PostgreSQL Introduction

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#python-web

Data Management

When Do We Need a Database?

Storage vs. Management

SALES RECEIPT

Date: 07/16/2016
Order#:[00315]

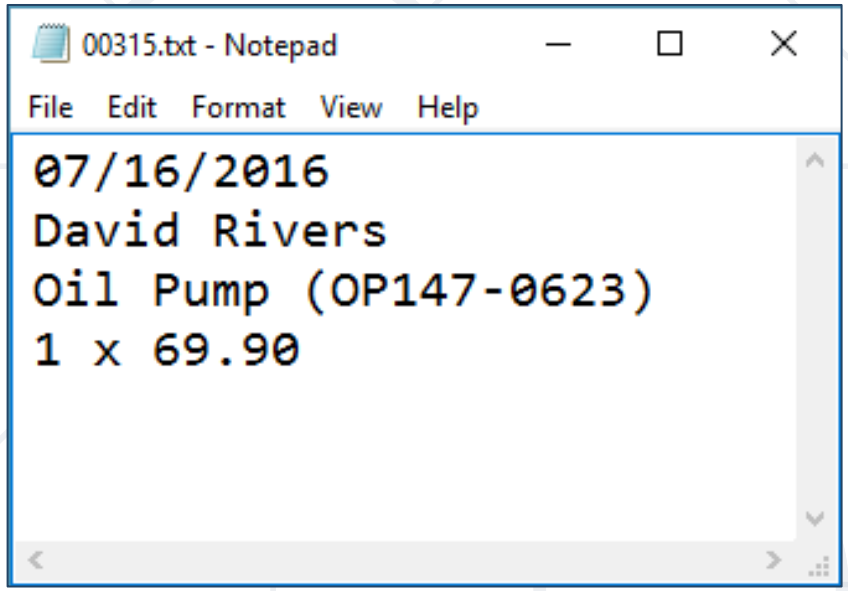
Customer: David Rivers
Product: Oil Pump
S/N: OP147-0623

Unit Price: **69.90**
Qty: **1**

Total: **69.90**

00315 – 07/16/2016
David Rivers
Oil Pump (OP147-0623)
1 x 69.90

Storage vs. Management (2)



Order#	Date	Customer	Product	S/N	Qty
00315	07/16/2016	David Rivers	Oil Pump	OP147-063	1

Database

- A database is an **organized** collection of **related** information
  - The user doesn't have **direct access** to the stored data
  - Access to data is usually **provided by a DBMS**

Database Table Elements

- The table is the main **building block** of any database

Column

CustomerID	FirstName	Birthdate	CityID
1	Brigitte	03/12/1975	101
2	August	27/05/1968	102
3	Benjamin	15/10/1988	103
4	Denis	07/01/1993	104

Row

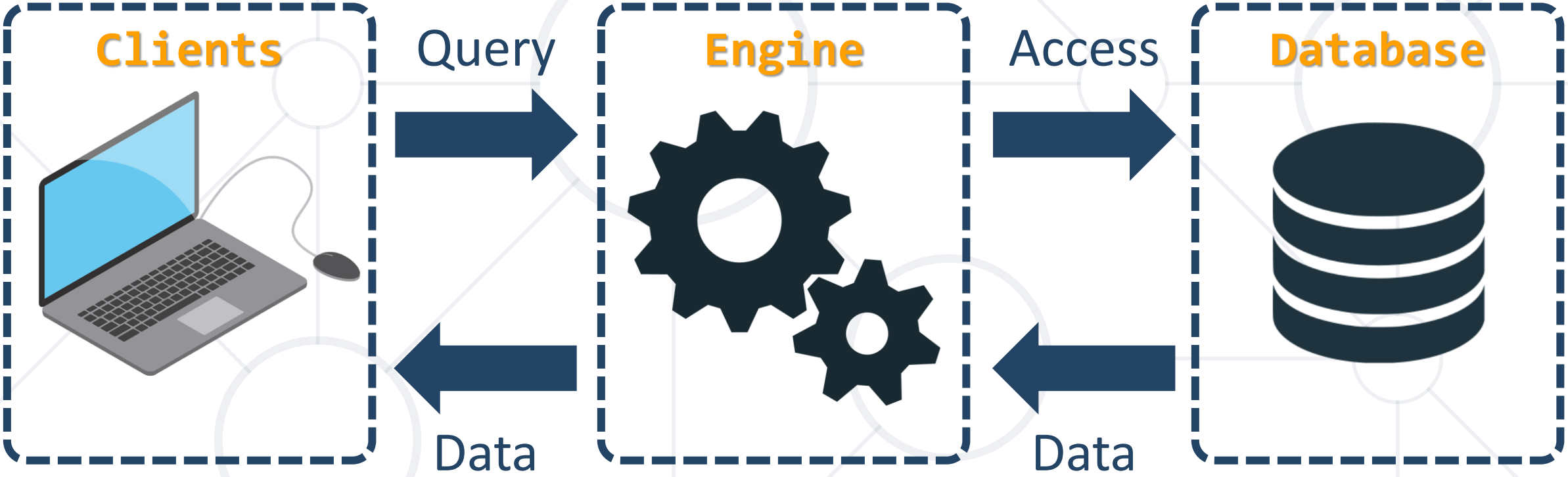
Cell

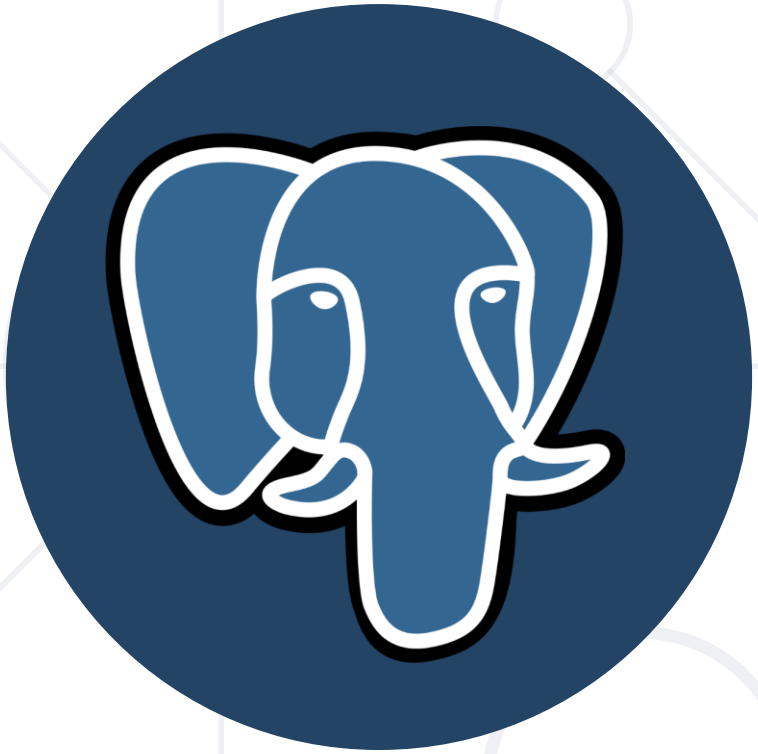
- Each **row** is called a **record** or **entity**
- Columns (**fields**) define the **type** of data they contain

Data Base Management System

- **D**ata **B**ase **M**anagement **S**ystem
  - **Provides tools** to define, manipulate, retrieve and manage data in a database
  - **Parses requests** from the user and takes the appropriate action

Database Engine Flow





PostgreSQL

What is PostgreSQL?

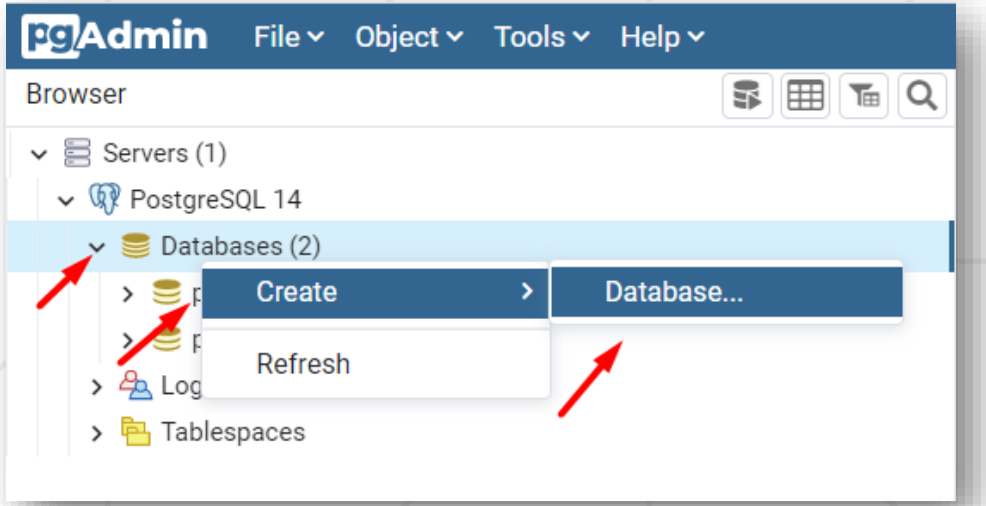
- **O**bject–**R**elational **D**atabase **M**anagement **S**ystem (**ORDBMS**)
- Widely used **open-source cross-platform** system

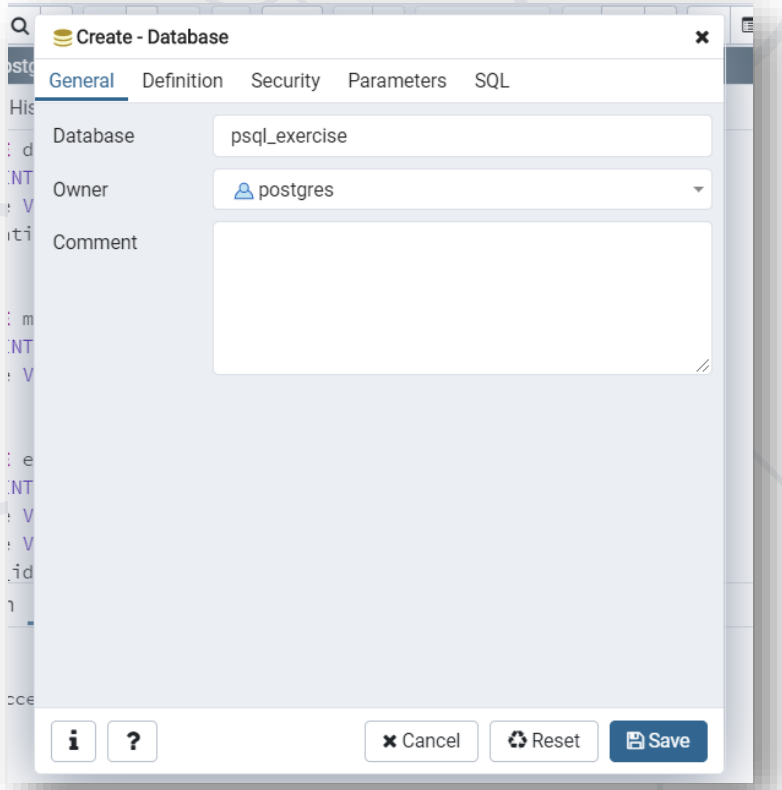
Rank			DBMS	Database Model	Score		
Jun 2022	May 2022	Jun 2021			Jun 2022	May 2022	Jun 2021
1.	1.	1.	Oracle	Relational, Multi-model	1287.74	+24.92	+16.80
2.	2.	2.	MySQL	Relational, Multi-model	1189.21	-12.89	-38.65
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	933.83	-7.37	-57.25
4.	4.	4.	PostgreSQL	Relational, Multi-model	620.84	+5.55	+52.32
5.	5.	5.	MongoDB	Document, Multi-model	480.73	+2.49	-7.49
6.	6.	↑ 7.	Redis	Key-value, Multi-model	175.31	-3.71	+10.06
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	159.19	-1.14	-7.85
8.	8.	8.	Elasticsearch	Search engine, Multi-model	156.00	-1.70	+1.29
9.	9.	↑ 10.	Microsoft Access	Relational	141.82	-1.62	+26.88
10.	10.	↓ 9.	SQLite	Relational	135.44	+0.70	+4.90

What makes PostgreSQL stand out?

- First DBMS that implements **multi-version concurrency control feature**
- Able to add **custom functions**
- Designed to be **extensible**
- Defining custom data types, plugins, etc.
- Very active community

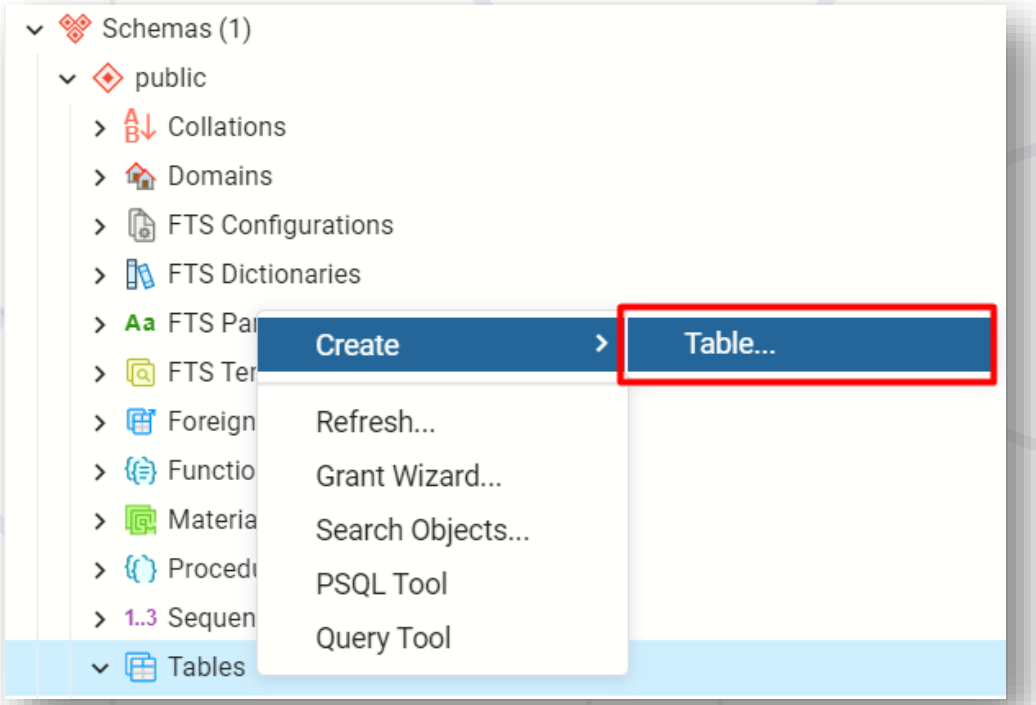
Create a New Database in pgAdmin 4

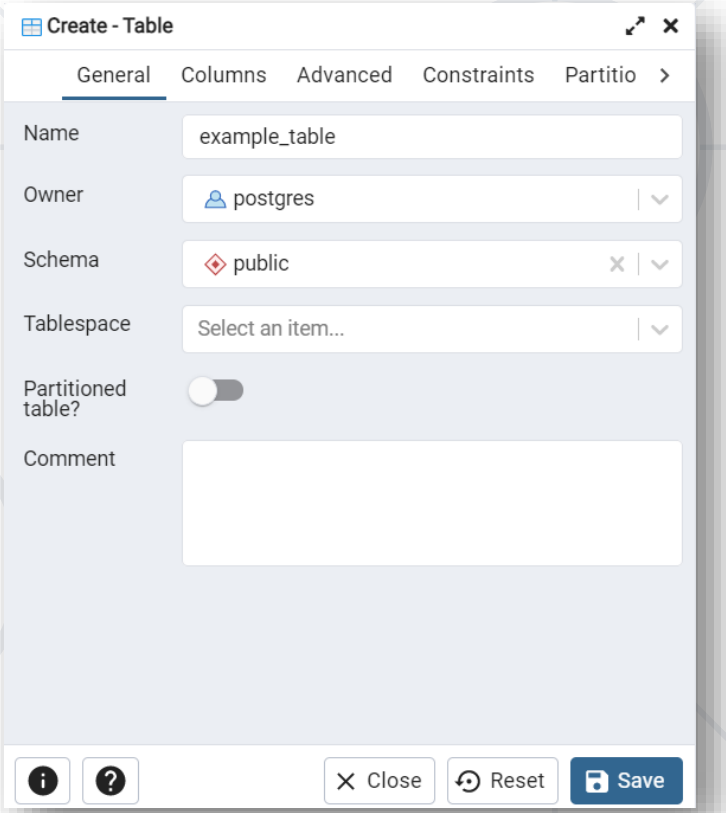




Create a New Table (1)

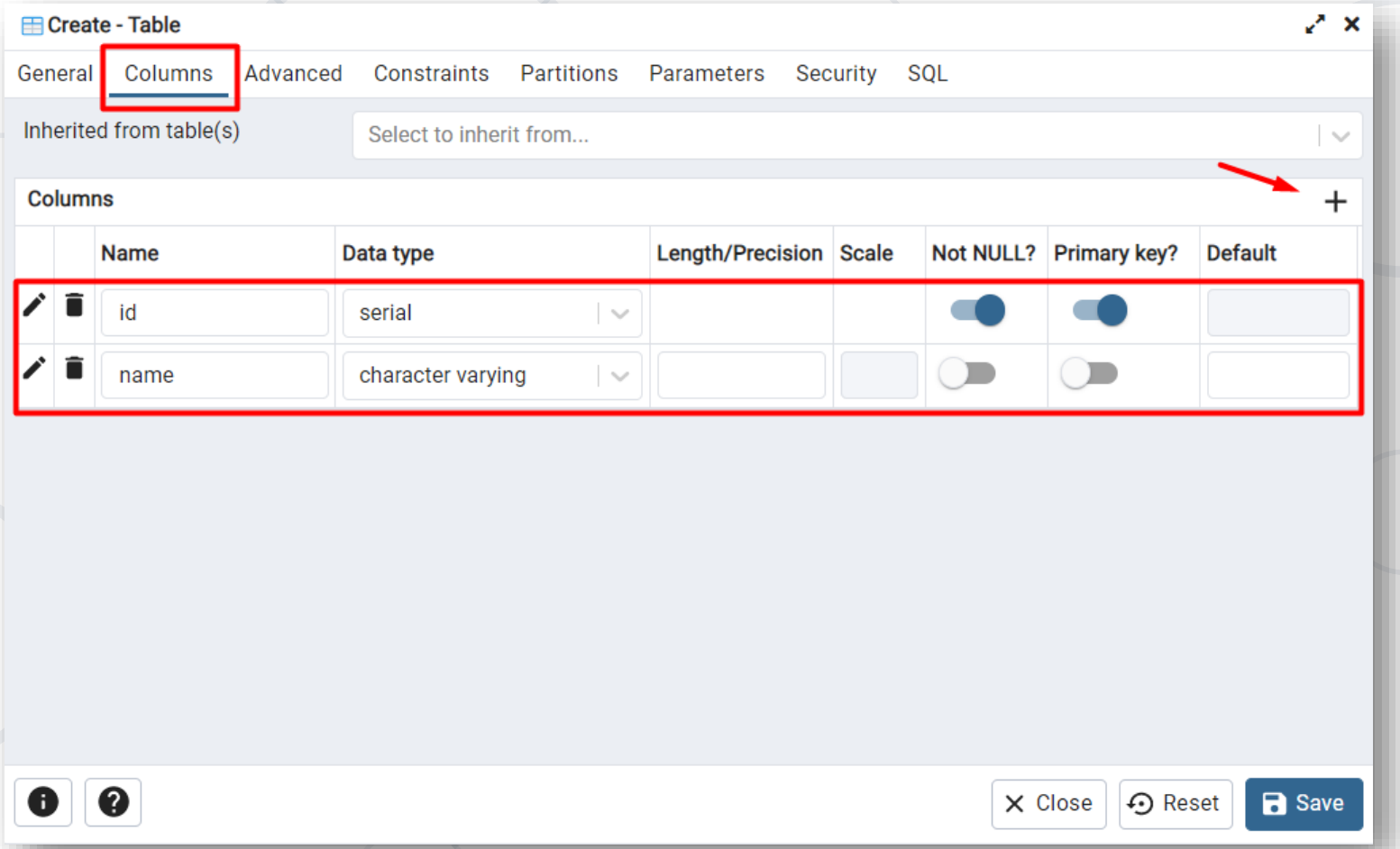
- Right click on the Database/ Schemas/ Tables





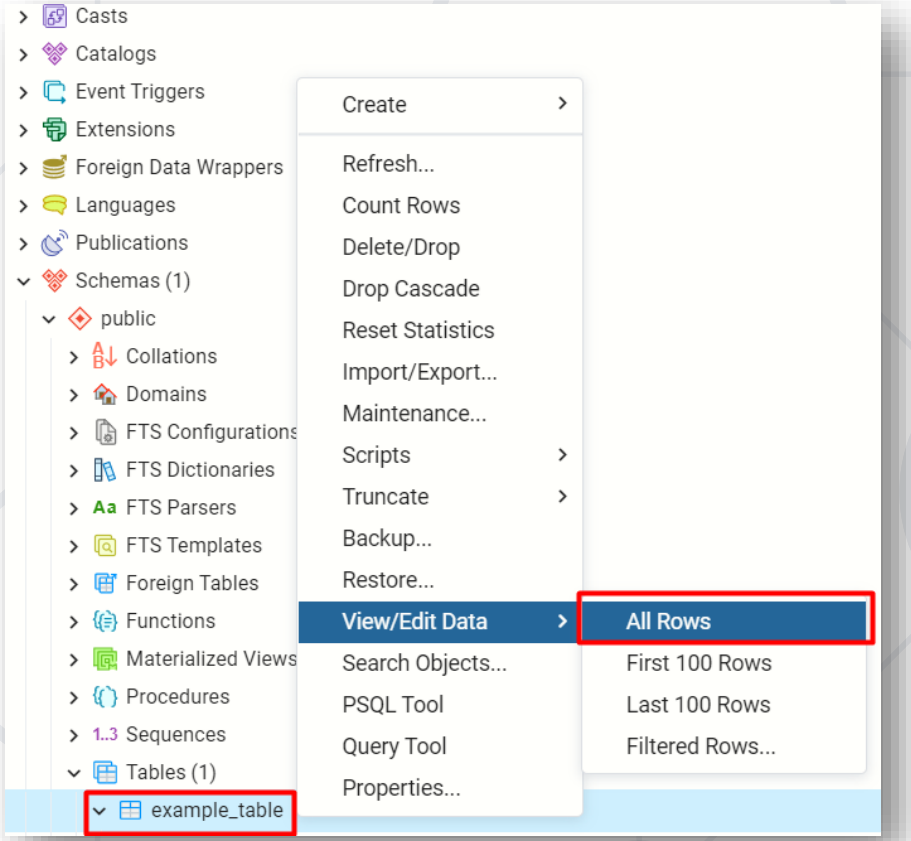
Create a New Table (2)

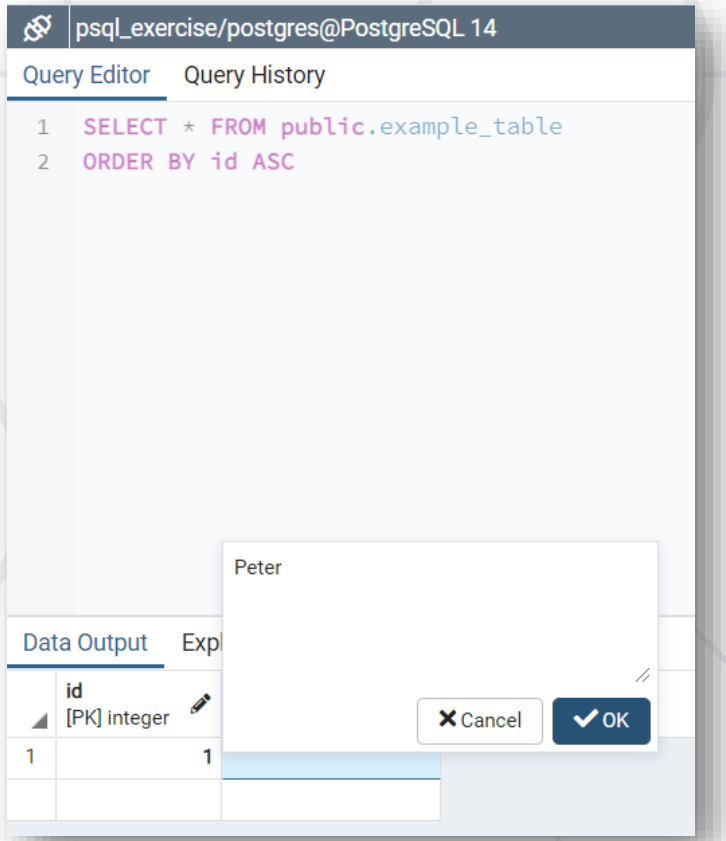
- Create columns in the table



View/ Edit Data

- Right click on the created table `example_table`





Structured Query Language

Structured Query Language (1)

- **Programming** language
- Designed for **managing** data in a **relational** database
  - Access **many records** with **one single command**
  - **Eliminates** the need to specify **how** to reach a record
- Developed at **IBM** in the early 1970s

Structured Query Language (2)

- Subdivided into several language elements
  - Queries
  - Clauses
  - Expressions
  - Predicates
  - Statements

Update clause

Expression

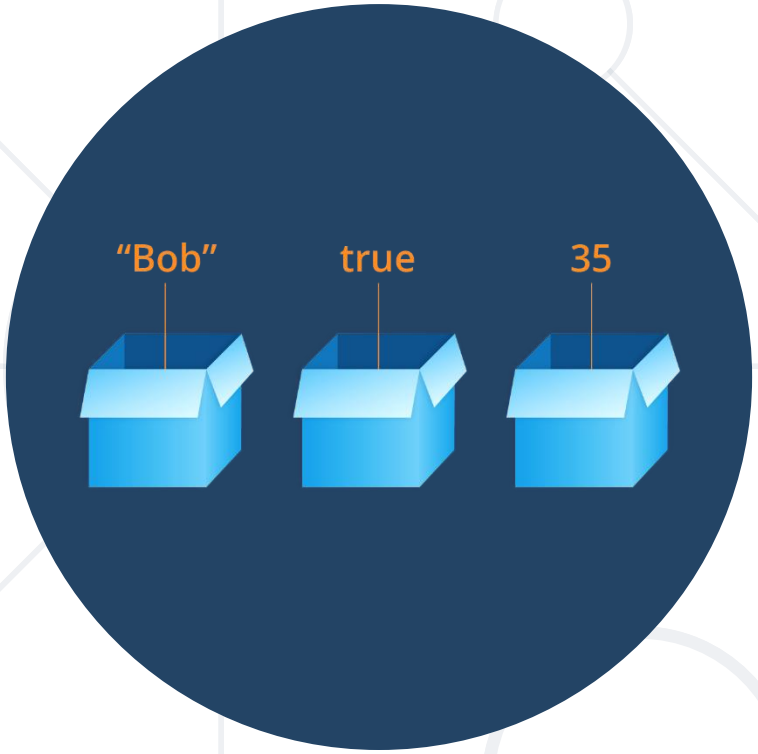
Statement

```
UPDATE employees
SET   salary = salary * 1.1
WHERE job_title = "Cashier";
```

Predicate

SQL vs NoSQL

- **SQL**:
  - Relational database management system
  - Predefined Schema
  - Suited for complex queries
  - Vertically scalable
- **NoSQL**:
  - Non-relational database system
  - Dynamic Schema
  - Suited for hierarchical data storage
  - Horizontally scalable



Data Types in SQL

String Types

- **CHARACTER/CHAR**[(M)]
  - Fixed-length e.g., CHAR(30)
  - CHAR without the length specifier (m) is the same as CHAR(1)
- **CHARACTER VARYING/VARCHAR**[(N)]
  - Variable-length with limit e.g., VARCHAR(30)
  - VARCHAR without (n) can store a string with unlimited length
- **TEXT**
  - Stores strings of any length

Numeric Data Types

- **Integer types**
  - SMALLINT, INTEGER/INT, BIGINT
- **Arbitrary Precision Numbers**
  - DECIMAL, NUMERIC
- **Floating-Point Types**
  - REAL, DOUBLE PRECISION
- **Serial Types**
  - SMALLSERIAL, SERIAL, BIGSERIAL

The type INTEGER/INT is the common choice

Recommended for storing quantities where exactness is required

Storing and retrieving a value might show a slight difference

Used for creating unique identifier columns

Column Properties

- Set **no repeating values** in the entire table

```
email VARCHAR (50) UNIQUE
```

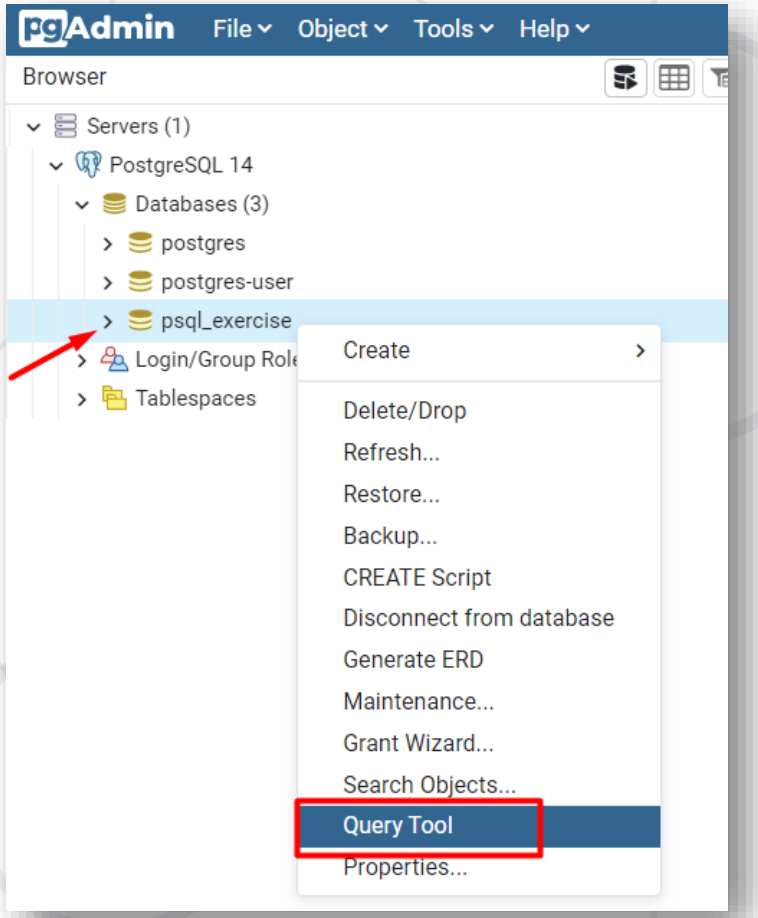
- If a value is **not specified**, use the default one

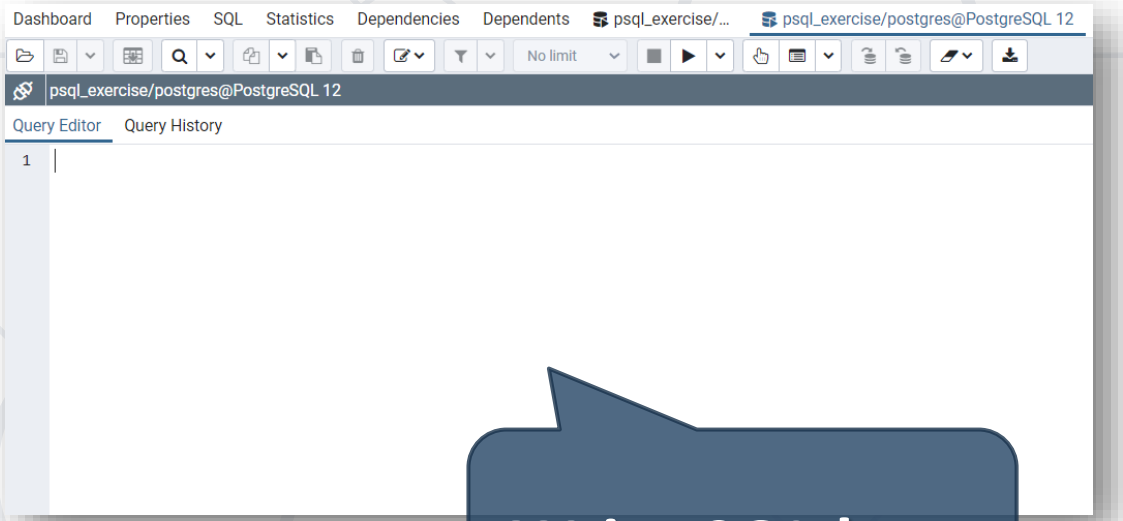
```
balance DECIMAL (10,2) DEFAULT 0
```

- Set a column that must **not assume a null value**

```
name VARCHAR (100) NOT NULL
```

Open the Query Tool





Write SQL here

Create Table Using SQL

```
CREATE TABLE department (
        dep_id SERIAL UNIQUE NOT NULL,
        dep_name VARCHAR (100) UNIQUE NOT NULL,
        dep_location VARCHAR (100) DEFAULT 'Sofia'
);
```

Column constraint

Column name

Data type

Table Relations

Relational Database Model in Action

Relationships

- **Relationships** between tables are based on interconnections: **PRIMARY KEY** / **FOREIGN KEY**

Primary key

Foreign key

towns

id	name	country_id
1	Sofia	1
2	Varna	1
3	Munich	2
4	Berlin	2
5	Moscow	3

Primary key

countries

id	name
1	Bulgaria
2	Germany
3	Russia

Relationships

Why Split Related Data?

Empty records

first	last	registered	email	email2
David	Rivers	05/02/2016	drivers@mail.cx	david@homedomain.cx
Sarah	Thorne	07/17/2016	sarah@mail.cx	*NULL*
M		5	walters_michael@mail.cx	*NULL*

Redundant information

order_id	date	customer	product	s/n	price
00315	07/16/2016	David Rivers	Oil Pump	OP147-0623	69.90
00315	07/16/2016	David Rivers	Accessory Belt	AB544-1648	149.99
00316	07/17/2016	Sarah Thorne	Wiper Fluid	WF000-0001	99.90
00317	07/18/2016	Michael Walters	Oil Pump	OP147-0623	69.90

Relationships (2)

- The **foreign key** is an **identifier** of a record located in another table (usually its primary key)
- By using relationships, we **avoid repeating data** in the database
- Relationships have multiplicity:
  - **One-to-many** – e.g., mountains / peaks
  - **Many-to-many** – e.g., student / course
  - **One-to-one** – e.g., example driver / car

One-to-One

Primary key

Foreign key

Primary key

cars

car_id	driver_id
1	166
2	102

drivers

driver_id	driver_name
166	…
102	…

Relation

One-to-Many/Many-to-One

Primary key

Mountains

mountain_id	name
1	Caucasus

Primary key

Foreign key

Peaks

peak_id	mountain_id
61	1
66	1

Relation

Many-to-Many

Primary key

employees

employee_id	name
1	…
40	…

Primary key

projects

project_id	name
4	…
24	…

Mapping table

employees_projects

employee_id	project_id
1	4
1	24
40	24

Junction Tables

students

student_id	name
1	Peter
2	George

classrooms

classroom_id	number
1	10
2	20

student_classroom

student_id	classroom_id	pk_student_classroom
1	1	11
2	2	22
1	2	12

Combination of the 2 ids

Connecting Tables

- Set a **primary key** to uniquely define a record

```
id BIGINT PRIMARY KEY
```

- Set a **foreign key** to reference the pk of another table

```
fk_column_name REFERENCES parent_table
```

```
fk_column_name REFERENCES parent_table (column_name)
```

```
FOREIGN KEY (first_fk, second_fk)
REFERENCES other_table (first_column, second_column)
```

On Delete Option

- When a record that holds a relation is **removed**, we have a few options:
  - **Disallow deleting** the referenced (parent) record
  - **Delete** the referenced (parent) record and all its references as well
  - **Delete** the referenced (parent) record but **keep** the references

DELETE CASCADE statement

- **Restrict** deleting the record

```
fk_column_name REFERENCES parent_table ON DELETE RESTRICT;
```

- **Automatically delete** rows referencing a deleted record

```
fk_column_name REFERENCES parent_table ON DELETE CASCADE;
```

- **Set NULL** to the foreign key columns when the referenced record is deleted

```
fk_column_name REFERENCES parent_table ON DELETE SET NULL;
```

Example

```
CREATE TABLE employee (
    employee_id BIGINT PRIMARY KEY,
    manager_id BIGINT,
    dep_id BIGINT,
    name VARCHAR (100),
    start_date DATE NOT NULL DEFAULT CURRENT_DATE,
    manager_id REFERENCES manager ON DELETE SET NULL,
    dep_id REFERENCES department ON DELETE RESTRICT
);
```

```sql
CREATE TABLE people (
  id INT NOT NULL,
  email VARCHAR NOT NULL,
  first_name VARCHAR(50),
  last_name VARCHAR(50)
);
```

Basic SQL Commands

Create, Read, Update, Delete

Create a Table Using SQL

Table name

```
CREATE TABLE people
(
  id SERIAL PRIMARY KEY,
  email VARCHAR (50) NOT NULL,
  first_name VARCHAR (50),
  last_name VARCHAR (50)
);
```

Inserting Data Using SQL

- The SQL **INSERT** command

```
INSERT INTO towns VALUES (33, 'Paris');
```

Values for all columns

```
INSERT INTO projects(name, start_date)
     VALUES ('Reflective Jacket', NOW());
```

Specify columns

- **Bulk data** can be recorded in a single query, separated by comma

```
INSERT INTO employees_projects
     VALUES (229, 1),
            (229, 2),
            (229, 3), …
```

Problems: Create and Insert

- **Create** new Database "game_bar"
- **Create Tables**:
  - **employees** – id, first_name, last_name
  - **categories** – id, name
  - **products** – id, name, category_id
- **Insert** Data:
  - Populate the **employees** table with 3 test values

Retrieve/Read Records Using SQL

- Get **all** information from a table

```
SELECT * FROM people;
```

* for all

Table name

- You can **limit** the columns and number of records

```
SELECT first_name, last_name FROM people LIMIT 5;
```

List of columns

Number of records

Filtering the Selected Rows

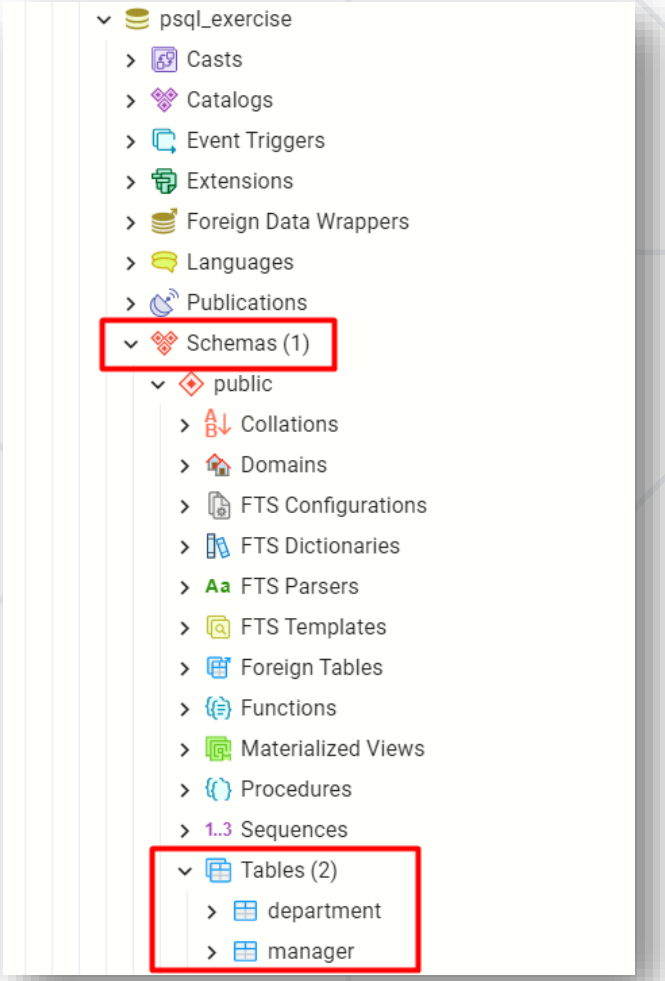
- You can filter rows by specific conditions using the **WHERE** clause

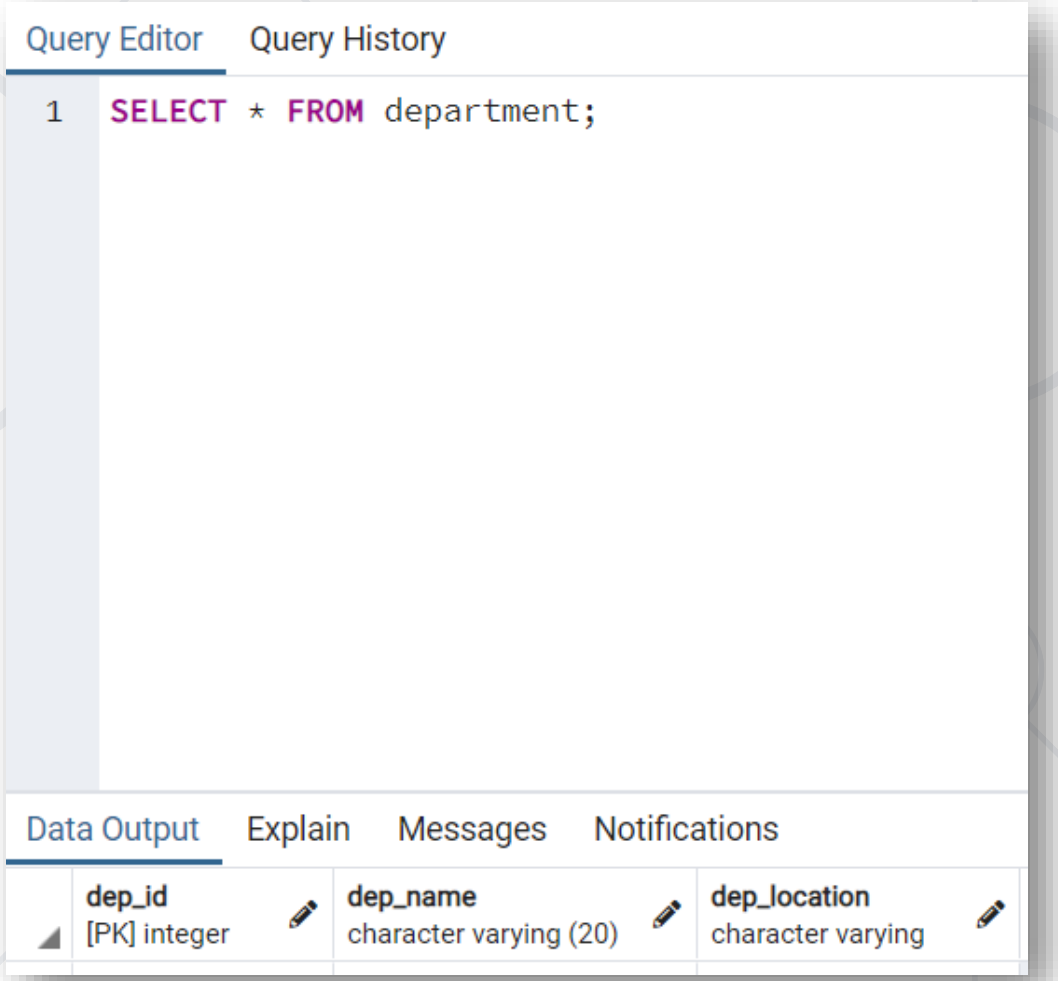
```
SELECT last_name, department_id
FROM employees
WHERE department_id = 1;
```

- **Logical** and **comparison operators** can be used for better control

```
SELECT last_name, salary
FROM employees
WHERE salary >= 10000 AND salary <= 20000;
```

Example


psql_exercise
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Procedures
Sequences
Tables (2)
department
manager


Query Editor
Query History
1 SELECT * FROM department;
Data Output
Explain
Messages
Notifications
dep_id
[PK] integer
dep_name
character varying (20)
dep_location
character varying

Updating Data

- The SQL **UPDATE** command

New values

```
UPDATE employees
   SET last_name = 'Brown'
 WHERE employee_id = 1;
```

```
UPDATE employees
   SET salary = salary * 1.10,
       job_title = CONCAT('Senior',' ', job_title)
 WHERE department_id = 3;
```

- Note: If you **left out the WHERE** clause, **all rows** in the table would be updated

Example: Update Employees Salary

- **Increase** all employees' salaries whose **job_title** is "**Manager**" with **100**

```
UPDATE employees
SET salary = salary + 100
WHERE job_title = 'Manager';
SELECT salary
FROM employees;
```

Altering Tables Using SQL (1)

- A table can be changed using the keywords **ALTER TABLE**

```
ALTER TABLE employees;
```

Table name

- Add new column

```
ALTER TABLE employees
ADD COLUMN salary DECIMAL;
```

Column name

Data type

Altering Tables Using SQL (2)

- Delete existing column

```
ALTER TABLE people
DROP COLUMN full_name;
```

Column name

- Modify data type of existing column

```
ALTER TABLE people
ALTER COLUMN email TYPE VARCHAR(100);
```

Column name

New data type

Problems: Alter Tables

- **Alter** table
  - Add a new column – "middle_name" to the "employees" table
- Adding **Constraints**
  - Make "category_id" **foreign key** linked to "id" in the "categories" table
- **Modifying** Columns
  - Change the property "VARCHAR(50)" to "VARCHAR(100)" to the "middle_name" column in "employees" table

Dropping and Truncating

- To delete all the entries in a table

```
TRUNCATE TABLE employees;
```

Table name

- To drop a table - delete data and structure

```
DROP TABLE employees;
```

Table name

- To drop entire database

```
DROP DATABASE soft_uni;
```

Database name

Example: Delete from Table

- **Delete** all employees from the "**employees**" table who are in department **2 or 1**

Delete Data

OR Condition

```
DELETE FROM employees
WHERE department_id = 1
OR department_id = 2;

SELECT * FROM employees
```

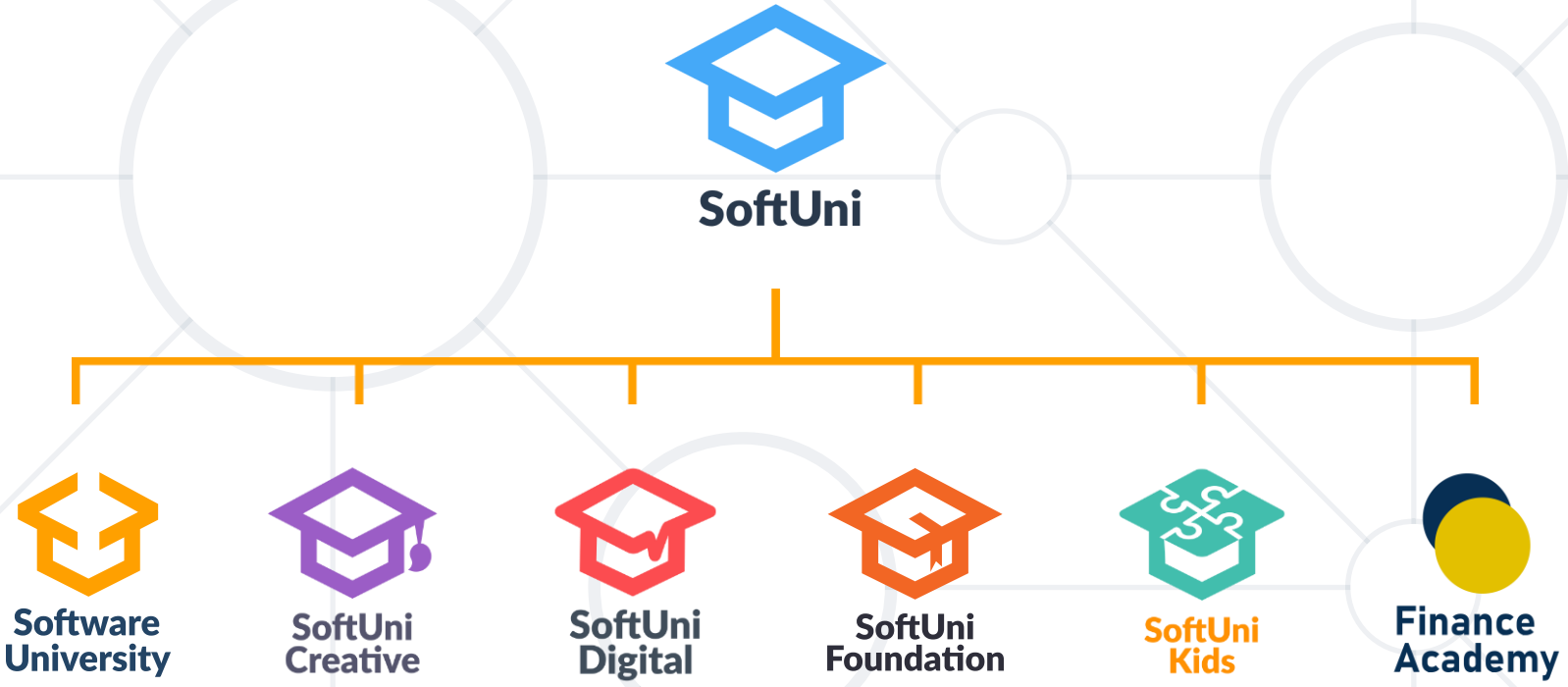
Live Demo

Writing SQL in pgAdmin 4

Summary

- Data Management
- Data Engine
- Table Relations
- Structured Query Language
- PostgreSQL
- Data Types
- Basic SQL Queries

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

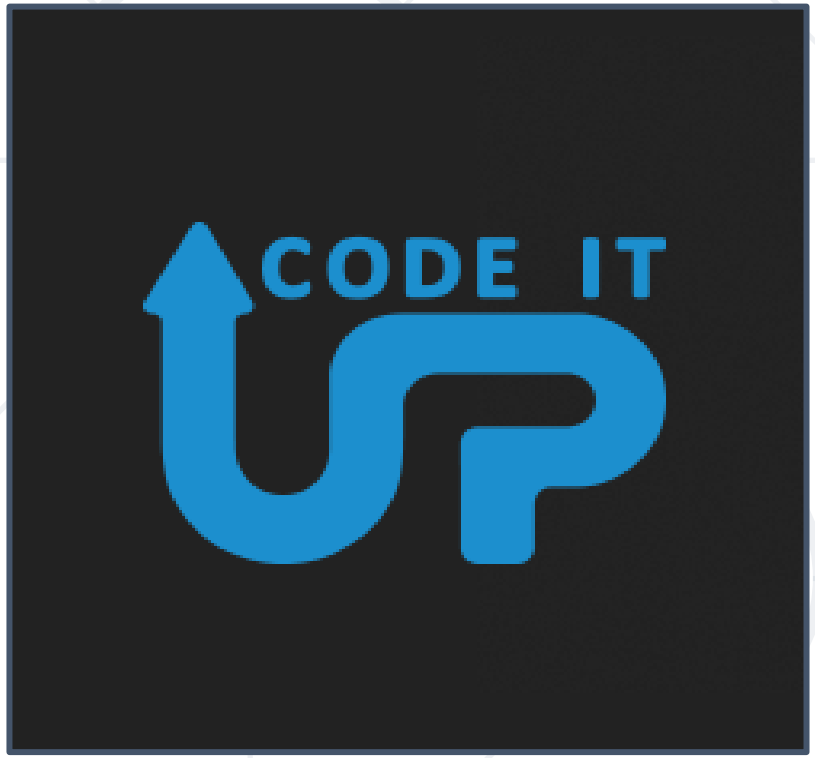
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg