# PostgreSQL

**Fanavaran Anisa**

**Iran Linux House**

Linux & Open Source Training Center

# Section 3 : DML & DQL(Part1)

# Section 2 Wrap Up – DDL Coomands

```
CREATE TYPE user_role AS ENUM ('job_seeker', 'employer');
CREATE TABLE users (
    user_id INT ,
    username VARCHAR(50) ,
    password_hash TEXT ,
    email VARCHAR(100),
    user_type user_role ,
    created_at TIMESTAMP
);
CREATE TABLE applications (
    application_id,
    job_id INT ,
    seeker_id INT,
    application_date TIMESTAMP,
    status VARCHAR(20)
);
```

# Section 2 Wrap Up – How Many Constraint? Any Improvements?

```
CREATE SEQUENCE custom_id_seq;

CREATE TABLE applications (
    application_id INT DEFAULT nextval('custom_id_seq')
PRIMARY KEY,
    job_id INT REFERENCES job_listings(job_id),
    seeker_id INT REFERENCES job_seekers(seeker_id),
    application_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) CHECK (status IN ('pending',
'accepted', 'rejected')) DEFAULT 'pending'
);

-- Another Approach
application_id INT GENERATED BY DEFAULT AS IDENTITY (START
WITH 10 INCREMENT BY 10)
```





# Section 2 Wrap Up – Which One is Better?

```sql
CREATE TABLE Orders (
    OrderDay DATE,
    OrderID INT,
    -- Other columns
    PRIMARY KEY (OrderDay, OrderID)
);

CREATE TABLE Orders (
    OrderDay DATE,
    OrderID INT,
    -- Other columns
    CONSTRAINT PK_Orders PRIMARY KEY (OrderDay, OrderID)
);
```

# Section 2 Wrap Up – Drop Both?

```
CREATE TABLE ExampleTable (
    Column1 INT CONSTRAINT DF_Column1 DEFAULT 0
NOT NULL,
    Column2 VARCHAR(255) CONSTRAINT NN_Column2
NOT NULL
);

ALTER TABLE ExampleTable
ALTER COLUMN Column1 DROP CONSTRAINT DF_Column1;
== ?
ALTER TABLE ExampleTable
ALTER COLUMN Column1 DROP DEFAULT, DROP NOT NULL;
```

# Section 2 Wrap Up – Foreign Key -> (OrderDay, OrderID) ?

```sql
CREATE TABLE Orders (
    OrderDay DATE,
    OrderID INT,
    -- Other columns
    PRIMARY KEY (OrderDay, OrderID)
);

CREATE TABLE OrderDetails (
    DetailID INT,
    OrderDay INT,
    OrderID INT,
    ProductName VARCHAR(255),
    Quantity INT,
    -- Other columns
    PRIMARY KEY (DetailID),
);
```





# Section 2 Wrap Up – Foreign Key -> (OrderDay, OrderID) ?

```
CREATE TABLE Orders (
    OrderDay DATE,
    OrderID INT,
    -- Other columns
    PRIMARY KEY (OrderDay, OrderID)
);

CREATE TABLE OrderDetails (
    DetailID INT,
    OrderDay INT,
    OrderID INT,
    ProductName VARCHAR(255),
    Quantity INT,
    -- Other columns
    UNIQUE (OrderDay, OrderID)
    PRIMARY KEY (DetailID),
    FOREIGN KEY (OrderDay, OrderID) REFERENCES Orders(OrderDay,
OrderID)
);
```



# Section 2 Wrap Up – Foreign Key -> (OrderDay, OrderID) ?

```
CREATE TABLE Orders (
    OrderDay DATE,
    OrderID INT,
    -- Other columns
    PRIMARY KEY (OrderDay, OrderID)
);

CREATE TABLE OrderDetails (
    DetailID INT,
    OrderDay INT,
    OrderID INT,
    ProductName VARCHAR(255),
    Quantity INT,
    -- Other columns
    UNIQUE (OrderDay, OrderID)
    PRIMARY KEY (DetailID),
    FOREIGN KEY (OrderDay, OrderID) REFERENCES Orders(OrderDay,
OrderID)
);
```

Any Mistake?

# Alter Table – Some Practical Samples

```
-- Column Operations
ALTER TABLE employees ADD COLUMN birthdate DATE;

ALTER TABLE employees DROP COLUMN salary;

ALTER TABLE employees ALTER COLUMN name TYPE
VARCHAR(100);

ALTER TABLE employees RENAME COLUMN name TO
full_name;
```





# Alter Table – Constraint Operations

```
-- Constraint Operations:
ALTER TABLE employees ADD CONSTRAINT salary_check
CHECK (salary > 0);

ALTER TABLE orders ADD CONSTRAINT
composite_key_constraint UNIQUE (order_date,
customer_id);

ALTER TABLE employees DROP CONSTRAINT
salary_check;

ALTER TABLE employees DISABLE TRIGGER ALL;
ALTER TABLE employees ENABLE TRIGGER ALL;
```

# Alter Table – Table Related Operations – Cluster?

```
ALTER TABLE old_table_name RENAME TO
new_table_name;
ALTER TABLE employees OWNER TO new_owner;

ALTER TABLE employees ENABLE TRIGGER
trigger_name;
ALTER TABLE employees DISABLE TRIGGER
trigger_name;


ALTER TABLE employees CLUSTER ON index_name
```

# When To Use Clustering on a Table ?

```
-- In PostgreSQL, there is no default clustering for tables. By default, tables are stored as an unordered heap, meaning that the physical storage order of rows on disk is not guaranteed to follow any particular order.
-- Create a table without specifying clustering
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    employee_name VARCHAR(100),
    salary NUMERIC
);

-- Create an index on the table
CREATE INDEX idx_employees_salary ON employees (salary);

-- Cluster the table based on the created index
CLUSTER employees USING idx_employees_salary;
-- or
ALTER TABLE employees CLUSTER ON idx_employees_salary;
```

# Alter Table – Table Related Operations. fillfactor=90?

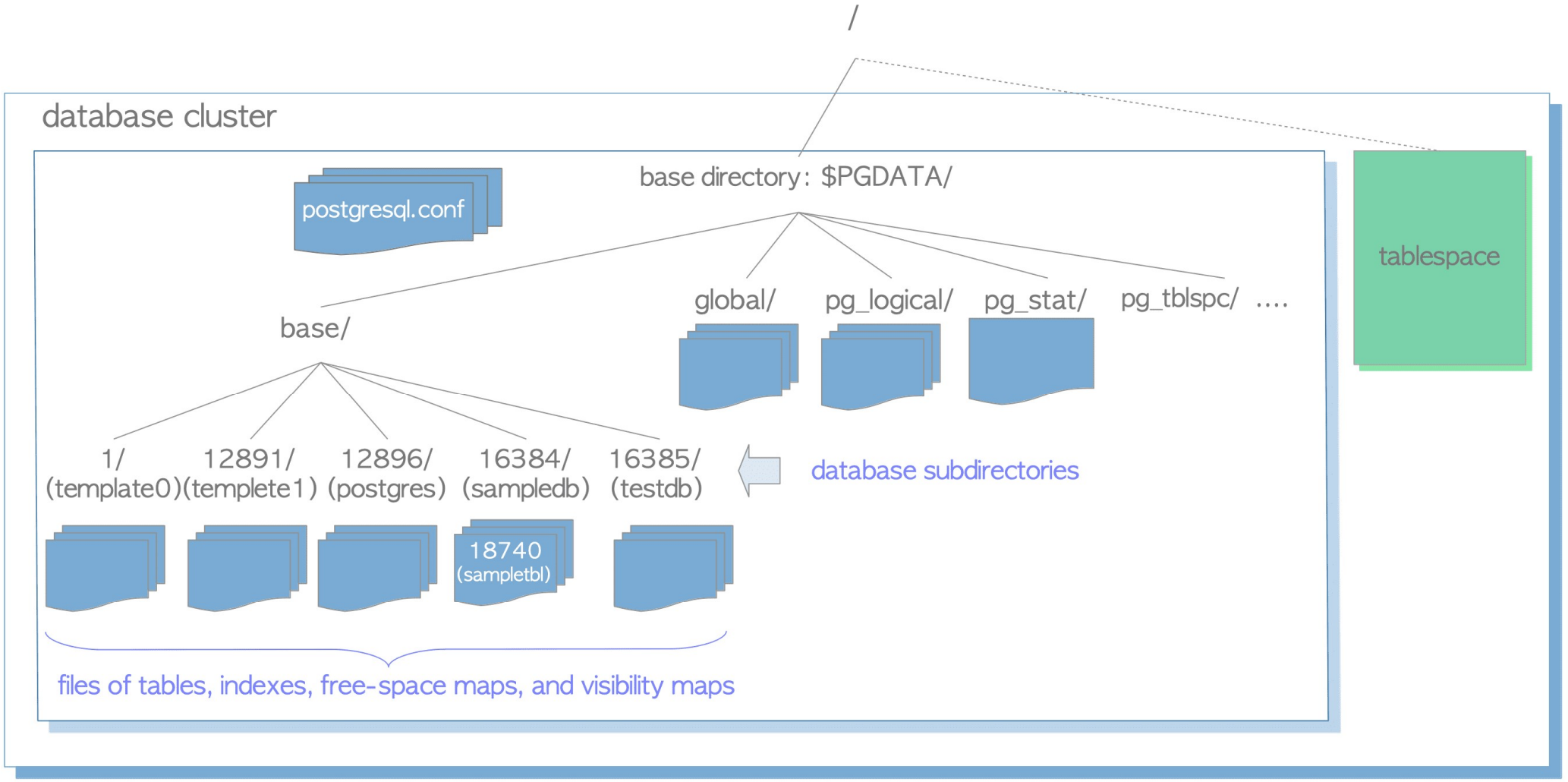
```
ALTER TABLE employees SET SCHEMA new_schema;

-- handling of Object IDs (OIDs – Rows ID) in PostgreSQL tables
-- this feature is deprecated.
ALTER TABLE employees SET WITHOUT OIDS;
ALTER TABLE employees SET WITH OIDS;

ALTER TABLE employees SET TABLESPACE new_tablespace;

-- The fill factor is a percentage that determines how full PostgreSQL should try to pack data page.
ALTER TABLE employees SET (fillfactor = 70);
-- Databases read and write in (data)pages. When you read a row from a table, the database finds the page where the row lives and identifies the file and offset where the page is located on disk
```

# A Quick Look at Tablespaces – Default Tablespace




https://www.interdb.jp/pg/pgsql01.html

# Postgres Storage Overview

**1.Tablespace:**

1. A tablespace in PostgreSQL is a location where the database stores its data.
2. A PostgreSQL database can have multiple tablespaces, each represented by a directory in the file system.

**2.Table Files within a Tablespace:**

1. Tables within a database are associated with files within tablespaces.
2. The files within tablespaces are used to store the actual data pages of multiple tables. Multiple tables can share the same file(s) within a tablespace.

**3.Table Structure:**

1. In PostgreSQL, tables are stored as heap files. Each heap file is divided into fixed-size pages.
2. The actual data rows of multiple tables may be stored in the same pages within the shared file(s).

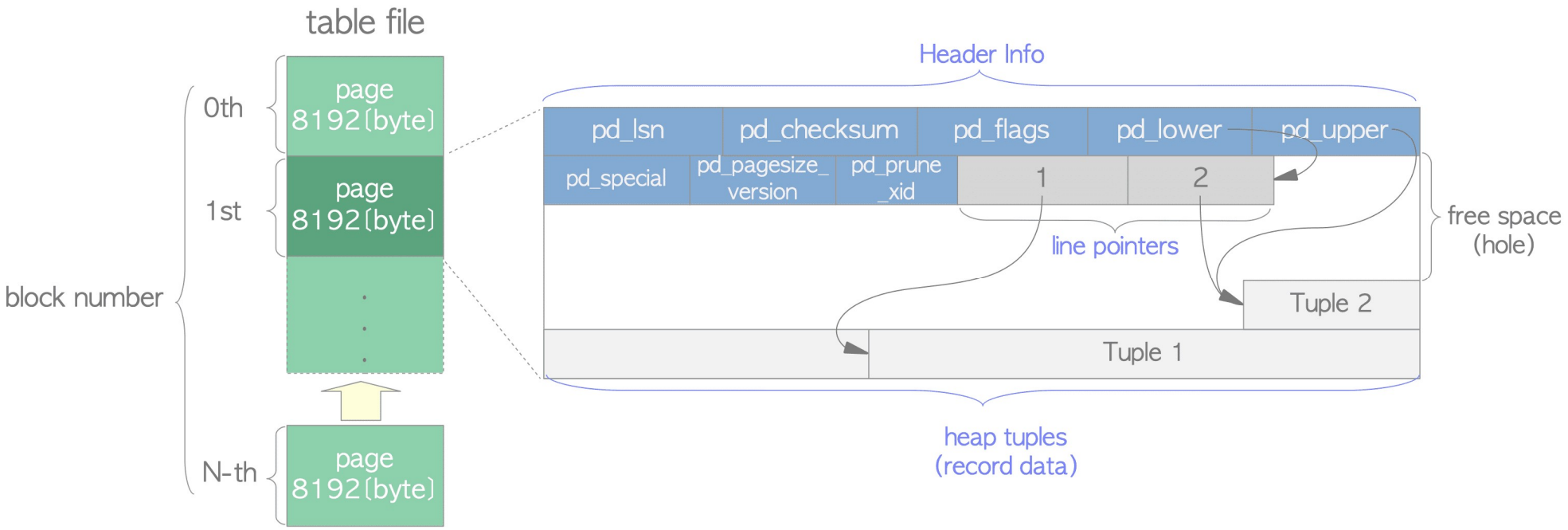
**4.Indexes:**

1. If indexes are created on the tables, each index is associated with its own set of files within the tablespaces, similar to the data files.

**5.Configuration:**

1. The configuration of table storage, including the choice of tablespaces and file locations, can be influenced by database administrators during table creation or modification.

# A Quick Look at Page Layout




https://www.interdb.jp/pg/pgsql01.html

# Section 3 Overview

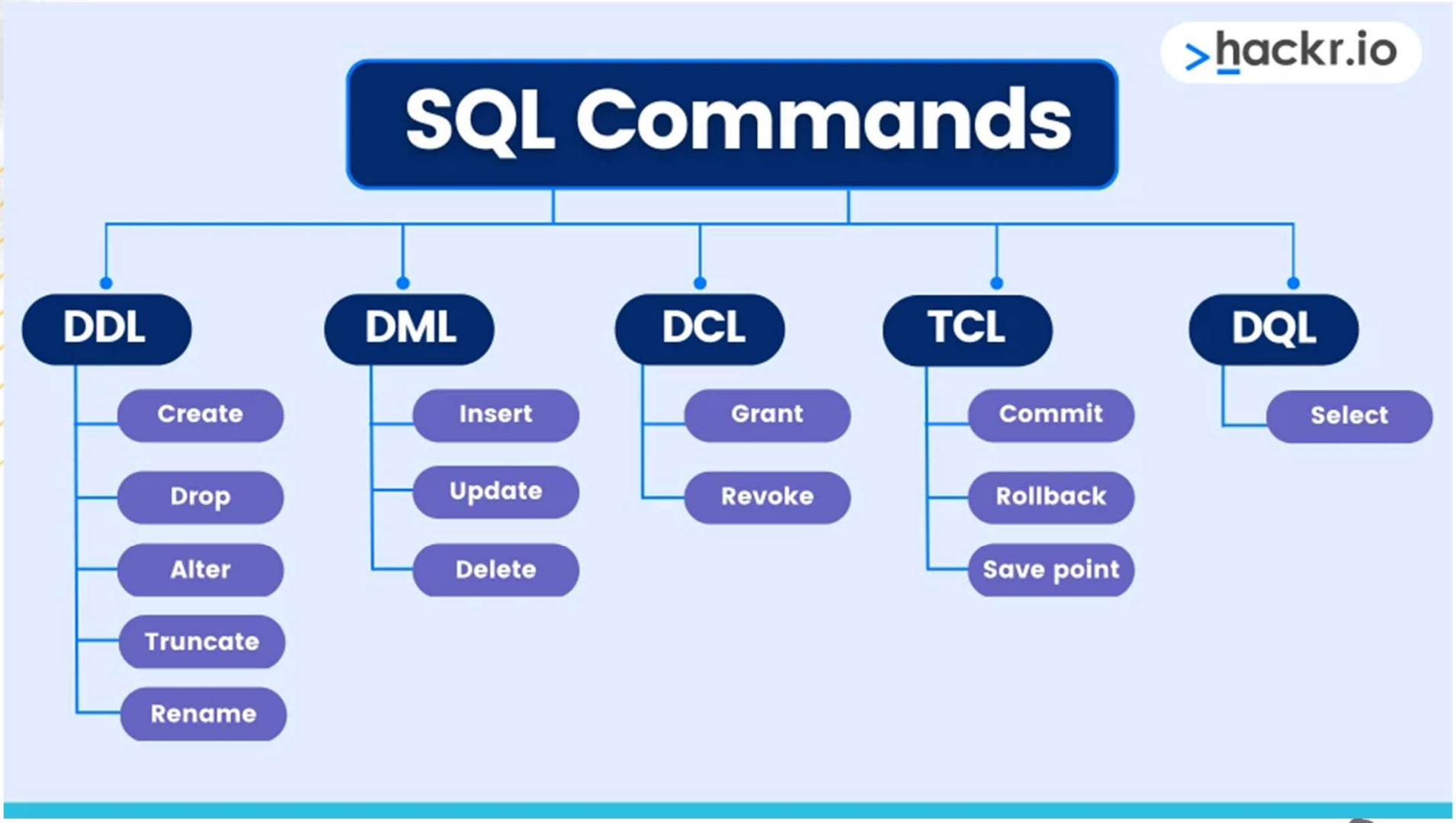
## Section 1: Basics of DML

- Insert
- Update
- Delete
- Transaction Intro

## Section 2: DQL Introduction

- Select Structure
- Projection(field selection) in Details

# SQL Commands Overview



# Engage and Extend – Today Insights

- **Which one resets the Identity column ?**
  - **Truncate**
  - **Delete**

```
CREATE TEMP TABLE YourTable (
    ID INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    Name VARCHAR(50)
);
-- Insert two records
INSERT INTO YourTable (Name) VALUES ('Record1'), ('Record2');

-- Delete the records
DELETE FROM YourTable;
INSERT INTO YourTable (Name) VALUES ('Record3');
SELECT * FROM YourTable;

-- Truncate the table
TRUNCATE YourTable;
INSERT INTO YourTable (Name) VALUES ('Record4');
SELECT * FROM YourTable;
```





# Engage and Extend – Today Insights

**- Which One is Wrong?**

```
CREATE temp TABLE YourTable (
    ID INT Generated By Default as IDENTITY PRIMARY KEY,
    Name VARCHAR(50)
);

INSERT INTO YourTable
DEFAULT VALUES;

INSERT INTO YourTable (name)
VALUES ();

INSERT INTO YourTable (name)
VALUES (NULL);
```





# Engage and Extend – Today Insights

**- Which One is Wrong?**

```
CREATE temp TABLE YourTable (
    ID INT Generated By Default as IDENTITY PRIMARY KEY,
    Name VARCHAR(50)
);

INSERT INTO YourTable
DEFAULT VALUES;

INSERT INTO YourTable (name)
VALUES ();

INSERT INTO YourTable (name)
VALUES (NULL);
```





# Engage and Extend – Today Insights

## - Output? Error? Errors?

```
 -- Create a temporary table
CREATE TEMP TABLE YourTable (
    ID INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    Name VARCHAR(50)
);

-- Insert 3 records
INSERT INTO YourTable (Name) VALUES ('Record1'),
(5,'Record2'), ('Record3');

SELECT * FROM YourTable;
```

# Engage and Extend – Today Insights

- Upsert in Postgres?

- If we assign a value to an IDENTITY, from where it continues? The new number or the last generated id?

- Dynamic Updates in One Command
- ….





# DML in Postgres

# Insert Command

```
-- Syntax
INSERT INTO table_name(column1, column2, column3, ...)
VALUES
(value11, value12, value13, ...)
(value21, value22, value23, ...)
... ;

-- insert a row in the Customers table
INSERT INTO Customers(customer_id, first_name,
last_name, age, country)
VALUES
(7, 'Ron', 'Weasley', 31, 'UK');
```





# Insert Command

Table: Customers

| customer_id | first_name | last_name | age | country |
|---|---|---|---|---|
| 1 | John | Doe | 31 | USA |
| 2 | Robert | Luna | 22 | USA |
| 3 | David | Robinson | 22 | UK |
| 4 | John | Reinhardt | 25 | UK |

```
INSERT INTO Customers(customer_id, first_name,
last_name, age, country)
VALUES (5, 'Harry', 'Potter', 31, 'USA');
```

| customer_id | first_name | last_name | age | country |
|---|---|---|---|---|
| 1 | John | Doe | 31 | USA |
| 2 | Robert | Luna | 22 | USA |
| 3 | David | Robinson | 22 | UK |
| 4 | John | Reinhardt | 25 | UK |
| 5 | Harry | Potter | 31 | USA |

# Insert Command

- **Order of columns ? Yes and No?**
- **What Happens when No Column Names and Some Field with Default Values?**
- **Number of Columns?**

```
-- insert a row in the Customers table
INSERT INTO Customers(customer_id, first_name,
last_name, age, country)
VALUES
(7, 'Ron', 'Weasley', 31, 'UK'),
(9, 'John', 'Snow', 39, 'US');
```

# Insert Command – Identity Column

```
-- Create a temporary table
CREATE TEMP TABLE YourTable (
    ID INT GENERATED BY DEFAULT AS IDENTITY
PRIMARY KEY,
    Name VARCHAR(50)
);

--  Error?
INSERT INTO YourTable(id, Name) values
(Null,'Record1');

-- Error??
INSERT INTO YourTable values ('Record1');
```

# All Null Columns Except : Not Null With Default Value

```sql
-- Create a table with a NOT NULL column and a default value
create Temp TABLE ExampleTable (
    ID SERIAL PRIMARY KEY,
    Name VARCHAR(50) NOT NULL DEFAULT 'Unknown'
);

-- Insert a row without specifying the NOT NULL column (Name)
INSERT INTO ExampleTable (ID) VALUES (1);

-- View the contents of the table
SELECT * FROM ExampleTable;
```





# Insert Command – Empty String vs NULL

```
-- Create a table with a NOT NULL column and a default
value
create Temp TABLE ExampleTable (
    ID SERIAL PRIMARY KEY,
    Name VARCHAR(50) NOT NULL DEFAULT 'Unknown'
);

-- Insert a row without specifying the NOT NULL column
(Name)
INSERT INTO ExampleTable(ID) VALUES (10);
INSERT INTO ExampleTable(Name) VALUES ('');
-- View the contents of the table
SELECT * FROM ExampleTable;
```





# Insert Into Command – Does it creates the destination Table?

```
INSERT INTO destination_table (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM source_table [join , ....]
[Where ... ];


-- copy data to an existing table
INSERT INTO OldCustomers
SELECT *
FROM Customers;


-- copy rows that satisfy the condition
INSERT INTO OldCustomers
SELECT *
FROM Customers
WHERE country = 'USA';
```

# Engage & Extend - Insert  Command Failure

## *2 Rows or No Rows Will Inserted?*

```
INSERT INTO employee (employee_id, first_name, last_name,
age, salary)
VALUES
    (1, 'John', 'Doe', 30, 50000),
    (2,  'Alice', 'Smith', 25, 60000),
    (1,  'Ali', 'Smith', 29, 80000);
```

## *With/Without:*

```
 ON CONFLICT (employee_id)
 DO NOTHING;
```





# Select Into Command / Create Table AS

```
SELECT
    select_list
INTO [ TEMPORARY | TEMP | UNLOGGED ] [ TABLE ] new_table_name
FROM
    table_name
WHERE
    search_condition;


CREATE TABLE new_table_name [( column_name_list)]
AS query;

CREATE TABLE new_table AS TABLE existing_table WITH NO DATA;


create TABLE new_table (LIKE existing_table INCLUDING ALL);

-- Try it Out: Which One Preserve the Structure(Keys/Constraints/Indexes)
```

# Create Table Like ...

- **INCLUDING DEFAULTS**: Copies default values associated with columns.
- **INCLUDING CONSTRAINTS**: Copies constraints, both column-level and table-level.
- **INCLUDING INDEXES**: Copies indexes defined on the source table.
- **INCLUDING STORAGE**: Copies storage parameters

```
CREATE TABLE new_employees (LIKE
old_employees INCLUDING DEFAULTS
INCLUDING CONSTRAINTS INCLUDING INDEXES);
```





# Update Command

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

```
-- update a single value in the given row
UPDATE Customers
SET age = 21
WHERE customer_id = 1;
```

# Update Command

Table: Customers

| customer_id | first_name | last_name | age | country |
|---|---|---|---|---|
| 1 | John | Doe | 31 | USA |
| 2 | Robert | Luna | 22 | USA |
| 3 | David | Robinson | 22 | UK |
| 4 | John | Reinhardt | 25 | UK |
| 5 | Betty | Doe | 28 | UAE |

```
UPDATE Customers
SET first_name = 'Johnny'
WHERE customer_id = 1;
```

| customer_id | first_name | last_name | age | country |
|---|---|---|---|---|
| 1 | Johnny | Doe | 31 | USA |
| 2 | Robert | Luna | 22 | USA |
| 3 | David | Robinson | 22 | UK |
| 4 | John | Reinhardt | 25 | UK |
| 5 | Harry | Potter | 31 | USA |

# Update  Multiple Rows / Resolve Conflict

```
-- update multiple rows satisfying the condition
UPDATE Customers
SET country = 'NP'
WHERE age = 22;

-- update all rows
UPDATE Customers
SET country = 'NP';


UPDATE employees
SET salary = salary * 1.1
WHERE age > 30
ON CONFLICT (employee_id)
DO UPDATE SET salary = COALESCE(EXCLUDED.salary,
employees.salary);
```





# Engage & Extend – What Conflict?

```
UPDATE employees
SET salary = salary * 1.1
WHERE age > 30
ON CONFLICT (employee_id)
DO UPDATE SET salary =
COALESCE(EXCLUDED.salary, employees.salary);
```

# Delete/Truncate Command

```
DELETE FROM table_name
[WHERE condition];

DELETE FROM Customers
WHERE customer_id = 5;

-- Delete All Records/Rows
DELETE FROM Customers;

TRUNCATE TABLE Customers;
```

SQL DELETE vs. TRUNCATE

# Delete/Truncate Command

Table: Customers

| customer_id | first_name | last_name | age | country |
|---|---|---|---|---|
| 1 | John | Doe | 31 | USA |
| 2 | Robert | Luna | 22 | USA |
| 3 | David | Robinson | 22 | UK |
| 4 | John | Reinhardt | 25 | UK |
| 5 | Betty | Doe | 28 | UAE |

```
DELETE FROM Customers
WHERE customer_id = 5;
```

| customer_id | first_name | last_name | age | country |
|---|---|---|---|---|
| 1 | John | Doe | 31 | USA |
| 2 | Robert | Luna | 22 | USA |
| 3 | David | Robinson | 22 | UK |
| 4 | John | Reinhardt | 25 | UK |

# SQL DELETE vs. TRUNCATE

| SQL DELELTE | SQL TRUNCATE |
| --- | --- |
| SQL DELETE supports the WHERE clause. | SQL TRUNCATE doesn't support the WHERE clause. |
| SQL DELETE can remove single, multiple, or all rows/records from a table. | SQL TRUNCATE can only remove all the records from a table. |
| The operation is logged, and you can use the ROLLBACK command (if in a transaction) to undo the delete operation before committing | Truncate is not logged as extensively as DELETE, and the operation cannot be rolled back. |
| It may cause fragmentation in the table and indexes, as empty spaces are not automatically reclaimed. | Truncating a table is usually faster than deleting all records, especially for large tables.<br>It releases allocated space, reducing fragmentation |

# DML : A Closer Look

# Returning Clause

```
INSERT INTO employees (first_name, last_name,
birthday, salary)
VALUES ('Chris', 'Miller', '2003-01-09'::date,
70000)
RETURNING employee_id;


---------------------------------

RETURNING *;
RETURNING employee_id,first_name, last_name, age
```





# Try it Out

3 IDs are returned or Only the last one?

```
INSERT INTO user (first_name, last_name)
VALUES
  ( 'John', 'Doe', other_values1),
  ( 'Jane', 'Smith', other_values2),
  ( 'Alice', 'Johnson', other_values3)
RETURNING id;
```

# INSERT with Conflict Resolution: Upsert

```
-- Assume there is a unique constraint on the "product_code"
column
INSERT INTO products (product_code, name, price)
VALUES ('P123', 'Product ABC', 29.99)
Returning *
ON CONFLICT (product_code) DO UPDATE SET price =
EXCLUDED.price;
```

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...)
ON CONFLICT (conflict_column)
DO UPDATE SET column1 = value1, column2 = value2, ...
   | NOTHING;
```





# Try It Out – Multiple Fields? Multiple Conflicts? Multiple With Same Target?

```
-- Assuming unique constraints on (username, email) and
(product_code)
INSERT INTO your_table (username, email, product_code,
other_columns)
VALUES ('john_doe', 'john@example.com', 'ABC123', other_values)
ON CONFLICT (username, email) DO UPDATE SET other_column1 =
new_value1
ON CONFLICT (product_code) DO UPDATE SET other_column2 =
new_value2;
```

# Engage & Extend – What does this code snippet do?

```sql
-- Assuming a check constraint on min_salary
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    salary INT CHECK (salary >= 7500000),
    other_columns TEXT
);

-- Test data with potential conflicts
INSERT INTO employees (name, salary, other_columns)
VALUES
    ('John Doe', 8000000, 'other_values'),
    ('Jane Smith', 7000000, 'other_values'),
    ('Alice Johnson', 9000000, 'other_values')
ON CONFLICT (salary) DO UPDATE SET salary = CASE
    WHEN EXCLUDED.salary < 7500000 THEN 7500000
    ELSE EXCLUDED.salary
END
RETURNING *;
```

# Try It Out – Is It Valid?

```sql
-- Assuming a check constraint on min_salary
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    salary INT CHECK (salary >= 7500000),
    other_columns TEXT
);

-- Test data with potential conflicts
INSERT INTO employees (name, salary, other_columns)
VALUES
    ('John Doe', 8000000, 'other_values'),
    ('Jane Smith', 7000000, 'other_values'),
    ('Alice Johnson', 9000000, 'other_values')
ON CONFLICT (salary) DO UPDATE SET salary = CASE
    WHEN EXCLUDED.salary < 7500000 THEN 7500000
    ELSE EXCLUDED.salary
END
RETURNING *;
```





# Try It Out – Is It Valid?

```sql
-- Assuming a check constraint on min_salary
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    salary INT CHECK (salary >= 7500000),
    other_columns TEXT
);

-- Test data with potential conflicts
INSERT INTO employees (name, salary, other_columns)
VALUES
    ('John Doe', 8000000, 'other_values'),
    ('Jane Smith', 7000000, 'other_values'),
    ('Alice Johnson', 9000000, 'other_values')
ON CONFLICT (salary) DO UPDATE SET salary = CASE
    WHEN EXCLUDED.salary < 7500000 THEN 7500000
    ELSE EXCLUDED.salary
END
RETURNING *;
```





# Delete with Returning

```
DELETE FROM employees
WHERE age > 30
RETURNING employee_id, first_name,
last_name;
```

# More Advanced Updates

```sql
UPDATE employees
SET is_manager = true
WHERE EXISTS (
    SELECT 1
    FROM managers
    WHERE managers.employee_id = employees.employee_id
);
```

```sql
UPDATE employees
SET salary = salary * 1.1
WHERE age > 30
LIMIT 10;
```

Make Sense?





# More Advanced Updates – Dynamic Update

```
UPDATE employees
SET salary = salary * 1.1
WHERE age > 30
RETURNING employee_id, salary;


UPDATE employees
SET salary =
    CASE
        WHEN age < 25 THEN salary * 1.2
        WHEN age >= 25 AND age < 35 THEN salary * 1.1
        ELSE salary
    END;
```





# More Advanced Deletes

```
DELETE FROM employees
WHERE age > 30
RETURNING employee_id, first_name, last_name;

DELETE FROM employees
WHERE age > 30
LIMIT 5;
```

# More Advanced Deletes

```
DELETE FROM employees
WHERE age > 30
RETURNING employee_id, first_name, last_name;

DELETE FROM employees
WHERE age > 30
LIMIT 5;
```

# More Advanced Deletes – Dependency on Other Tables

```
DELETE FROM employees
WHERE employee_id IN (SELECT employee_id FROM
old_employees);
```

```
DELETE FROM employees
USING old_employees
WHERE employees.employee_id = old_employees.employee_id;
```

# More Advanced Deletes – Using Keyword

```
DELETE FROM target_table
USING additional_table1
JOIN additional_table2 ON additional_table1.column_name =
additional_table2.column_name
WHERE target_table.column_name = additional_table1.column_name
  AND target_table.column_name = additional_table2.column_name;
```





# DQL : First Step

# Select Command – A Sample Table

Table: Customers

| customer_id | first_name | last_name | age | country |
|---|---|---|---|---|
| 1 | John | Doe | 31 | USA |
| 2 | Robert | Luna | 22 | USA |
| 3 | David | Robinson | 22 | UK |
| 4 | John | Reinhardt | 25 | UK |

# Select Command – Basic Structure