SELECT name, salary FROM
employees WHERE salary > 30000;
```

This retrieves the `name` and `salary` of employees whose salary is greater than 30,000.

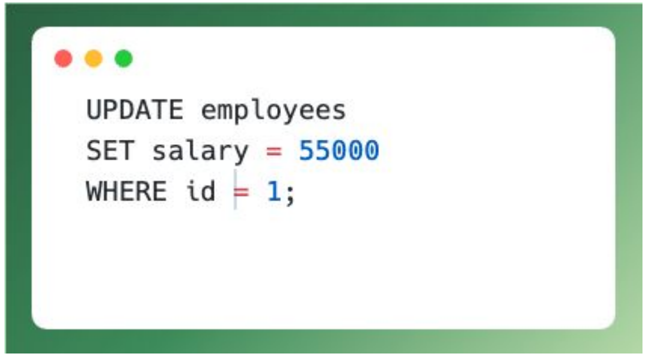
## Update (Modify Data)

This command modifies existing records in the table based on the specified condition.



```sql
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE conditon;
```

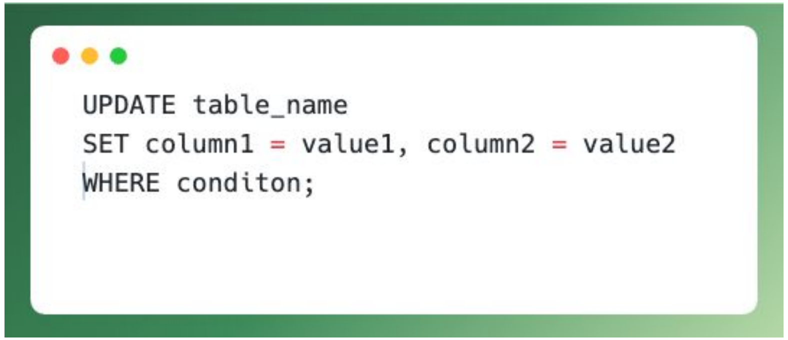
Example →



```sql
UPDATE employees
SET salary = 55000
WHERE id = 1;
```

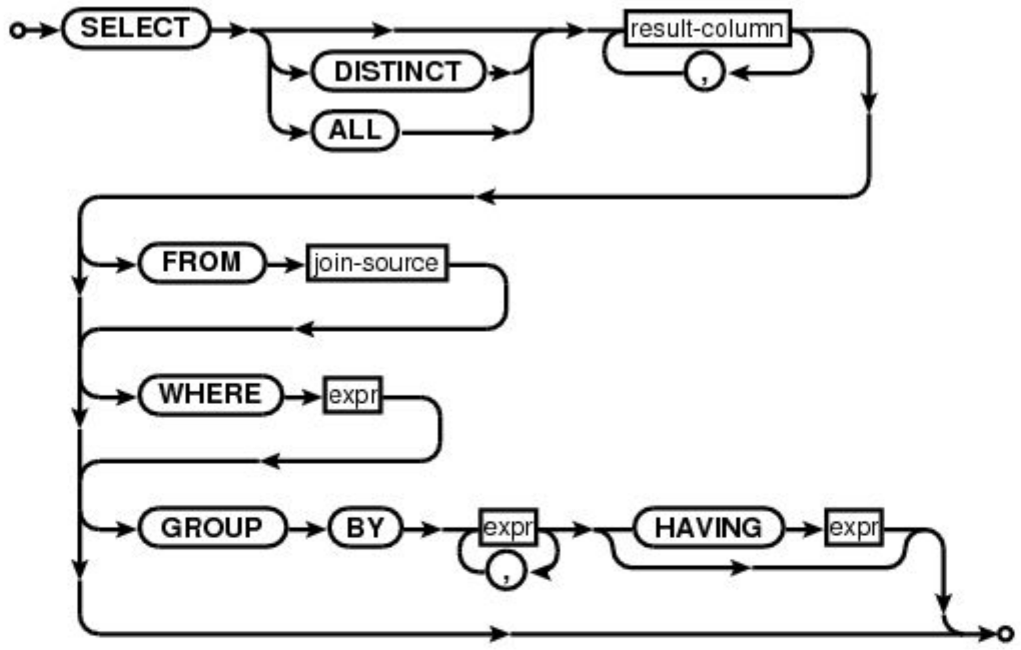
## Update Data

- **Table_name** : table_name is the name of the table whose column records we wish to update. Further, we need to specify all the columns and the expressions we want to assign to them in the comma-separated format after the SET keyword.
- **Column** : column1, column2 are the names of the column and expression is the value we want to assign to them.
- **Value** : Expression can be any literal values, constraints, any manipulated values of expressions that involve operations like addition, subtraction, product, division, square, etc or variables another column values or expressions formed from another table's columns in case of update join statement.
- **Restrictions [WHERE condition]** : Whenever, we have to mention that only some of the records that satisfy certain conditions should be updated from the update statement, we will have to mention all the conditions that should be fulfilled using the where clause in SQL. These conditions are referred to as restrictions. Using the where clause is optional.



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

## Clauses



**SQL system clauses** are keywords or statements to handle information. It helps to operate a group of the data and apply it to require conditions. The clauses apply conditions or select patterns to get information.

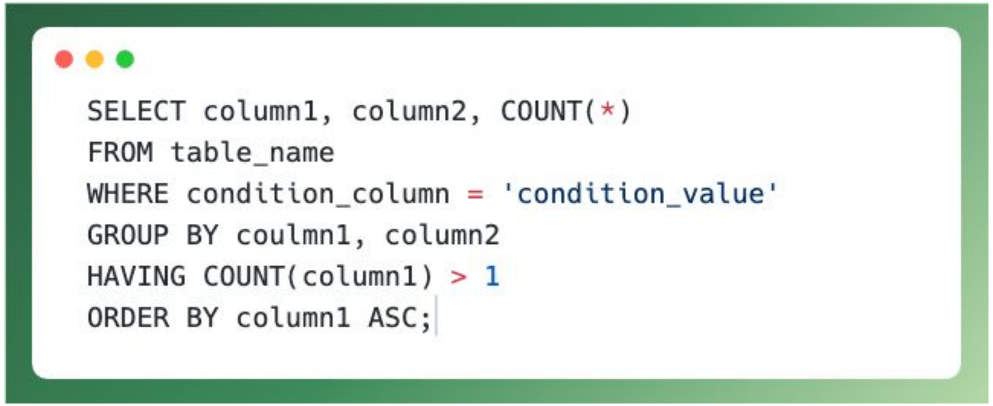
**SQL clauses** are not used to insert new data. You retrieve data using several clauses. The table uses either single or multiple clauses.

## Type of Clauses and Execution Order

| Clauses | Description |
|---|---|
| **"FROM" clause** | The database starts from the involved table(s), performs any joins if specified, and identifies the data source. |
| **"WHERE" clause** | Filters the rows based on the condition specified in the `WHERE` clause. Only rows that meet the condition will be processed further. |
| **"GROUP BY" clause** | After filtering, the selected rows can be grouped based on one or more columns. |
| **"HAVING" clause** | This clause filters the grouped results based on a condition. |
| **"SELECT" clause** | After filtering and grouping, the columns mentioned in the `SELECT` clause are retrieved and processed. |
| **"DISTINCT" clause** | If `DISTINCT` is used, duplicate rows are removed from the result set. |
| **"ORDER BY" clause** | The selected data is sorted based on the columns specified. |
| **“LIMIT/OFFSET” clause** | The final result can be limited in number using the `LIMIT` clause and/or offset using the `OFFSET` clause. |

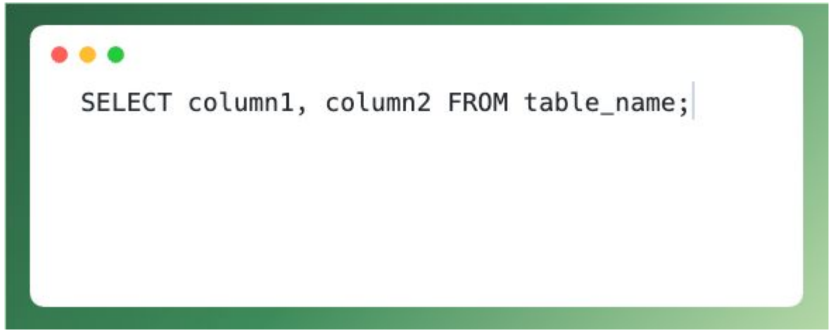
## Clauses Syntax

This syntax uses some clauses such as "FROM," "WHERE," "GROUP BY," and "ORDER BY." You select entire table columns or specific columns from the table. The particular field selects the other clauses.



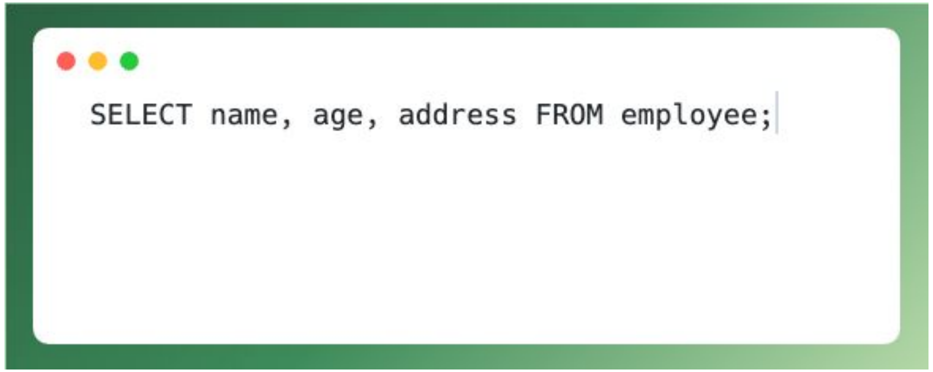
```
SELECT column1, column2, COUNT(*)
FROM table_name
WHERE condition_column = 'condition_value'
GROUP BY coulmn1, column2
HAVING COUNT(column1) > 1
ORDER BY column1 ASC;
```

## Clauses - From



```sql
SELECT column1, column2 FROM table_name;
```

SQL system stores multiple databases and tables. This table helps to display information in the application. Suppose an application requires a table from the system. The "FROM" operator gives a specific table. The "FROM" clause helps to get a specific table in the database.

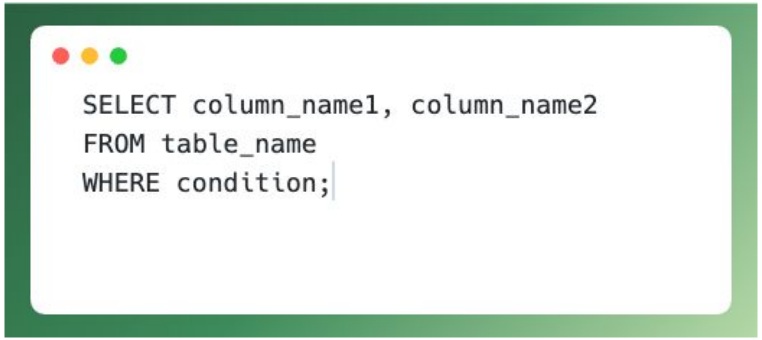


```sql
SELECT name, age, address FROM employee;
```

## Clauses - Where

WHERE clause applies a condition on the table field. The table displays conditional information of the columns and rows. It supports the insert, updates, retrieve, and delete operations. This clause helps to display a particular position of the table. WHERE clause helps to apply other clauses on the table field.

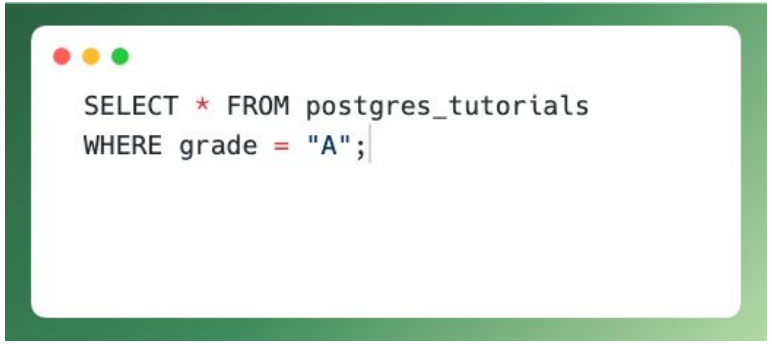
## Clauses - Where to Retrieve



```
SELECT column_name1, column_name2
FROM table_name
WHERE condition;
```

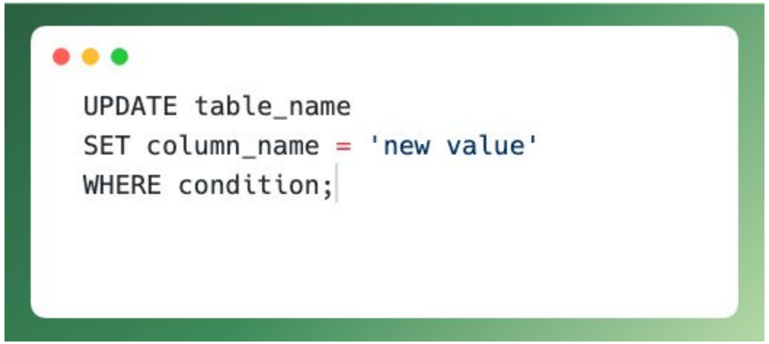
This clause comes with a condition of the table information. It requires a SQL basic expression to display data. WHERE clause syntax shows beside.

Execute the following query to retrieve data using the "WHERE" condition. Here, the condition applies to a "grade" column. The column is equal to an "A" grade



```
SELECT * FROM postgres_tutorials
WHERE grade = "A";
```

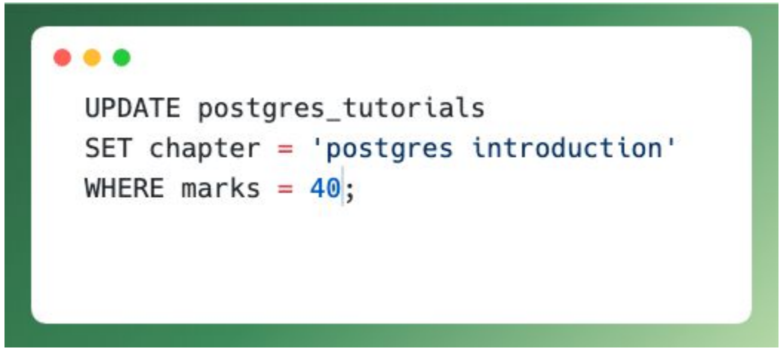
## Clauses - Where to Update



```
UPDATE table_name
SET column_name = 'new value'
WHERE condition;
```

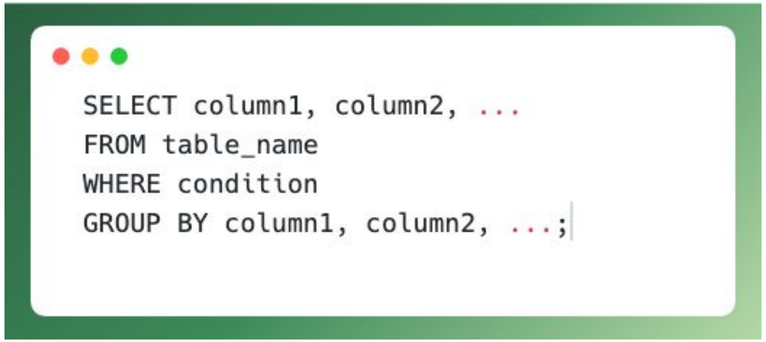
This clause comes with a condition of the table information. It requires a SQL basic expression to update data. WHERE clause syntax shows beside.

Execute the besides query to get the updated chapter name. Here chapter column updates using the "WHERE" condition. The condition uses a marks chapter where value exists 40.



```
UPDATE postgres_tutorials
SET chapter = 'postgres introduction'
WHERE marks = 40;
```

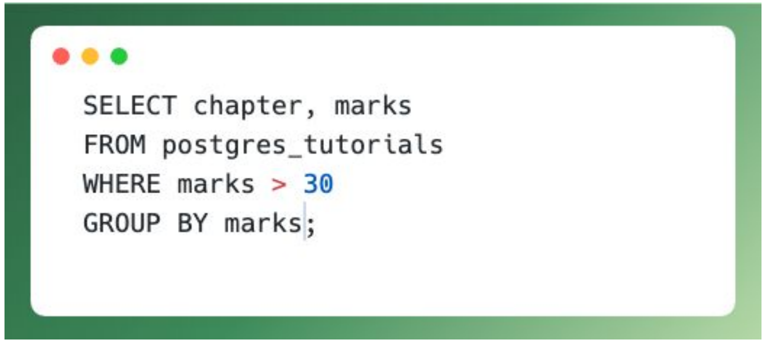
## Clauses - Group By to Retrieve Data



```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column1, column2, ...;
```

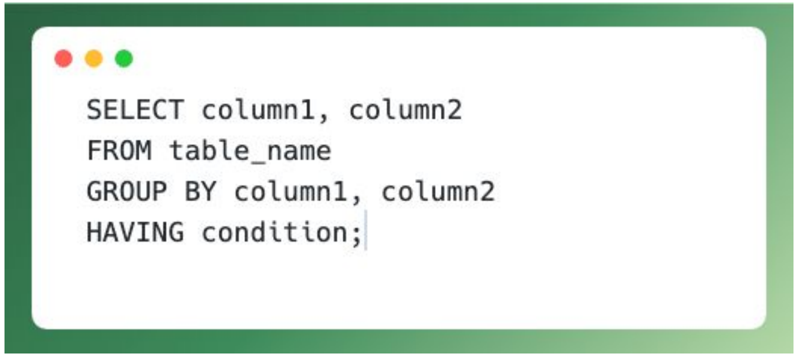
The "GROUP BY" clause collects multiple columns together and displays data.This clause supports the use of aggregation functions such as MIN, MAX, SUM, so on.It creates a group of a column and rows as per the requirement.  This clause works with the "WHERE" condition. But, the "WHERE" condition shows optional.

Execute the following query of the primary "Group BY" clause. The query uses a numerical column to create a group.



```
SELECT chapter, marks
FROM postgres_tutorials
WHERE marks > 30
GROUP BY marks;
```

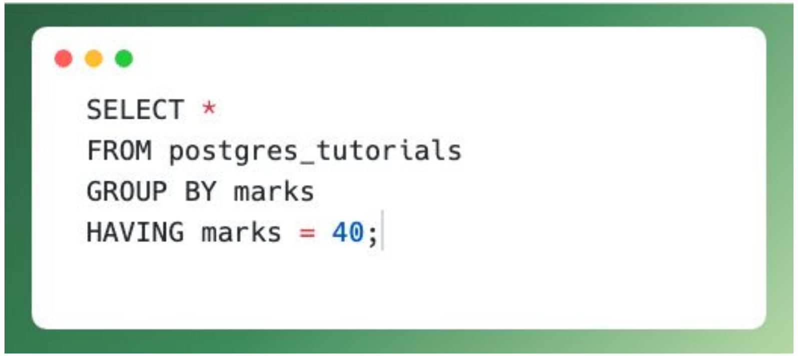
## Clauses - Having



```
SELECT column1, column2
FROM table_name
GROUP BY column1, column2
HAVING condition;
```

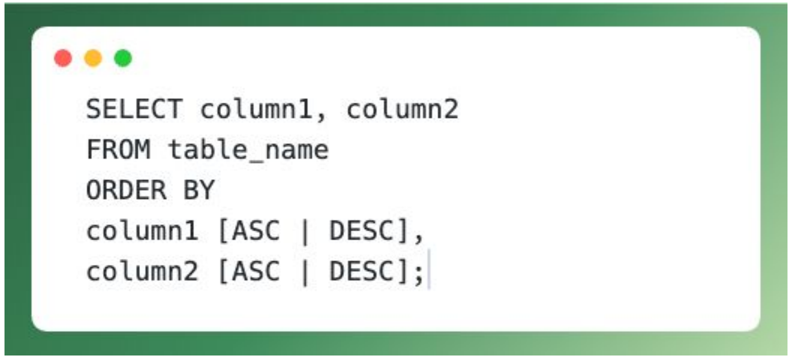
The "HAVING" clause comes with a "GROUP BY" clause. The "HAVING" clause is filtering data of specific rows. It provides a condition to the group of the row or aggregations. It works the same as a "where" clause after the "GROUP BY" clause.

Execute the following query to filter table data. Here, you search data using the "Having" condition with the "GROUP BY" clause. This example shows the use of a basic "having" query.



```
SELECT *
FROM postgres_tutorials
GROUP BY marks
HAVING marks = 40;
```

## Clauses - Order By



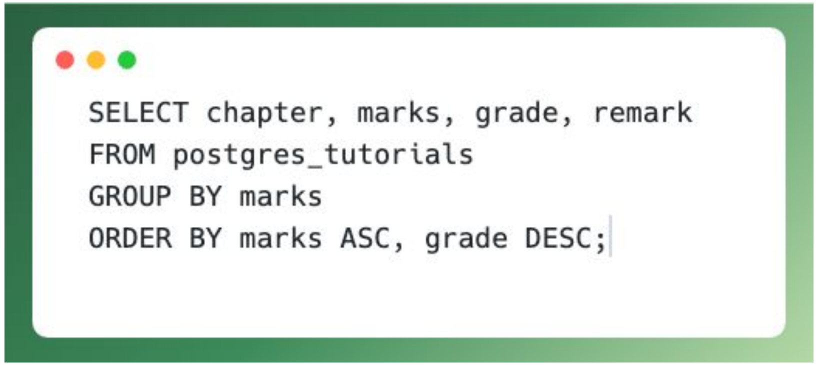
```
SELECT column1, column2
FROM table_name
ORDER BY
column1 [ASC | DESC],
column2 [ASC | DESC];
```

The "ORDER BY" clause helps to display the table field per order.

ASC: This keyword displays column data in ascending order.

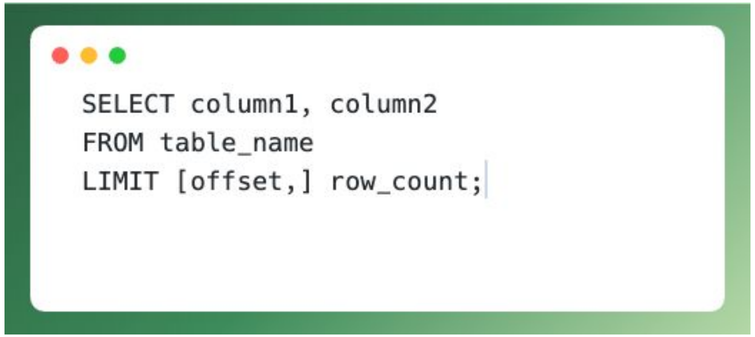
DESC: This keyword displays column data in descending order.

Execute the following query to filter table data. Here, you search data using the "Having" condition with the "GROUP BY" clause. This example shows the use of a basic "having" query.



```
SELECT chapter, marks, grade, remark
FROM postgres_tutorials
GROUP BY marks
ORDER BY marks ASC, grade DESC;
```

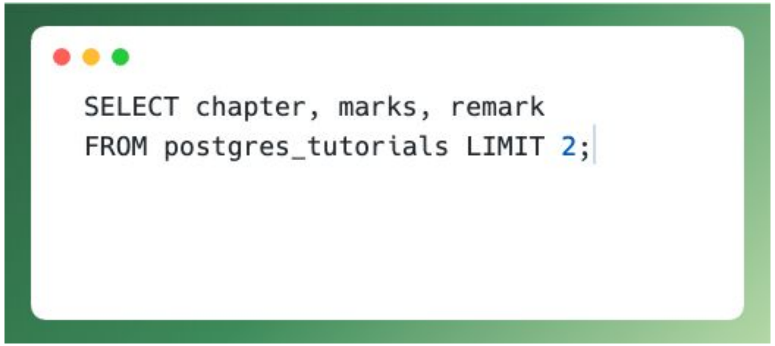
## Clauses - Limit



```
SELECT column1, column2
FROM table_name
LIMIT [offset,] row_count;
```

The "LIMIT" clause is displaying several outputs. This clause is showing information from the starting row in the table. The "LIMIT" clause is working with the "SELECT" statement. This clause can use the WHERE condition, but it is not necessary.

Execute the following query to limit the row result.



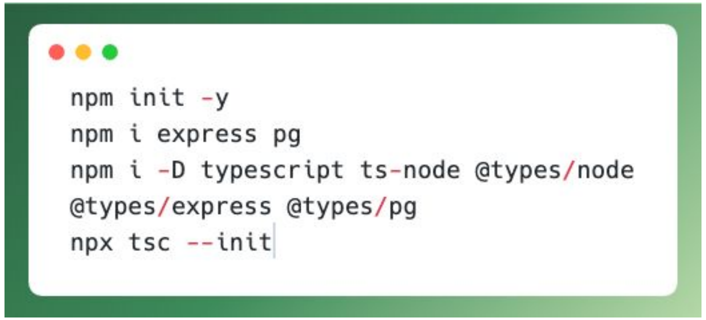
```
SELECT chapter, marks, remark
FROM postgres_tutorials LIMIT 2;
```

## Integrating PostgreSQL with NodeJs

- PostgreSQL package is a Node.JS driver for PostgreSQL database. It’s written in JavaScript, doesn’t require compiling & 100% MIT licensed.
- More info: https://www.npmjs.com/package/pg
- On project dir, install PostgreSQL package:
  - $ npm install pg

## Integrating PostgreSQL with NodeJs

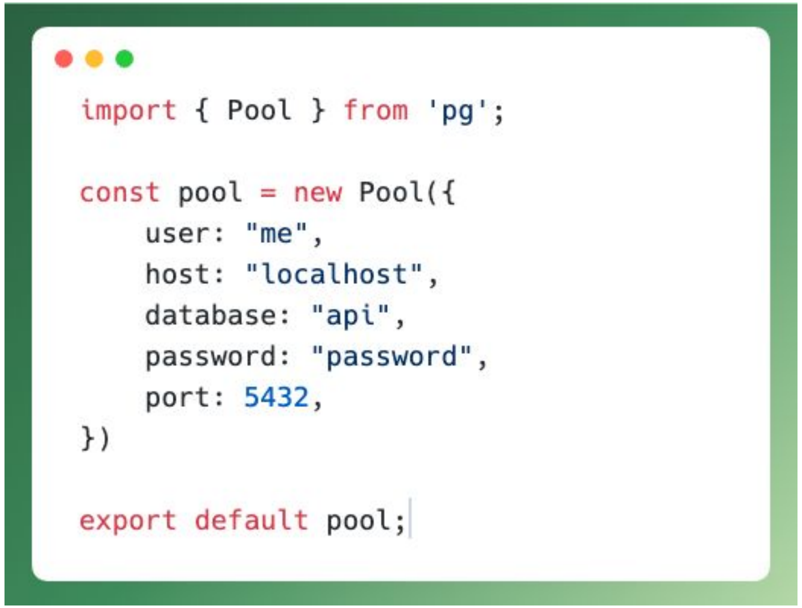
Initiate new project using:



```
npm init -y
npm i express pg
npm i -D typescript ts-node @types/node
@types/express @types/pg
npx tsc --init
```

Create /config/db.ts file and define your pg-pool and connection

in the projects.



```
import { Pool } from 'pg';

const pool = new Pool({
    user: "me",
    host: "localhost",
    database: "api",
    password: "password",
    port: 5432,
})

export default pool;
```

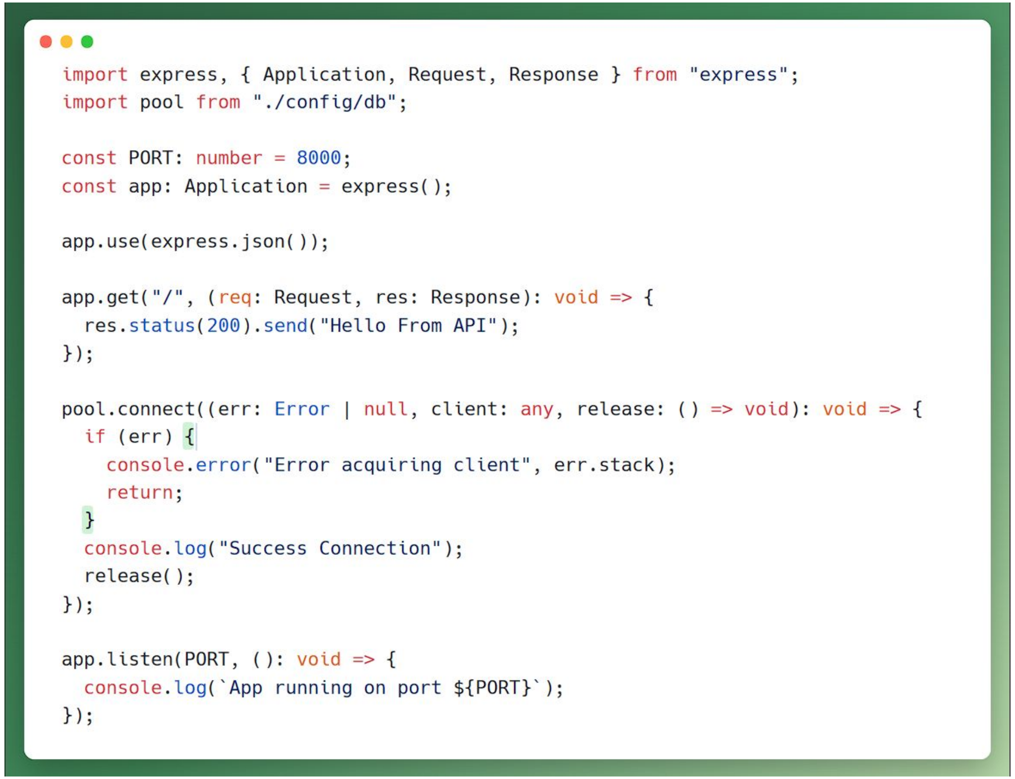
## Integrating PostgreSQL with NodeJs

After define your db connection at /config/db.ts,

Use pool.connect method to check your connection in index.ts file.

Write down check connection at index.ts.

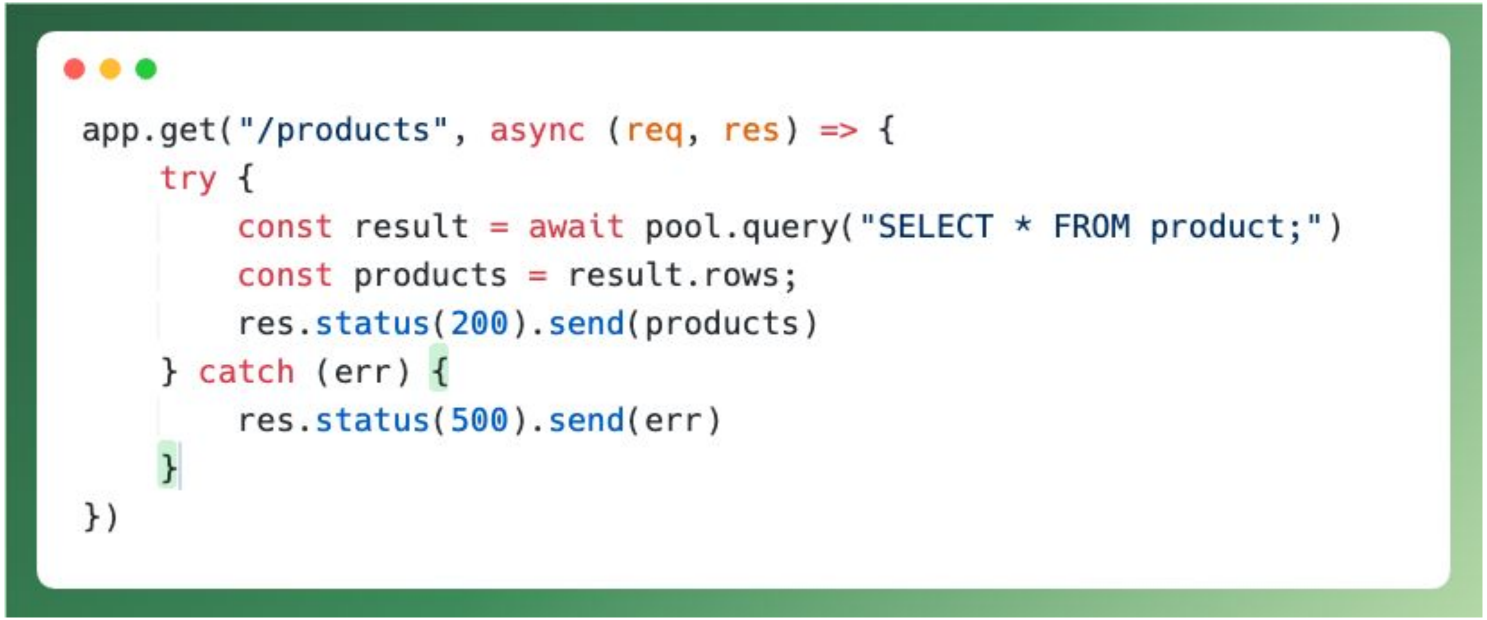
And run your script.



```
import express, { Application, Request, Response } from "express";
import pool from "./config/db";

const PORT: number = 8000;
const app: Application = express();

app.use(express.json());

app.get("/", (req: Request, res: Response): void => {
  res.status(200).send("Hello From API");
});

pool.connect((err: Error | null, client: any, release: () => void): void => {
  if (err) {
    console.error("Error acquiring client", err.stack);
    return;
  }
  console.log("Success Connection");
  release();
});

app.listen(PORT, (): void => {
  console.log(`App running on port ${PORT}`);
});
```

## Try to get data from PostgreSQL

After test connection success, write down route and middleware for get data from database.



```
app.get("/products", async (req, res) => {
    try {
        const result = await pool.query("SELECT * FROM product;")
        const products = result.rows;
        res.status(200).send(products)
    } catch (err) {
        res.status(500).send(err)
    }
})
```

## Exercise

Create REST API using ExpressJS with all HTTP request methods.

**Goal:**

Build a simple REST API using Express.js, store data in a PostgreSQL database.

**Notes:**

- Use fs module for file read/write
- Send responses in JSON
- Use proper status codes (200, 201, 404, etc.)
- Handle invalid ID or file errors

| Method | Endpoint | Description |
|---|---|---|
| GET | **/todos** | Get all todos |
| GET | **/todos/:id** | Get todo by ID |
| POST | **/todos** | Add new todo |
| PUT | **/todos/:id** | Update todo |
| DELETE | **/todos/:id** | Delete todo |

## Thank you