PostgreSQL

Advanced Postgres

Surendra Panpaliya

Schema Design & Data Types

Surendra Panpaliya

Agenda

- Tables, schemas, sequences
- PostgreSQL vs Oracle data types
- SERIAL vs GENERATED ALWAYS AS IDENTITY

Hands-On

Create customer-product schema

using SERIAL and constraints

Assignment

Convert an Oracle schema DDL to PostgreSQL DDL

Tables, schemas, sequences

Surendra Panpaliya

Understanding the Mapping

Oracle Concept	PostgreSQL Equivalent
Schema	Schema
Table	Table
Sequence	Sequence (Auto Number)
Dual Table / FROM DUAL	Use SELECT without FROM or use SELECT 1;

Example: Create Schemas

Oracle Style:

CREATE USER telecom IDENTIFIED BY password;

Oracle creates a **user-schema** by default.

Example: Create Schemas

PostgreSQL Equivalent:

CREATE SCHEMA telecom;

Note: In PostgreSQL, CREATE SCHEMA

just creates a namespace (not a user).

PostgreSQL (Preferred way using SERIAL):

```
CREATE TABLE telecom.customers (
    customer_id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    mobile_number VARCHAR(15),
    city VARCHAR(50),
    balance NUMERIC(10,2)
);
```

Oracle Equivalent:

```
CREATE TABLE telecom.customers (
    customer_id NUMBER GENERATED BY DEFAULT AS IDENTITY
PRIMARY KEY,
    name VARCHAR2(100),
    mobile_number VARCHAR2(15),
    city VARCHAR2(50),
    balance NUMBER(10,2)
);
```

Insert Data

```
INSERT INTO telecom.customers (name, mobile_number, city, balance)
VALUES
('John Doe', '9876543210', 'Pune', 500.00),
('Jane Smith', '8765432109', 'Mumbai', 1200.00);
```

Create Plans Table

```
CREATE TABLE telecom.plans (
    plan_id SERIAL PRIMARY KEY,
    plan_name VARCHAR(50),
    monthly_fee NUMERIC(10,2)
);
```

Create Subscriptions Table (Foreign Keys)

```
CREATE TABLE telecom.subscriptions (
    subscription_id SERIAL PRIMARY KEY,
    customer_id INT REFERENCES telecom.customers(customer_id),
    plan_id INT REFERENCES telecom.plans(plan_id),
    start_date DATE DEFAULT CURRENT_DATE
);
```

Insert Data into Plans

```
INSERT INTO telecom.plans (plan_name, monthly_fee)
VALUES ('Basic', 199.00), ('Premium', 499.00);
```

Query Data with Join

```
SELECT c.name, p.plan_name, s.start_date
FROM telecom.customers c
JOIN telecom.subscriptions s ON c.customer_id = s.customer_id
JOIN telecom.plans p ON s.plan_id = p.plan_id;
```

View Sequences

List sequences in telecom schema:

```
SELECT sequence_name FROM information_schema.sequences
WHERE sequence_schema = 'telecom’;
```

Check next value of a sequence:

```
SELECT nextval('telecom.customer_seq');
```

Drop Objects

```
DROP TABLE telecom.subscriptions;
DROP TABLE telecom.customers;
DROP TABLE telecom.plans;
DROP SEQUENCE telecom.customer_seq;
DROP SCHEMA telecom CASCADE;
```

Summary

Concept	Oracle	PostgreSQL
Schema	USER/SCHEMA	CREATE SCHEMA
Auto-Increment	IDENTITY	SERIAL or SEQUENCE
Dual Table	FROM DUAL	SELECT 1;
Sequences	CREATE SEQUENCE	Same syntax

PostgreSQL vs Oracle data types

Surendra Panpaliya

Core Data Type Comparison

Oracle	PostgreSQL	Notes
NUMBER(p,s)	NUMERIC(p,s) / DECIMAL(p,s)	Exact precision, stores as text internally.
NUMBER (no precision)	NUMERIC / DOUBLE PRECISION	NUMERIC for exact; DOUBLE PRECISION for approximate.

Core Data Type Comparison

Oracle	PostgreSQL	Notes
VARCHAR2(n)	**VARCHAR(n)**	Functionally identical. PostgreSQL does not distinguish VARCHAR and VARCHAR2.
CHAR(n)	**CHAR(n)**	Fixed-length, padded with spaces.

Core Data Type Comparison

Oracle	PostgreSQL	Notes
CLOB	**TEXT**	No size limit in PostgreSQL.
BLOB / RAW	**BYTEA**	Binary data.

Core Data Type Comparison

Oracle	PostgreSQL	Notes
DATE	**DATE**	Both store **YYYY-MM-DD**. PostgreSQL DATE has no time part.
TIMESTAMP	**TIMESTAMP WITHOUT TIME ZONE**	Same format, no timezone.

Core Data Type Comparison

Oracle	PostgreSQL	Notes
TIMESTAMP WITH TIME ZONE	**TIMESTAMP WITH TIME ZONE**	PostgreSQL handles time zones more strictly.
INTERVAL	**INTERVAL**	Supports complex date intervals (years, months, days, etc.).

Core Data Type Comparison

Oracle	PostgreSQL	Notes
BOOLEAN (Oracle 12c+)	**BOOLEAN**	PostgreSQL has native BOOLEAN. Oracle uses workarounds in older versions.
ROWID	No direct equivalent	Use ctid for internal row address (not for application use).
BFILE	No direct equivalent	Use BYTEA or store file paths.

Auto-Incrementing IDs

Oracle	PostgreSQL
NUMBER GENERATED BY DEFAULT AS IDENTITY	SERIAL / BIGSERIAL / GENERATED AS IDENTITY

Special Types

Oracle	PostgreSQL	Use Case
XMLTYPE	XML	XML storage
JSON / JSONB	JSON / JSONB	PostgreSQL natively supports JSON with indexing
USER-DEFINED TYPES (UDT)	COMPOSITE TYPES / DOMAIN	Custom data types

Function & String Handling

Oracle	PostgreSQL	Notes
SYSDATE	CURRENT_DATE / NOW()	NOW() returns timestamp.
SYSTIMESTAMP	CURRENT_TIMESTAMP	
TO_DATE()	TO_DATE()	Same function name but syntax may vary.
NVL()	COALESCE()	Same functionality.
DECODE()	CASE	PostgreSQL does not have DECODE().

Data Types Summary Cheat Sheet

Category	Oracle	PostgreSQL
Integer	NUMBER(10)	INTEGER / BIGINT
Float	BINARY_FLOAT / BINARY_DOUBLE	FLOAT / REAL / DOUBLE PRECISION
Character	VARCHAR2 / CHAR	VARCHAR / CHAR
Large Text	CLOB	TEXT
Binary	BLOB / RAW	BYTEA
Date/Time	DATE / TIMESTAMP	DATE / TIMESTAMP
Boolean	Workaround (CHAR(1))	BOOLEAN
Sequence	SEQUENCE	SEQUENCE
JSON	Oracle 21c JSON	JSON / JSONB

Key Differences to Remember

Oracle	PostgreSQL
VARCHAR2, NVARCHAR2 distinction	Only VARCHAR
NUMBER with arbitrary precision	Use NUMERIC for precision
BLOB/CLOB separate	BYTEA / TEXT handles large data
DATE includes time	DATE is date only in PostgreSQL
JSON less flexible (Oracle pre-21c)	Native JSON support with JSONB indexing

Customer Billing Table Example

Oracle:

```
CREATE TABLE CUSTOMER_BILLING (
    BILL_ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY
KEY,
    CUSTOMER_ID NUMBER,
    BILL_AMOUNT NUMBER(10,2),
    BILL_DATE DATE
);
```

Customer Billing Table Example

PostgreSQL:

```
CREATE TABLE customer_billing (
    bill_id SERIAL PRIMARY KEY,
    customer_id INTEGER,
    bill_amount NUMERIC(10,2),
    bill_date DATE
);
```

SERIAL vs GENERATED ALWAYS AS IDENTITY

Surendra Panpaliya

What is SERIAL?

SERIAL is a

PostgreSQL-specific shorthand

to auto-increment a column.

How it works?

Creates an **INTEGER column**

Automatically creates an **associated SEQUENCE**

Uses nextval() to auto-generate values

Example

```
CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
    name VARCHAR(100)
);
```

Example

Behind the scenes:

PostgreSQL will do this:

CREATE SEQUENCE employees_emp_id_seq;

Example

```
CREATE TABLE employees (
 emp_id INTEGER NOT NULL DEFAULT
nextval('employees_emp_id_seq'),
 name VARCHAR(100)
);
```

What is GENERATED ALWAYS AS IDENTITY?

GENERATED AS IDENTITY is

ANSI SQL standard syntax

Introduced in PostgreSQL 10

Syntax:

```
CREATE TABLE employees (
    emp_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name VARCHAR(100)
);
```

Key Differences

Feature	SERIAL	GENERATED AS IDENTITY
Standard Compliance	PostgreSQL-specific	ANSI SQL Standard
Sequence Control	Creates an implicit sequence (manual management needed)	Fully managed internally
Customization	Limited (requires ALTER SEQUENCE)	Supports START WITH, INCREMENT BY, MINVALUE, CYCLE

Key Differences

Feature	SERIAL	GENERATED AS IDENTITY
DROP TABLE Behavior	Leaves the sequence unless CASCADE is used	Drops the sequence automatically
Insert Override	Can insert custom values by default	**ALWAYS**: Cannot insert custom values unless OVERRIDING SYSTEM VALUE is used **BY DEFAULT**: Allows manual inserts

ALWAYS vs BY DEFAULT

Option	Behavior
GENERATED ALWAYS AS IDENTITY	Auto-generates values and rejects manual insert values (unless overridden)
GENERATED BY DEFAULT AS IDENTITY	Auto-generates values but allows manual inserts if specified

Example Comparison

SERIAL

```
CREATE TABLE test_serial (
    id SERIAL PRIMARY KEY,
    name TEXT
);
```

Insert custom value:

```
INSERT INTO test_serial (id, name) VALUES (100, 'Custom ID'); -- ✅ Allowed
```

IDENTITY

```
CREATE TABLE test_identity (
   id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
   name TEXT
);
```

Insert custom value:

```
INSERT INTO test_identity (id, name) VALUES (100, 'Custom ID');
```

-- ❌ ERROR: cannot insert into identity column

Override behavior

```
INSERT INTO test_identity (id, name)

OVERRIDING SYSTEM VALUE
VALUES (100, 'Custom ID'); -- ✅ Allowed
```

Summary Table

Feature	SERIAL	IDENTITY
SQL Standard	❌	✅
Sequence Name Control	Manual	Automatic
DROP Behavior	Leaves sequence	Drops sequence
Custom Inserts	Allowed	Controlled (ALWAYS/BY DEFAULT)
Preferred For New Systems	🚫 Deprecated in future	✅ Recommended

Which One Should You Use?

Use Case	Recommendation
New Projects	GENERATED AS IDENTITY (future-proof & standard)
Legacy Support	SERIAL (if maintaining older code)

Real-World Amdocs Scenario

Oracle	PostgreSQL
NUMBER GENERATED AS IDENTITY	GENERATED ALWAYS AS IDENTITY
SEQUENCE + TRIGGER (older Oracle)	SERIAL or Manual SEQUENCE

Final Recommendation

Use GENERATED AS IDENTITY

for all new PostgreSQL development

Especially for Oracle to PostgreSQL migrations,

to match modern standards.

Practice Lab on SERIAL vs IDENTITY in PostgreSQL

Surendra Panpaliya

Lab Objective

Understand the **differences**

between SERIAL and IDENTITY columns

through practical examples.

Lab Objective

Concept	Practice
SERIAL usage	✅
GENERATED AS IDENTITY usage	✅
Sequence management	✅
Insert overrides	✅
Drop behavior	✅

Agenda

Basic Concepts and SQL Commands

Database, schema, tables, rows, and columns

Data types and constraints

Primary keys and foreign keys

Agenda

Basic SQL Commands

CREATE, INSERT, SELECT, UPDATE, DELETE

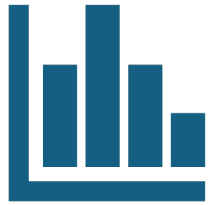
Filtering and sorting data

Aggregate function

Database

A PostgreSQL database is

Collection of schemas



Store data

Database

A company database

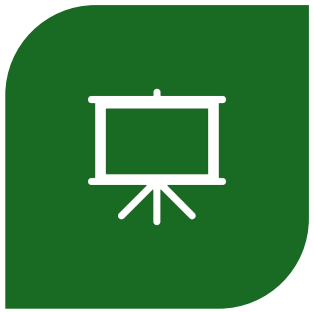
store data related to employees,

departments, products

CREATE DATABASE company_db;

Schema

LOGICAL CONTAINER INSIDE A DATABASE.



HOLDS TABLES, VIEWS,

INDEXES, SEQUENCES,

FUNCTIONS

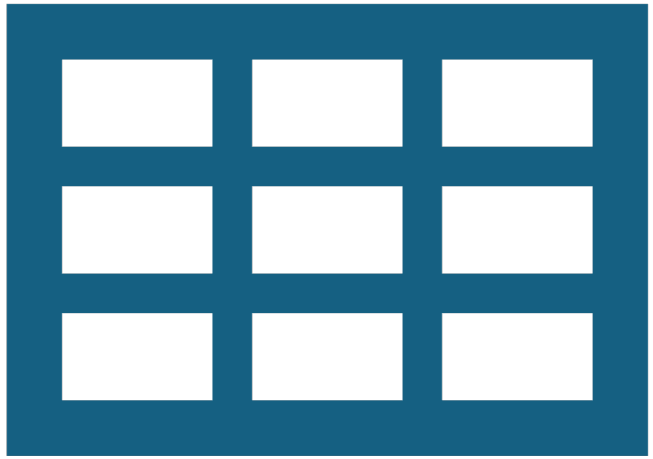
Example

public (default)

hr (human resources)

sales

CREATE SCHEMA hr;



Table

- Collection of
- Related data entries
- Consists of
- Rows
- Columns

Table Example

An employees table

store details like

employee ID, name,

position, salary

Creating a Table

```
CREATE TABLE hr.employees (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    hire_date DATE,
    salary NUMERIC(8, 2)
);
```

Creating a Table

employee_id:
An integer column that
automatically increments with each new row,
serving as the primary key.

Creating a Table

first_name & last_name:
A string column with
a maximum length of
50 characters.

Creating a Table

email:
A string column with a
maximum length of 100 characters,
unique for each row.

Creating a Table

hire_date:

A date column.

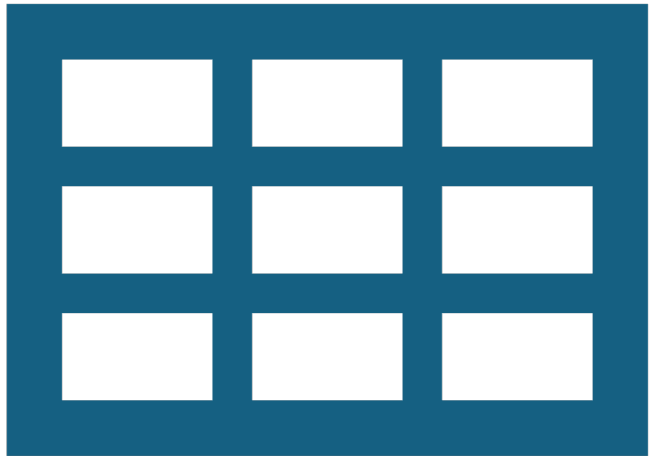
Creating a Table

salary:

A numeric column

allows up to 8 digits,

including 2 decimal places



Row

A row (or record)

Represents

Single,

implicitly structured

data item in a table.

Row Examples

A row in

Employees table

Contain data

about a Single employee.

Inserting Row into a Table

INSERT INTO hr.employees (first_name, last_name, email, hire_date, salary)

VALUES (‘Dev’, ‘Rathi', ‘dev.rathi@example.com', ‘2024-07-18', 50000.00);

Column

A column (or field)

a set of data values

of a particular type,

one for each row

of the table

Column Example

The salary column

in the employee table

stores the salary of

each employee.

Column

```
SELECT employee_id, first_name, last_name, email, hire_date, salary
FROM hr.employees;
```

Column

```
ALTER TABLE hr.employees
ADD COLUMN phone_number VARCHAR(20);
```

Selecting Data from a Table

```
SELECT employee_id, first_name, last_name, email, hire_date, salary FROM hr.employees;
```

Updating Data in a Table

```
UPDATE hr.employees
SET salary = 55000.00
WHERE employee_id = 1;
```

Deleting Data from a Table

```
DELETE FROM hr.employees
WHERE employee_id = 1;
```

Adding a Column to a Table

```
ALTER TABLE hr.employees
ADD COLUMN phone_number VARCHAR(20);
```

Dropping a Column from a Table

```
ALTER TABLE hr.employees
DROP COLUMN phone_number;
```

Dropping a Table

```
DROP TABLE hr.employees;
```

Dropping a Schema

```
DROP SCHEMA hr CASCADE;
```

Dropping a Database

DROP DATABASE

Data Types

Define the kind of data that

Can be stored in each column

Constraints enforce rules on the data.

Data Types

SERIAL:

Auto-incrementing integer,

often used for primary keys.

VARCHAR(n):

Variable-length character string.

Data Types

TEXT:

Variable-length character string

with no specific maximum length.

INTEGER: Whole number.

Data Types

DATE:

Calendar date (year, month, day).

TIMESTAMP:

Date and time.

Data Types

BOOLEAN:

True or false.

NUMERIC(p, s):

Exact numeric of selectable precision.

Constraints

Constraints are rules

Applied to columns

To ensure

Data integrity

Constraints

PRIMARY KEY:

Uniquely identifies each row in a table.

FOREIGN KEY:

Ensures referential integrity between tables.

Constraints

NOT NULL:

Ensures that a column cannot have a NULL value.

UNIQUE:

Ensures that all values in a column are unique.

Constraints

CHECK:

Ensures that values in a column

satisfy a specific condition.

Numeric Types

INTEGER or INT

Whole numbers.

Numeric Types

SERIAL:

Auto-incrementing integer

often used for primary keys

Numeric Types

NUMERIC(precision, scale):

Exact numeric values with a specified precision and scale.

FLOAT, REAL, DOUBLE PRECISION:

Floating-point numbers.

Integer Types

smallint:

Stores 2-byte integer values.

Range:

-32768 to 32767.

Integer Types

integer:

Stores 4-byte integer values.

Range:

-2147483648 to 2147483647.

Integer Types

bigint:

Stores 8-byte integer values.

Range:

-9223372036854775808 to

9223372036854775807

Integer Types

```
CREATE TABLE numeric_example (
    id serial PRIMARY KEY,
    small_number smallint,
    medium_number integer,
    large_number bigint
);
```

Integer Types

INSERT INTO numeric_example (small_number, medium_number, large_number) VALUES (100, 20000, 3000000000);

Serial Types

serial:

Auto-incrementing 4-byte integer.

bigserial:

Auto-incrementing 8-byte integer.

Serial Types

```
CREATE TABLE serial_example (
    id serial PRIMARY KEY,
    big_id bigserial
);
```

Decimal and Floating-Point Types

Decimal or numeric:

Variable precision number.

Real:

4-byte floating point number.

Decimal and Floating-Point Types

Double precision:

8-byte floating point number.

Decimal and Floating-Point Types

```
CREATE TABLE decimal_example (
    decimal_value decimal(10, 2),
    real_value real,
    double_value double precision
);
```

Decimal and Floating-Point Types

```
INSERT INTO decimal_example
(decimal_value, real_value, double_value)
VALUES (12345.67, 1.23, 123456789.123456);
```

Character Types

CHAR(n):

Fixed-length character string.

Character Types

VARCHAR(n):

Variable-length character string

with a maximum length of n.

Character Types

TEXT:

Variable-length

character string

with no specific length limit.

Fixed-Length Character

char(n):
Fixed length, blank-padded

```
CREATE TABLE char_example (
    fixed_length char(10)
);
```

Fixed-Length Character

```
INSERT INTO char_example (fixed_length)
VALUES ('abc');
```

Variable-Length Character

varchar(n):

Variable length with a limit.

text:

Variable unlimited length.

Variable-Length Character

```
CREATE TABLE varchar_example (
    variable_length varchar(50),
    unlimited_length text
);
```

Variable-Length Character

```
INSERT INTO varchar_example
(variable_length, unlimited_length)
VALUES ('Hello, World!', 'This is a long text field.');
```

Date/Time Types

DATE:

Calendar date (year, month, day).

TIME [WITHOUT TIME ZONE]:

Time of day (hours, minutes, seconds).

Date/Time Types

TIMESTAMP [WITHOUT TIME ZONE]:

Date and time.

INTERVAL:

Time interval.

Boolean Type

BOOLEAN:

Logical Boolean

TRUE or FALSE

Binary Data Types

BYTEA

Binary data

byte array

Array Types

PostgreSQL supports

Arrays of any data type

INTEGER[]

TEXT[]

Array Types

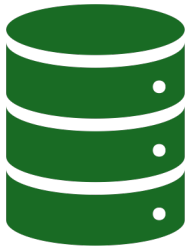
```
-- Stores a list of values

CREATE TABLE array_example (
    integer_array integer[],
    text_array text[]
);
```

Array Types

```
INSERT INTO array_example
(integer_array, text_array)
VALUES ('{1, 2, 3}', '{"apple", "banana", "cherry"}');
```

JSON (JavaScript Object Notation)

Stores



JSON data

as text.

JSON Example

```
CREATE TABLE json_example (
    json_data json
);

INSERT INTO json_example (json_data) VALUES ('{"name": ”Dev", "age": 30, "city": "New York"}');
```

JSONB

Stores JSON data in

Binary format

For faster processing

JSONB Example

```
CREATE TABLE jsonb_example (
    jsonb_data jsonb
);

INSERT INTO jsonb_example (jsonb_data) VALUES ('{"name": ”Dev", "age": 25, "city": "San Francisco"}');
```

Constraints

Constraints are rules applied

To table columns

To enforce data integrity

Ensure the accuracy

Reliability of the data

NOT NULL Constraint

Ensures that a column cannot have a NULL value.

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL
);
```

UNIQUE Constraint

Ensures that all values in a column are unique.

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

PRIMARY KEY Constraint

A combination of NOT NULL and UNIQUE.

Uniquely identifies each row in a table.

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL
);
```

FOREIGN KEY Constraint

Ensures the referential integrity of the data in one table to match values in another table.

```
CREATE TABLE departments (
    department_id SERIAL PRIMARY KEY,
    department_name VARCHAR(50) NOT NULL
);
```

FOREIGN KEY Constraint

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department_id INTEGER REFERENCES
  departments(department_id)
);
```

CHECK Constraint

Ensures that all values in a column satisfy a specific condition.

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    salary NUMERIC(8, 2) CHECK (salary > 0)
);
```

DEFAULT Constraint

Sets a default value for a column when no value is specified.

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    hire_date DATE DEFAULT CURRENT_DATE
);
```

Primary Key

A Column

Combination of columns

Uniquely identifies

each row in a table

Primary Key

Each table can have

only one primary key,

which can consist of

single or multiple columns.

Single Column Primary Key

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);
```

Composite Primary Key

```
CREATE TABLE project_assignments (
    employee_id INTEGER,
    project_id INTEGER,
    PRIMARY KEY (employee_id, project_id)
);
```

Foreign Key

A foreign key is

a column (or a combination of columns)

Establishes a link between

data in two tables.

Foreign Key

Ensures that the value in

Foreign key column(s)

Must match values in

Referenced primary key column(s)

of another table.

Foreign Key Example

```
CREATE TABLE departments (
    department_id SERIAL PRIMARY KEY,
    department_name VARCHAR(50)
);
```

Foreign Key Example

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INTEGER,
    FOREIGN KEY (department_id) REFERENCES
departments(department_id)
);
```

Foreign Key with ON DELETE CASCADE

Ensures When a row in

Referenced table is deleted,

All related rows in

Referencing table are deleted.

Foreign Key with ON DELETE CASCADE

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INTEGER,
    FOREIGN KEY (department_id) REFERENCES departments(department_id) ON DELETE CASCADE
);
```

Filtering and Sorting Data

WHERE Clause

SELECT * FROM employees WHERE department_id = 1;

- Retrieves all columns and rows
- from the employees table
- where the department_id is 1.

Multiple Conditions with AND

```
SELECT * FROM employees
WHERE department_id = 1 AND salary > 50000;
```

- Retrieves rows from the employees table
- where the department_id is 1 and
- the salary is greater than 50000.

Using OR in WHERE Clause

```
SELECT * FROM employees WHERE department_id = 1 OR department_id = 2;
```

- Retrieves rows from the employees table
- where the department_id is either 1 or 2.

Using IN Operator

SELECT * FROM employees WHERE department_id IN (1, 2);

- Retrieves rows from the employees table
- where the department_id is 1 or 2.

Using LIKE Operator for Pattern Matching

SELECT * FROM employees WHERE first_name LIKE 'J%’;

- Retrieves rows from the employees table
- where the first_name starts with 'J'.

Ordering Results with ORDER BY

SELECT * FROM employees ORDER BY last_name ASC;

- Retrieves all columns and rows
- from the employees table and
- sorts the results by last_name
- in ascending order (ASC)

Ordering Results with DESC

SELECT * FROM employees ORDER BY salary DESC;

- Retrieves all columns and rows
- from the employees table and
- sorts the results by salary in
- descending order (DESC)

Aggregate Functions

- Perform calculations on
- Multiple rows
- To return
- Single value

Aggregate Functions

COUNT:

SELECT COUNT(*) FROM employees;

- Returns the number of rows
- in the employees table

SUM

SELECT SUM(salary) FROM employees;

- Returns the sum of
- all salary values
- in the employees table.

AVG

SELECT AVG(salary) FROM employees;

- Returns the average
- salary value in
- the employees table.

MIN

- SELECT MIN(salary) FROM employees;

- Returns the minimum
- salary value in
- the employees table.

MAX

- SELECT MAX(salary) FROM employees;

- Returns the maximum
- salary value in
- the employees table

Summary

Filtering and sorting data

Essential capabilities in SQL

For retrieving specific subsets of data

Arranging them in a desired order

Summary

Aggregate functions provide

Powerful tools for calculating

Summary statistics or

Performing calculations

Across multiple rows of data.

Summary

UNDERSTANDING THESE CONCEPTS

CRUCIAL FOR EFFECTIVE DATA MANIPULATION

ANALYSIS IN DATABASE OPERATIONS

Surendra Panpaliya
Founder and CEO
GKTCS Innovations
https://www.gktcs.com