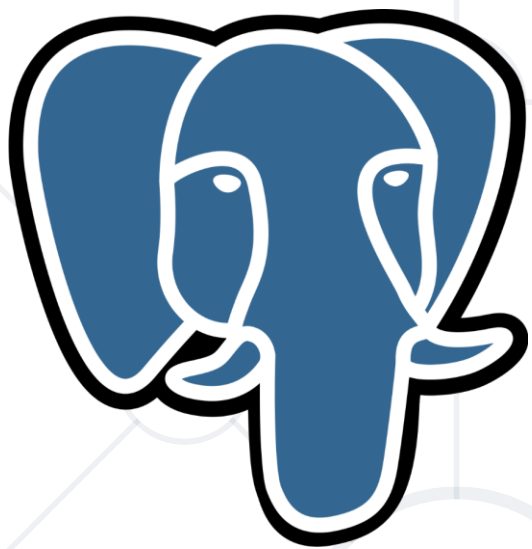


# Basic CRUD

## Data Manipulation



**SoftUni Team**

**Technical Trainers**



**SoftUni**



**Software University**

<https://softuni.bg>

sli.do

#python-db

# Table of Contents

1. Lexical structure in pgSQL
2. Retrieving data
3. Data manipulation
  - Create, update, delete
4. Views






# Lexical Structure in PostgreSQL

# Lexical Structure in PostgreSQL (1)

- Keywords and unquoted table/column names are **case insensitive**



```
CREATE TABLE CUSTOMER();  
create table customer();  
CrEaTe tAbLe CuStOmEr();
```

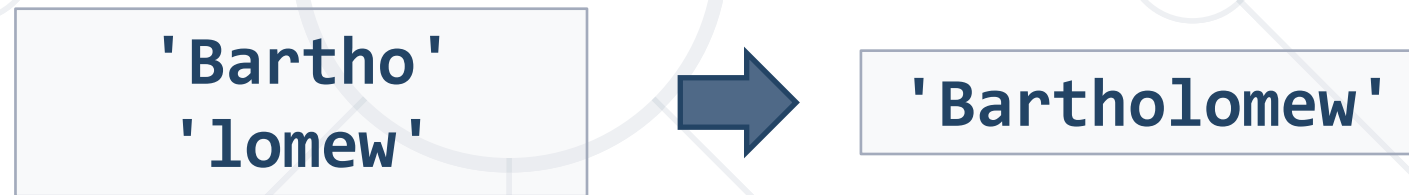
These are  
equivalent

- The convention is to write the **keywords in uppercase** and the **unquoted names in lowercase**

```
CREATE TABLE customer();
```

# Lexical Structure in PostgreSQL (2)

- **String literals** are enclosed in [ ' ](single quotes)
  - String separated by one or more **newlines** can be **concatenated as one string**



- **Table** and **column** names containing **special symbols** use [ " ] (double quotes)
  - Allows constructing names that would otherwise not be possible e.g., "**select**", "**Full Name**"
  - Makes them **case-sensitive**



# Comments in PostgreSQL

- Adding a comment on a **single line**

```
-- comment
```

```
CREATE DATABASE db_name; -- comment
```

- Adding a comment on **multiple lines** or anywhere **in the SQL statement**

```
CREATE TABLE /* comment */ ();
```





# Retrieving Data

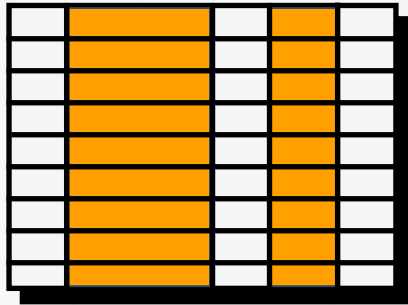
Using SQL SELECT



# Capabilities of SQL SELECT

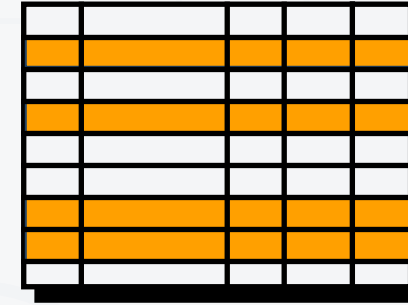
## Projection

Take a subset of the columns




## Selection

Take a subset of the rows




## Join

Combine tables by  
some column

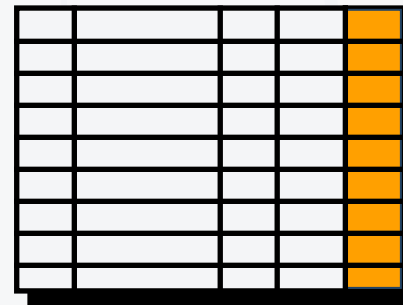



Table 1




Table 2

# Retrieve/Read Records Using SQL (1)

- Get **all** information from a table

\* for all

Table name

```
SELECT * FROM clients;
```



id	first_name	last_name	room_id
1	Peter	Petrov	1
2	George	Georgiev	2
3	Mariya	Marieva	28
...	...	...	...

# Retrieve/Read Records Using SQL (2)

- Get **specific** columns from a table

List of columns

```
SELECT first_name, last_name FROM clients;
```



first_name	last_name
Peter	Petrov
George	Georgiev
Mariya	Marieva
...	...

- Aliases **rename** a table or a column **heading**

Display Name

```
SELECT p.first_name AS "First Name"  
FROM clients AS p;
```



First Name
Peter
George
...

- Hint: use it to **shorten fields** or **clarify abbreviations**

# Concatenation Operator

- You can **concatenate** column records using the **||** operator

```
SELECT first_name || ' ' || last_name AS "Full Name"  
FROM clients;
```



Full Name
Peter Petrov
George Georgiev
...

# Problem: Select Employee Information

- Write a query to select all employees  
**id, first\_name and last\_name (as Full Name),  
job\_title (as Job Title)**
  - Concatenate fields **first\_name** and **last\_name** into **'Full Name'**
  - Display column **job\_title** as **'Job Title'**

```
SELECT id,  
       first_name || ' ' || last_name  
         AS "Full Name",  
       job_title AS "Job Title"  
FROM employees;
```

# Limiting the Selected Rows

- Retrieve the first N rows from a table
  - Could be returned fewer, if the query itself yields fewer rows

```
SELECT id, first_name FROM clients LIMIT 2;
```



id	first_name
1	Peter
2	George



# Sorting with ORDER BY

- Sort rows with the **ORDER BY** clause
  - ASC**: ascending order, default
  - DESC**: descending order

```
SELECT last_name, salary  
FROM employees  
ORDER BY salary;
```

```
SELECT last_name, salary  
FROM employees  
ORDER BY salary DESC;
```

last_name	salary
Johnson	880
Smith	900
Petrov	990
...	...

last_name	salary
Petrov	2100
Jackson	1800
Ivanov	1600
...	...



# **Filtering Selected Rows**

Using SQL WHERE Clause

# Filtering the Selected Rows

- Eliminate duplicate results

```
SELECT DISTINCT first_name FROM employees;
```

- Eliminate duplicate results based on the combination of values

```
SELECT DISTINCT first_name, last_name FROM employees;
```

- Eliminate duplicate results based on the first column

```
SELECT DISTINCT ON (first_name) first_name, last_name  
FROM employees;
```


# Filtering the Selected Rows

- Filter rows by specific conditions

```
SELECT id, first_name, last_name  
FROM employees  
WHERE id <= 2;
```

Clause

Comparison  
Operator



	id	first_name	last_name
1		John	Smith
2		John	Johnson

# Comparison Operators

Description	Operator
Equal to	=
Different from	!=, <>
Greater than	>
Greater than or equal to	>=
Less than	<
Less than or equal to	<=

# Problem: Employees Filtered and Ordered

- Select all employees (**id**, **first\_name** and **last\_name** (as **full\_name**), **job\_title**, **salary**)
  - Whose salaries are higher than 1000.00
  - Ordered by **id**
  - Concatenate fields **first\_name** and **last\_name** into **full\_name**

# Solution: Employees Filtered and Ordered

```
SELECT id,  
       first_name || ' ' || last_name  
       AS full_name,  
       job_title, salary  
FROM employees  
WHERE salary > 1000.00  
ORDER BY id;
```

- To allow the existence of multiple conditions

```
SELECT last_name FROM employees  
WHERE salary = 900 AND first_name = 'John';
```

- To reverse the meaning of the logical operator

```
SELECT last_name FROM employees  
WHERE NOT salary = 900;
```

- To combine multiple conditions

```
SELECT last_name FROM employees  
WHERE salary = 900 OR salary = 1100;
```



- Conditions can be combined using **brackets**

```
SELECT last_name, salary FROM employees  
WHERE NOT (salary = 900 OR salary = 1100);
```

- Add **comparison operators** for better control

```
SELECT last_name, salary FROM employees  
WHERE salary >= 900 AND salary <= 2100;
```

- Use the BETWEEN operator to **specify a range** (inclusive)

```
SELECT last_name, salary FROM employees  
WHERE salary BETWEEN 900 AND 2100;
```

- Use **IN** to specify a set of values:

```
SELECT first_name, last_name  
FROM employees  
WHERE salary IN (2100, 1100, 900, 880);
```

- Use **NOT IN** to specify a set of values:

```
SELECT first_name, last_name  
FROM employees  
WHERE salary NOT IN (2100, 1100, 900, 880);
```

# Problem: Employees by Multiple Filters

- Select all **employees**
  - Whose **salaries** are higher than or equal to **1000.00**
  - Their **department id** is **4**
  - Ordered by **id**

# Solution: Employees by Multiple Filters

```
SELECT * FROM employees
WHERE salary >= 1000.00
      AND department_id = 4
ORDER BY id;
```

- **NULL** is a special value that means missing value
  - Not the same as **0** or a **blank space**
- Checking for **NULL** values

```
SELECT first_name, room_id FROM clients  
WHERE last_name = NULL;
```

This is always **false**!

```
SELECT first_name, room_id FROM clients  
WHERE last_name IS NULL;
```

```
SELECT first_name, room_id FROM clients  
WHERE last_name IS NOT NULL;
```



# Data Manipulation

Using SQL INSERT, UPDATE, DELETE

# Inserting Data (1)

- Insert a record in a table

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

id	name
33	Paris

- Insert data into specific columns of a table

```
INSERT INTO towns(name) VALUES ('Sofia');
```

id	name
33	Paris
34	Sofia

- Bulk data can be recorded in a single query

```
INSERT INTO towns(name)  
VALUES ('London'),  
      ('Rome');
```

id	name
33	Paris
34	Sofia
35	London
36	Rome

# Inserting Data (2)

- You can use existing records to create a **new table**

```
CREATE TABLE customer_data  
AS SELECT id, last_name, room_id  
FROM clients;
```

New table name

Existing source

- Or into an existing table

```
INSERT INTO projects(name, start_date)  
SELECT name || ' Restructuring', '2023-01-01'  
FROM departments;
```

List of columns



# Problem: Insert New Employees

- Insert new records into the table **employees**
  - Insert all values with a single query

first name	last name	job title	department id	salary
Samantha	Young	Housekeeping	4	900
Roger	Palmer	Waiter	3	928.33

- Retrieve the information from table **employees** to check the new entries

# Solution: Insert New Employees

```
INSERT INTO employees
  (first_name, last_name, job_title, department_id,
   salary)
VALUES
  ('Samantha', 'Young', 'Housekeeping', 4, 900),
  ('Roger', 'Palmer', 'Waiter', 3, 928.33);
```

```
SELECT * FROM employees;
```

- The SQL **UPDATE** command

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

New values

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

Multiple updates

- Note: If you **skip the WHERE** clause, **all rows** in the table will be updated

# Problem & Solution: Update Employees Salary

- Increase all employees' salaries whose
  - **job\_title** is "Manager"
  - Increase with **100**

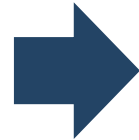
```
UPDATE employees  
SET salary = salary + 100  
WHERE job_title = 'Manager';
```

```
SELECT * FROM employees  
WHERE job_title = 'Manager';
```

# Example: Update Clients


- Set **all missing** last names to "Unknown" in table `clients`
  - Hint: Missing names have their `last_name` set to **NULL**

first_name	last_name
Kate	null
Steven	null
Christo	null
...	...



first_name	last_name
Kate	Unknown
Steven	Unknown
Christo	Unknown
...	...

# Solution: Update Clients



```
UPDATE clients
  SET last_name = 'Unknown'
 WHERE last_name IS NULL;
```

Filters only records  
with no values

- Deleting **specific rows** from a table

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

Condition

- Note: If you **skip the WHERE** clause, **all rows** in the table will be deleted
- Delete all rows from a table
  - **TRUNCATE** works faster than **DELETE**

```
TRUNCATE TABLE employees;
```

# Problem & Solution: Delete from Table

- **Delete** all employees from the "**employees**" table
  - Who are in department **2** or department **1**
  - Select all records from table "**employees**" to check the result

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

```
SELECT * FROM employees  
ORDER BY id;
```





# Views

CREATE VIEW ... AS

# Definition and Usage

- Views are **virtual tables** made from other tables, views, or joins between them
- When you create a view, you basically **create a query and assign a name** to the query
- Usage
  - To **simplify** writing complex queries
  - To **limit access** to data for certain users



Table 1		
Column 1	Column 2	Column 3

Table 2		
Column 1	Column 2	Column 3



v_table1_table2		
Column 1	Column 2	Column 3

- Get employee names and salaries, sorted by department id

```
CREATE VIEW hr_result_set AS
SELECT
    employees.first_name || ' ' || employees.last_name
    AS "Full Name",
    employees.salary
FROM employees ORDER BY department_id;
```

```
SELECT * FROM hr_result_set;
```

# Problem & Solution: Top-paid Employee

- Select all information about the **top-paid employee**

```
CREATE VIEW top_paid_employee AS  
  SELECT * FROM employees  
  ORDER BY salary DESC LIMIT 1;
```

Sorting column

Greatest value first

```
SELECT * FROM top_paid_employee;
```

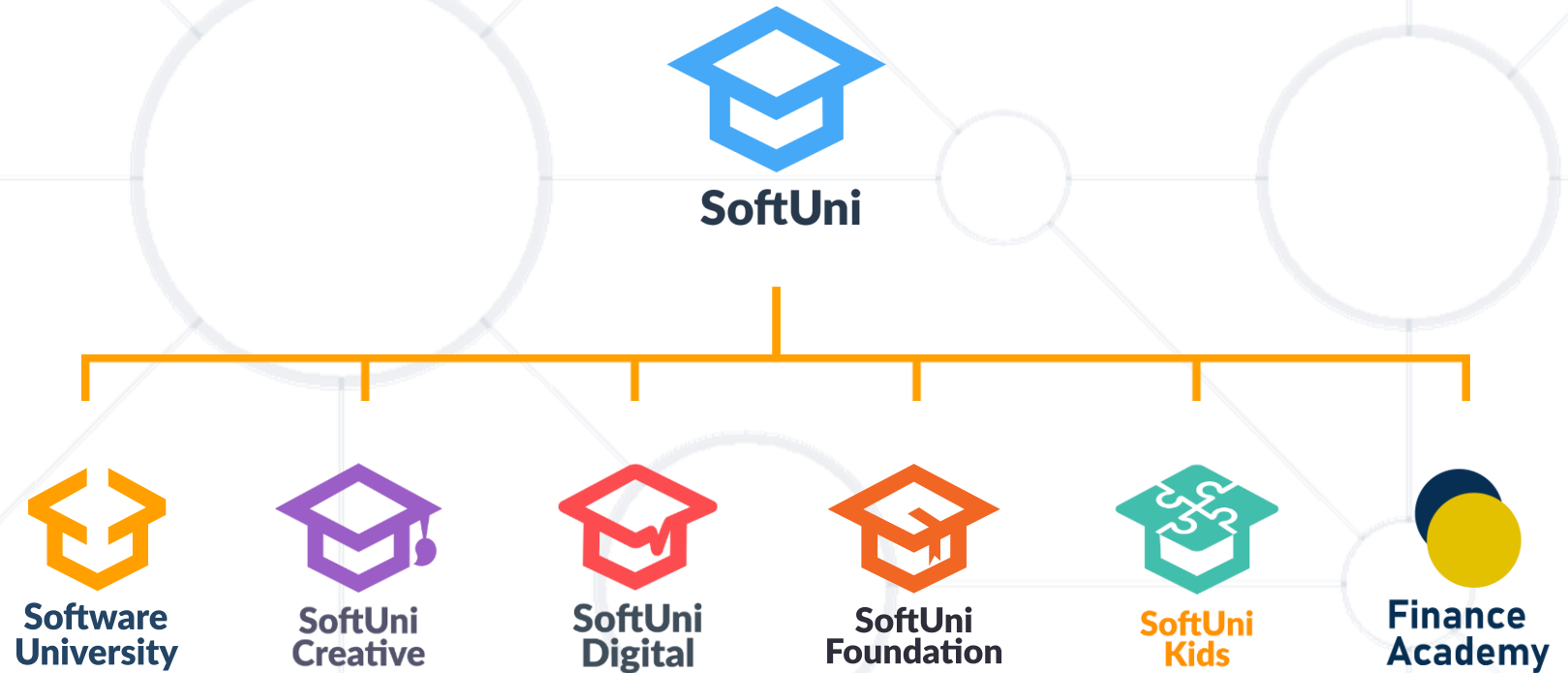


**Live Demo**

- Lexical structure in pgSQL
- Retrieving data
- Data manipulation
  - Create, update, delete
- Views



# Questions?





# SoftUni Diamond Partners

**SUPER  
HOSTING  
.BG**



**Coca-Cola HBC  
Bulgaria**

 **Flutter**<sup>TM</sup>  
International

**INDEAVR**  
Serving the high achievers



**AMBITIONED**

 **DRAFT  
KINGS**



**BOSCH**

 **Postbank**  
*Решения за твоето утре*

 **PHAR  
VISION**



**SmartIT**

**DXC**  
TECHNOLOGY

createX  


- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [softuni.org](http://softuni.org)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)

- Software University Forums

- [forum.softuni.bg](http://forum.softuni.bg)



- 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>

