SQL for DAML

PostgreSQL Installation Guide

Steps to install PostgreSQL on your local system can be found in the below link.

Windows
https://www.postgresqltutorial.com/install-postgresql/

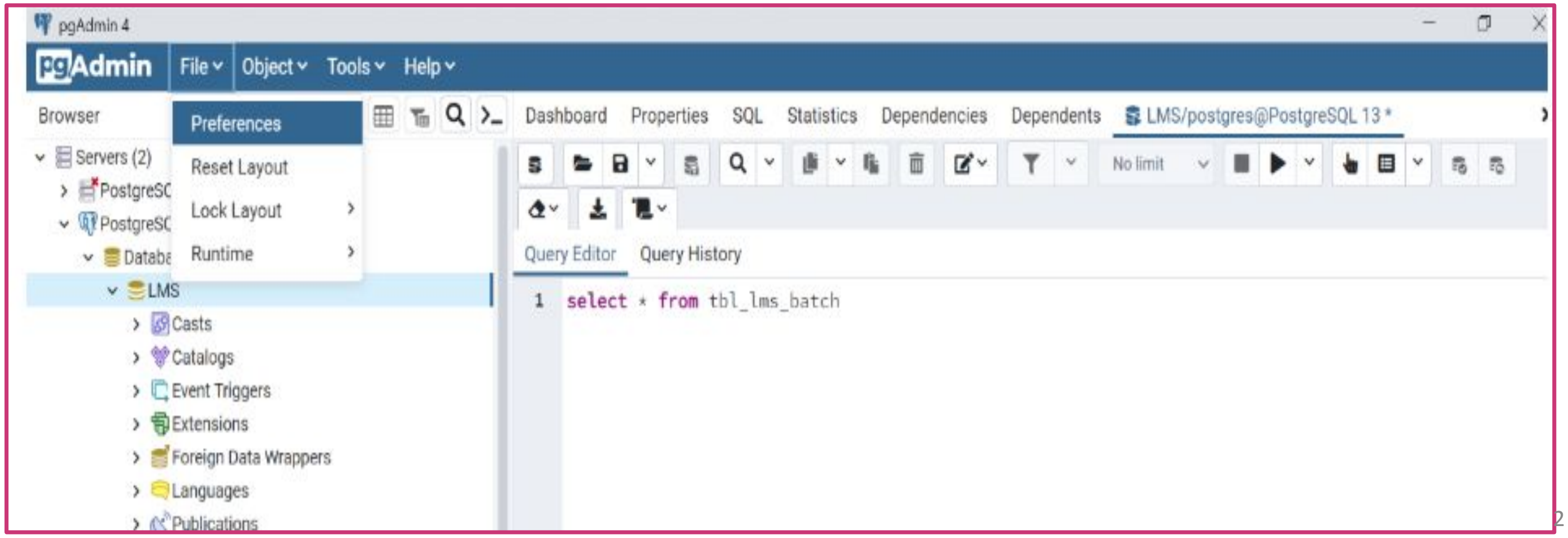
MacOS
https://www.postgresqltutorial.com/install-postgresql-macos/

Steps to connect to PostgreSQL Database Server

https://www.postgresqltutorial.com/connect-to-postgresql-database/
Flow the steps to connect to PostgreSQL database server using pgAdmin

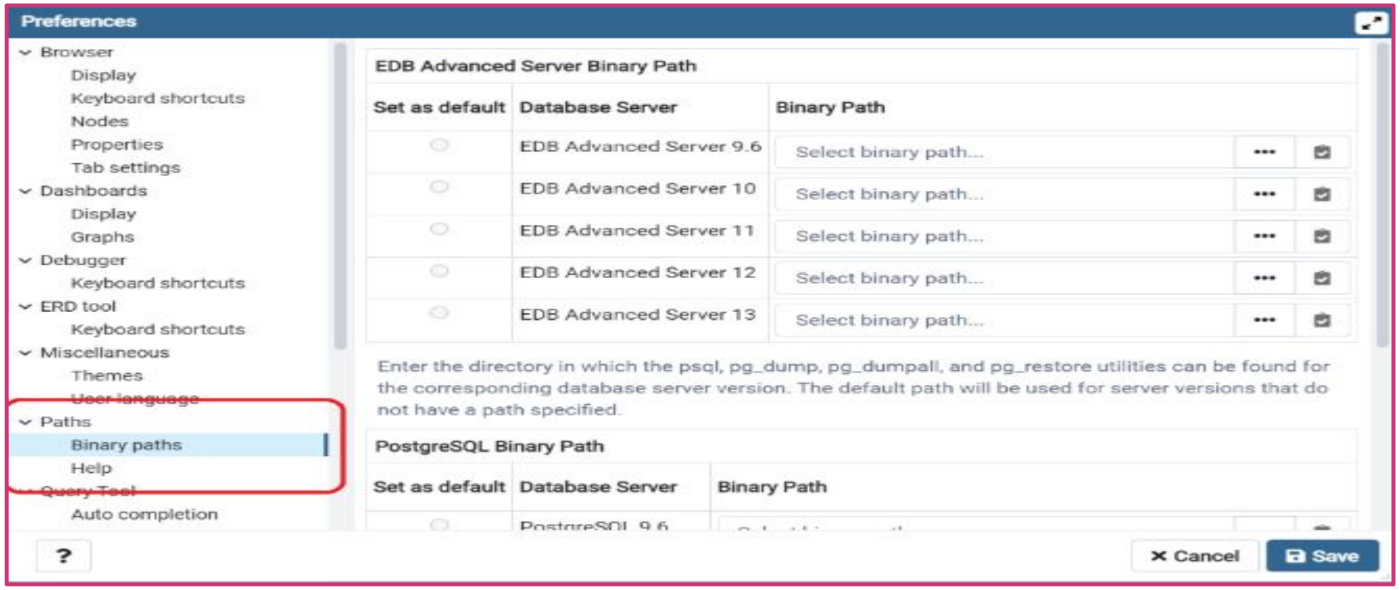
Setting up the binary path

- Click on File->Preferences as shown



```
select * from tbl_lms_batch
```

- Select Binary path as shown



Preferences

- Browser
  - Display
  - Keyboard shortcuts
  - Nodes
  - Properties
  - Tab settings
- Dashboards
  - Display
  - Graphs
- Debugger
  - Keyboard shortcuts
- ERD tool
  - Keyboard shortcuts
- Miscellaneous
  - Themes
  - User language
- Paths
  - Binary paths
  - Help
- Query Tool
  - Auto completion

EDB Advanced Server Binary Path

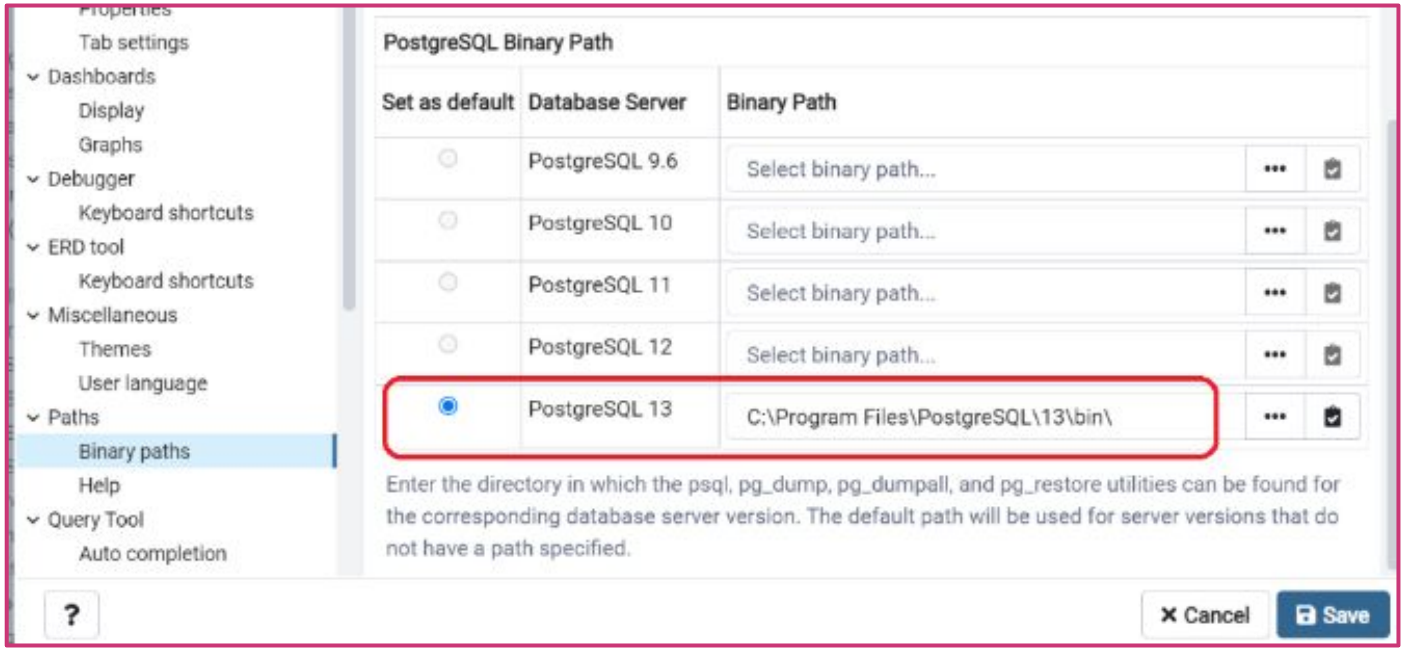
Set as default	Database Server	Binary Path
○	EDB Advanced Server 9.6	Select binary path...
○	EDB Advanced Server 10	Select binary path...
○	EDB Advanced Server 11	Select binary path...
○	EDB Advanced Server 12	Select binary path...
○	EDB Advanced Server 13	Select binary path...

Enter the directory in which the psql, pg_dump, pg_dumpall, and pg_restore utilities can be found for the corresponding database server version. The default path will be used for server versions that do not have a path specified.

PostgreSQL Binary Path

Set as default	Database Server	Binary Path
○	PostgreSQL 9.6	

- Scroll below and set the binary path as shown below



- Dashboards
  - Display
  - Graphs
- Debugger
  - Keyboard shortcuts
- ERD tool
  - Keyboard shortcuts
- Miscellaneous
  - Themes
  - User language
- Paths
  - Binary paths
  - Help
- Query Tool
  - Auto completion

PostgreSQL Binary Path

Set as default	Database Server	Binary Path
○	PostgreSQL 9.6	Select binary path...
○	PostgreSQL 10	Select binary path...
○	PostgreSQL 11	Select binary path...
○	PostgreSQL 12	Select binary path...
●	PostgreSQL 13	C:\Program Files\PostgreSQL\13\bin\

Enter the directory in which the psql, pg_dump, pg_dumpall, and pg_restore utilities can be found for the corresponding database server version. The default path will be used for server versions that do not have a path specified.

STEPS TO DOWNLOAD AND INSTALL SAMPLE DATABASE

Download the PostgreSQL sample database

You can download the PostgreSQL DVD Rental sample database via the following link:

Download DVD Rental Sample Database

The database file is in zip format (dvdrental.zip) so you need to extract it to dvdrental.tar before loading the sample database into the PostgreSQL database server.

Download printable ER diagram

You can download and print the ER diagram for reference while practicing PostgreSQL.

Download the Printable ER Diagram

Load PostgreSQL Sample Database

Go to this link https://www.postgresqltutorial.com/postgresql-getting-started/load-postgresql-sample-database/ and follow steps to

load the DVD Rental database using the pgAdmin.

Chapter 1: Intro to DBMS/RDBMS

1. Introduction of DBMS/RDBMS concepts/definitions
2. Brief intro about popular DB's
3. SQL Intro
4. Create, Insert, Update, Delete, Truncate, Drop SQL statements

Chapter 2: Queries

5. Select, where, Distinct, Order by, Like
6. Alias as, Between, And, OR, Wildcard-%, Null values

Chapter 3: Aggregate Functions and SQL Joins

7. Count, Min/Max, Group by
8. Avg, Sum, Having
9. Inner Join
10. Left Join
11. Right Join

Chapter 4: SQL Joins and Sub queries

12. Self Join
13. Cross Join
14. Sub queries

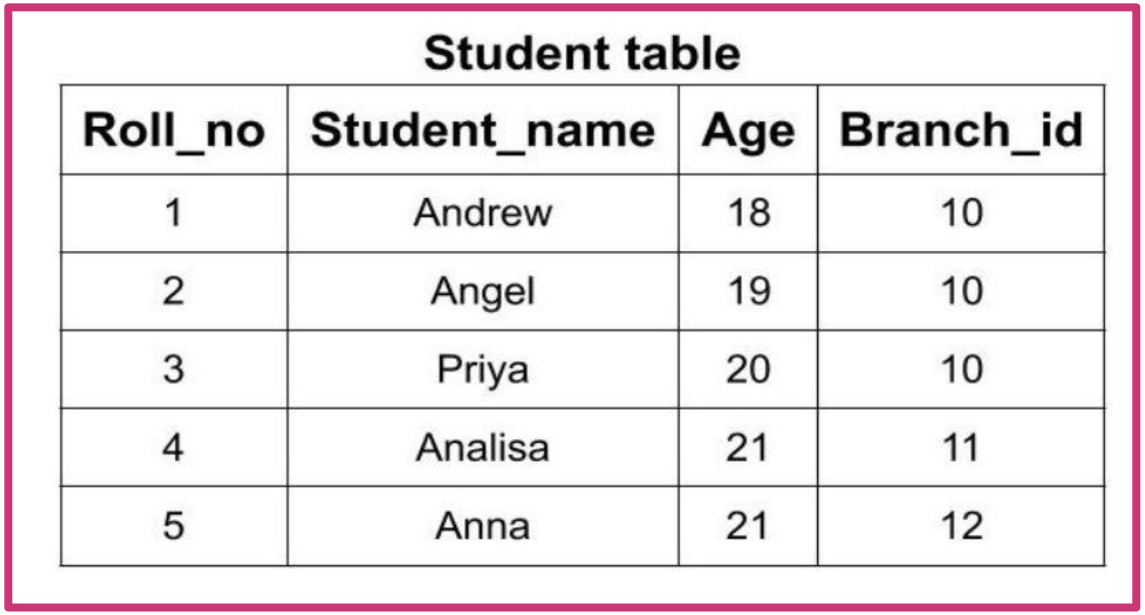
Chapter 1: Intro to DBMS/RDBMS

What is Database?

- A database is typically designed so that it is easy to store and access information.
- A good database is crucial to any company or organization. This is because the database stores all the relevant details about the company such as employee records, transactional records, salary details etc.
- Initial days people are storing the data in the form of files. Ex: Text files . Later Excel files.

What is DBMS?

- Database management system (DBMS):
- We can store the data in the form of tables in the hard Drive.

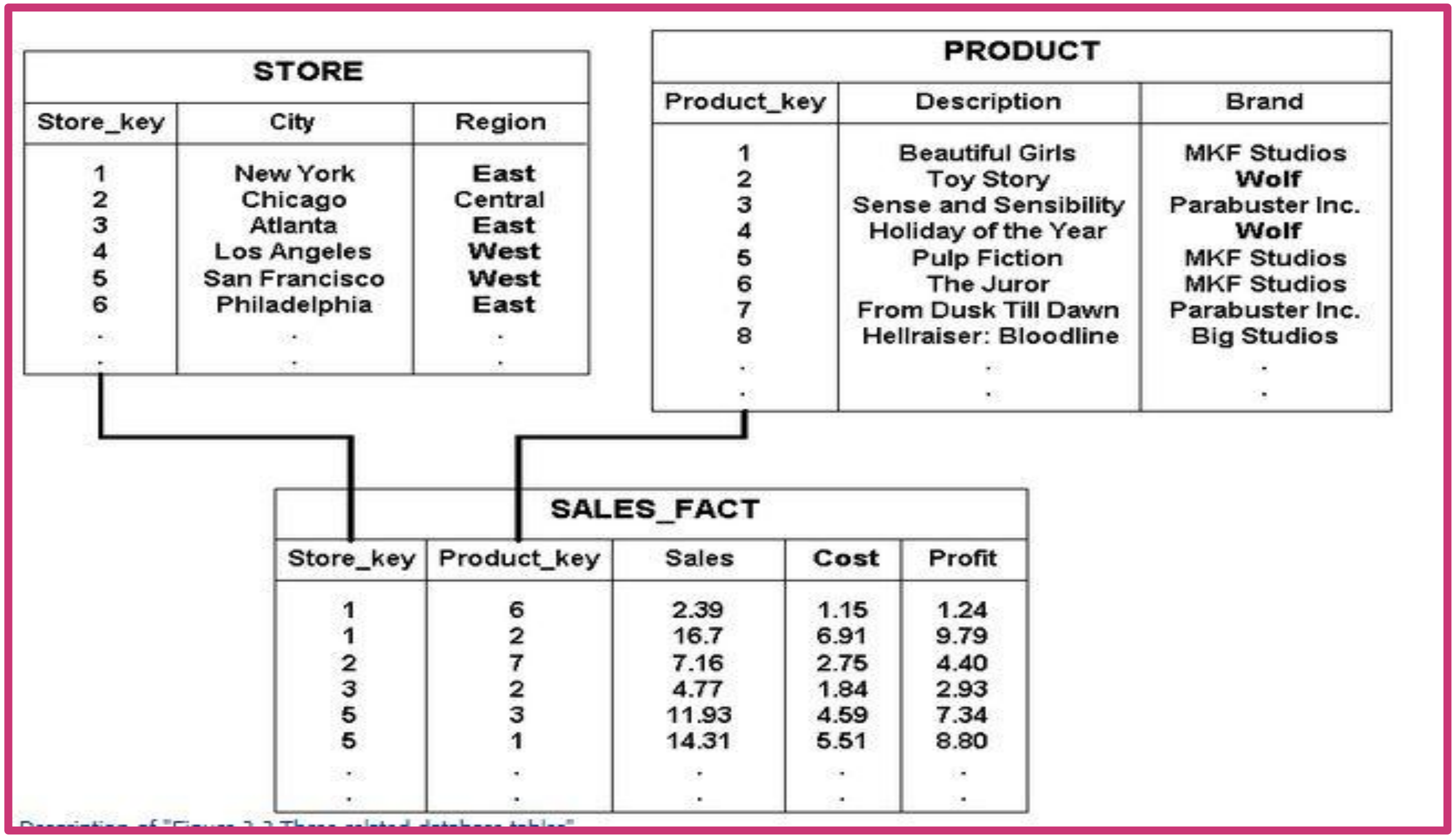


Student table

Roll_no	Student_name	Age	Branch_id
1	Andrew	18	10
2	Angel	19	10
3	Priya	20	10
4	Analisa	21	11
5	Anna	21	12

What is RDBMS?

- RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, PostgreSQL, sand Microsoft Access.



STORE

Store_key	City	Region
1	New York	East
2	Chicago	Central
3	Atlanta	East
4	Los Angeles	West
5	San Francisco	West
6	Philadelphia	East
.	.	.
.	.	.

PRODUCT

Product_key	Description	Brand
1	Beautiful Girls	MKF Studios
2	Toy Story	Wolf
3	Sense and Sensibility	Parabuster Inc.
4	Holiday of the Year	Wolf
5	Pulp Fiction	MKF Studios
6	The Juror	MKF Studios
7	From Dusk Till Dawn	Parabuster Inc.
8	Hellraiser: Bloodline	Big Studios
.	.	.
.	.	.

SALES_FACT

Store_key	Product_key	Sales	Cost	Profit
1	6	2.39	1.15	1.24
1	2	16.7	6.91	9.79
2	7	7.16	2.75	4.40
3	2	4.77	1.84	2.93
5	3	11.93	4.59	7.34
5	1	14.31	5.51	8.80
.	.	.	.	.
.	.	.	.	.

Description of "Figure 2-2 Three related database tables"

What is SQL?

- SQL stands for Structured **Query** Language
- SQL lets you access and manipulate databases

What Can SQL do?

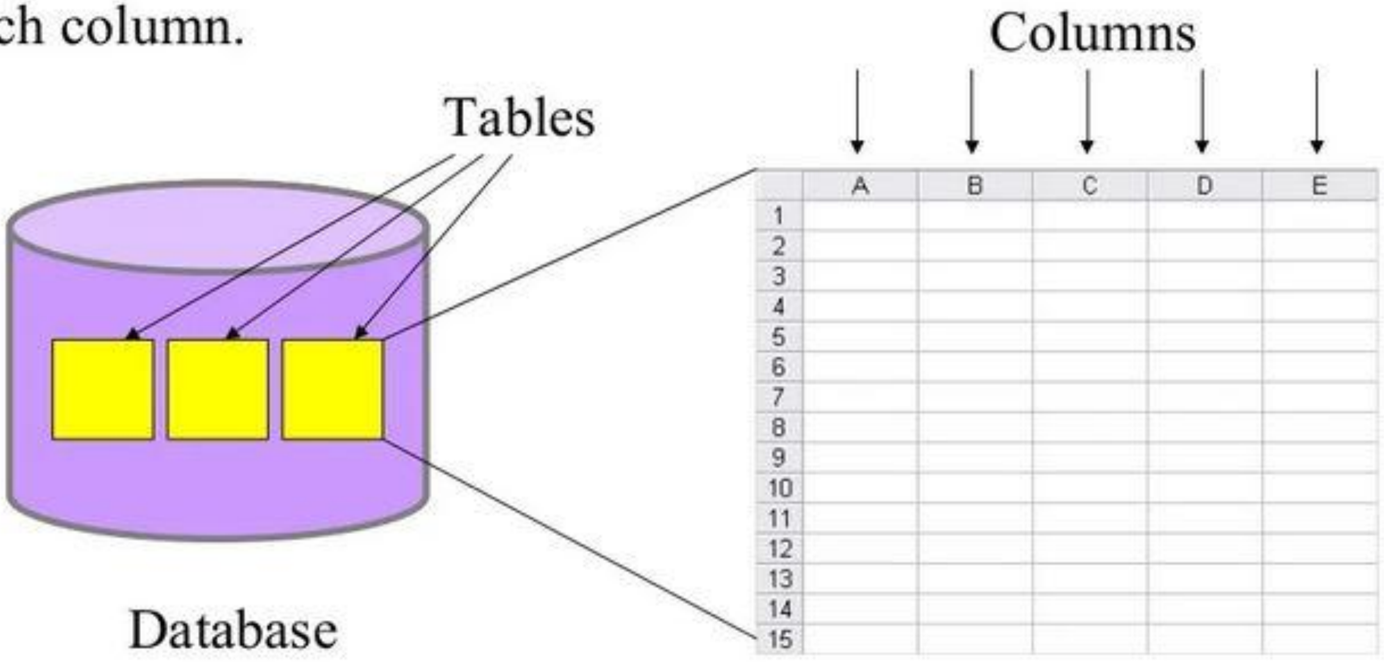
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database

SQL Consists of

- Data Manipulation Language(DML)
  - SELECT, INSERT, UPDATE, DELETE
- Data Definition Language(DDL)
  - CREATE, DROP, ALTER

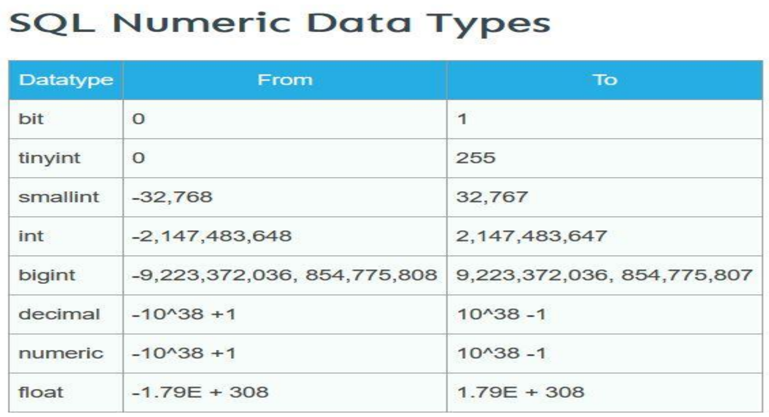
Database Basics

Each table consists of columns and rows. Each column is a field in a record, and there is a column name associated with each column.



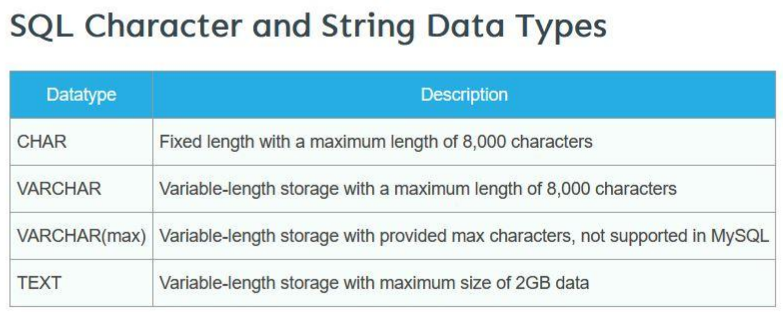
Data Types

Every relational database vendor has its own maximum size limit for different data types, you don't need to remember the limit. Idea is to have the knowledge of what data type to be used in a specific scenario.



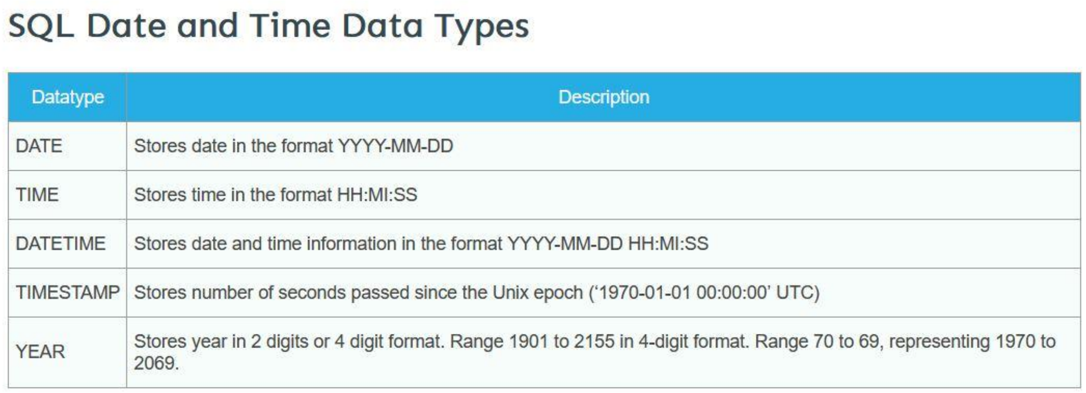
SQL Numeric Data Types

Datatype	From	To
bit	0	1
tinyint	0	255
smallint	-32,768	32,767
int	-2,147,483,648	2,147,483,647
bigint	-9,223,372,036, 854,775,808	9,223,372,036, 854,775,807
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
float	-1.79E + 308	1.79E + 308



SQL Character and String Data Types

Datatype	Description
CHAR	Fixed length with a maximum length of 8,000 characters
VARCHAR	Variable-length storage with a maximum length of 8,000 characters
VARCHAR(max)	Variable-length storage with provided max characters, not supported in MySQL
TEXT	Variable-length storage with maximum size of 2GB data



SQL Date and Time Data Types

Datatype	Description
DATE	Stores date in the format YYYY-MM-DD
TIME	Stores time in the format HH:MI:SS
DATETIME	Stores date and time information in the format YYYY-MM-DD HH:MI:SS
TIMESTAMP	Stores number of seconds passed since the Unix epoch ('1970-01-01 00:00:00' UTC)
YEAR	Stores year in 2 digits or 4 digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069.

Creating table

The CREATE TABLE statement is used to create a new table in a database. SQL is not case sensitive.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Example:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Insert Into Statement

Syntax:

INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)

VALUES (*value1*, *value2*, *value3*, ...);

Example:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table

Syntax:

UPDATE *table_name*

SET *column1* = *value1*, *column2* = *value2*, ...

WHERE *condition*;

Example:

```
UPDATE Customers
   SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
   WHERE CustomerID = 1;
```

SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

Syntax:

DELETE FROM *table_name* WHERE *condition*;

Example:

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste';
```

DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax:

DROP TABLE *table_name*;

Example:

```
DROP TABLE Shippers;
```

Truncate

The TRUNCATE TABLE command deletes the data inside a table, but not the table.

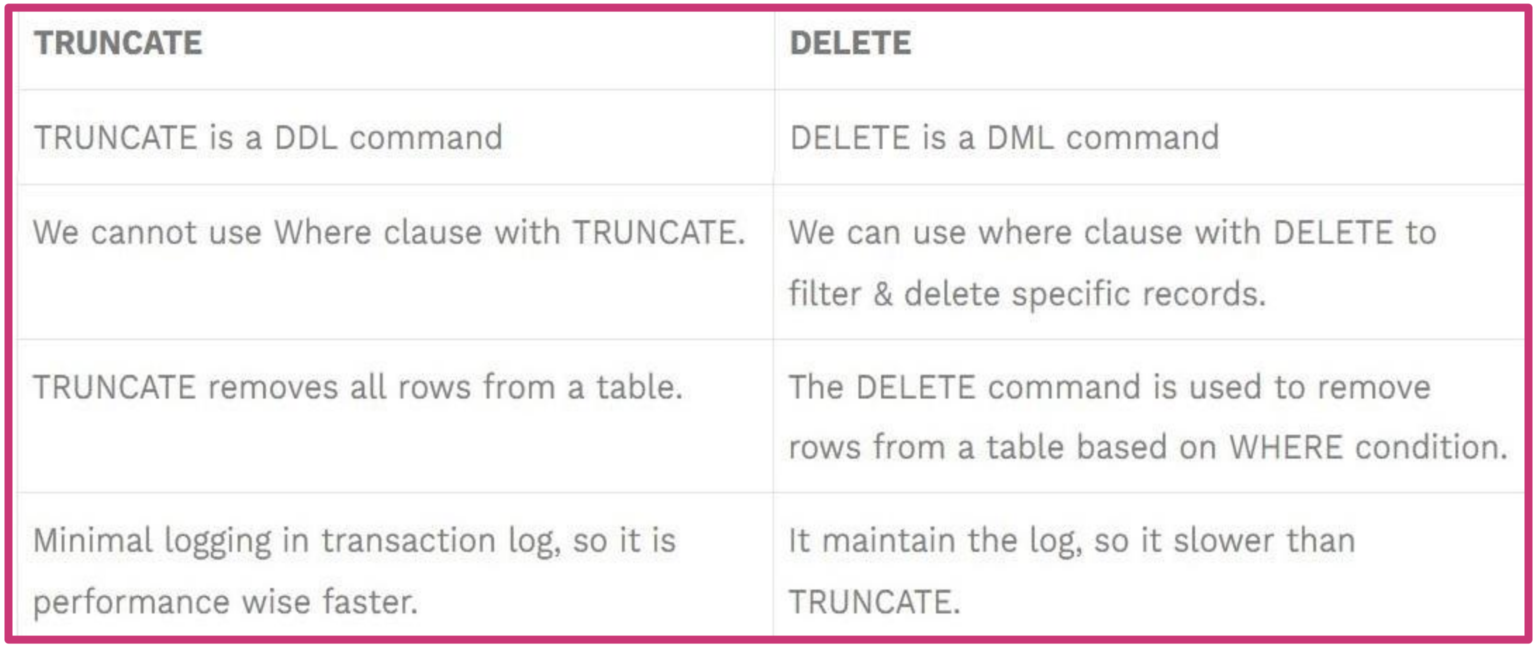
Syntax:

- TRUNCATE TABLE table_name;

Delete Command

- We use SQL Delete command in SQL Server to remove records from a table. We can remove all records or use a Where clause to remove records matching the criteria.

TRUNCATE vs DELETE



TRUNCATE	DELETE
TRUNCATE is a DDL command	DELETE is a DML command
We cannot use Where clause with TRUNCATE.	We can use where clause with DELETE to filter & delete specific records.
TRUNCATE removes all rows from a table.	The DELETE command is used to remove rows from a table based on WHERE condition.
Minimal logging in transaction log, so it is performance wise faster.	It maintain the log, so it slower than TRUNCATE.

1. Practice Questions

1. Write a query to create a simple table **Movies** including columns **movie_id,movie_name,release_year,rental_rate** .
2. Write a query to insert 5 records with your own value into the table **Movies** against each column.The **release_year** must be **2016,2017,2022,2025,2023**.
3. Write a query to insert rows from **film** table to **Movies** table.
4. Write a query to update the **rental_rate** to **9999** for those Movies whose **release_year** is greater than 2022.
5. Delete the rows from the **Movies** table where the **rental_rate** is **9999**.
6. Drop the table **Movies**.

Chapter 2: Queries

SQL SELECT Statement

The SELECT statement is used to select data from a database. (similar to GET in postman)

Syntax:

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

Example:

```
SELECT CustomerName, City FROM Customers;
```

Syntax For all records:

```
SELECT * FROM table_name;
```

WHERE Clause

- The WHERE clause is used to filter records.
- It is used to extract only those records that fulfill a specified condition.
- The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc

Syntax:

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

ORDER BY

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Syntax:

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

LIKE operator

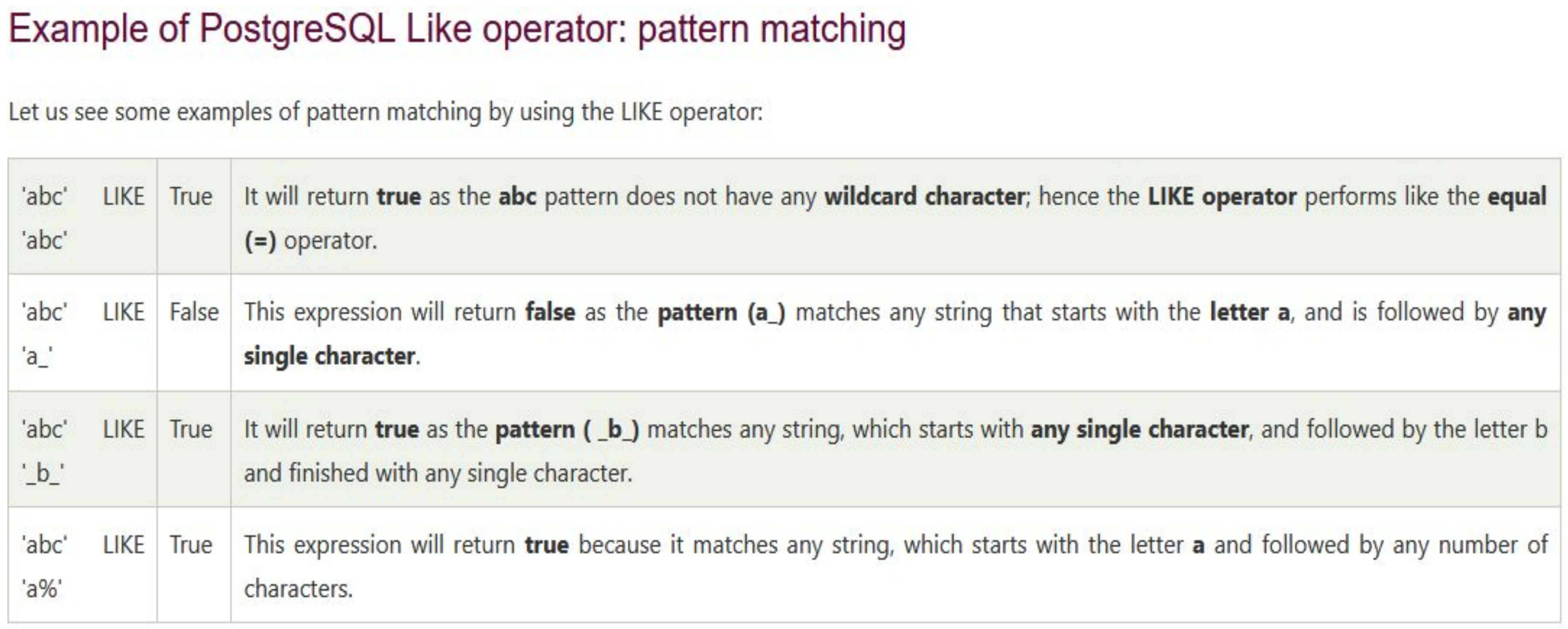
- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;

SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

WILDCARD Characters

- A wildcard character is used to substitute one or more characters in a string.
- Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- PostgreSQL provides you with two wildcards:
  - **Percent sign (%) matches any sequence of zero or more characters.**
  - **Underscore sign (_) matches any single character.**



Example of PostgreSQL Like operator: pattern matching

Let us see some examples of pattern matching by using the LIKE operator:

'abc' LIKE 'abc'	True	It will return **true** as the **abc** pattern does not have any **wildcard character**; hence the **LIKE operator** performs like the **equal (=)** operator.
'abc' LIKE 'a_'	False	This expression will return **false** as the **pattern (a_)** matches any string that starts with the **letter a**, and is followed by **any single character**.
'abc' LIKE '_b_'	True	It will return **true** as the **pattern (_b_)** matches any string, which starts with **any single character**, and followed by the letter b and finished with any single character.
'abc' LIKE 'a%'	True	This expression will return **true** because it matches any string, which starts with the letter **a** and followed by any number of characters.

ALIASES

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- The renaming is a temporary change and the actual table name does not change in the database.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the 'AS ' keyword.
- To reduce the confusion when using joins.

Column Syntax:

```
SELECT column_name AS alias_name
FROM table_name;
```

Table Syntax:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

- NOT BETWEEN: To display the products outside the range of the previous example, use NOT BETWEEN.

Example:

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

BETWEEN Dates Example: The following SQL statement selects all orders with an OrderDate between '01-July 1996' and '31-July 1996':

AND, OR and NOT Operators

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition.
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax:

```
SELECT column1, column2, ...
  FROM table_name
  WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax:

```
SELECT column1, column2, ...

FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax:

```
SELECT column1, column2, ...

FROM table_name
WHERE NOT condition;
```

NULL VALUES

- A field with a NULL value is a field with no value.
- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.
- Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

How to Test for NULL Values?

- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

CONSTRAINTS IN SQL:

SQL constraints are used to specify rules for the data in a table.
Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
Constraints can be column level or table level. **Column level** constraints apply to a column, and **table level** constraints apply to the whole table or more than one columns.
Some of the constraints in SQL are:

- **NOT NULL**: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
- **UNIQUE**: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
- **PRIMARY KEY**: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
- **FOREIGN KEY**: A foreign key is a column or a group of columns in a table that reference the primary key of another table. And this constraint is used to specify a field as Foreign key.
- **CHECK**: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
- **DEFAULT**: This constraint specifies a default value for the column when no value is specified by the user.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

2. Practice Questions

1. Display all the names of the **customers** with **store_id = 1**.
2. Display all the **customer names ordered in descending order of their address.**
3. Display the **addresses** whose phone number is **empty**.
4. Display all the distinct years of the movie released from the films.
5. Display the names of the **customers** whose f**irst names start with A** and **last name ends with ‘l’.**
6. Display the address whose **city id** is in between **30 and 190.**
7. Display the address whose **city id** is in between **30 and 190** and whose **district is Texas.**
8. Display all the **inventories** whose **film id is 1 or 2.**
9. Display all the records whose **city_id is not 200.**
10. Write a query to create a table **Movies** and set **NOT NULL and PRIMARY KEY** constraints for **movie_name and movie_id** .

Chapter 3: SQL Joins and Aggregate Functions

WHAT IS AN AGGREGATE FUNCTION IN SQL?

- An aggregate function in SQL performs a calculation on multiple values and returns a single value.
- SQL provides many aggregate functions that include avg, count, sum, min, max, etc. An aggregate function ignores NULL values when it performs the calculation, except for the count function.

COUNT() Function:

The COUNT() function returns the number of rows in a database table.

Syntax:

SELECT COUNT(*column_name*)
FROM *table_name*
WHERE *condition*;

AVG () Function

The AVG() function returns the average value of a numeric column.

Syntax:

SELECT AVG(*column_name*)
FROM *table_name*
WHERE *condition*;

SUM () Function

The SUM() function returns the total sum of a numeric column.

Syntax:

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

MIN() Function

The MIN() function returns the smallest value of the selected column.

MIN() Syntax:

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() function

The MAX() function returns the largest value of the selected column.

MAX() Syntax:

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

GROUP BY Statement

- The GROUP BY statement groups rows that have the same values into summary rows.
- The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Important Points:

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.
- In the query, GROUP BY clause is placed before ORDER BY clause if used any.

HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

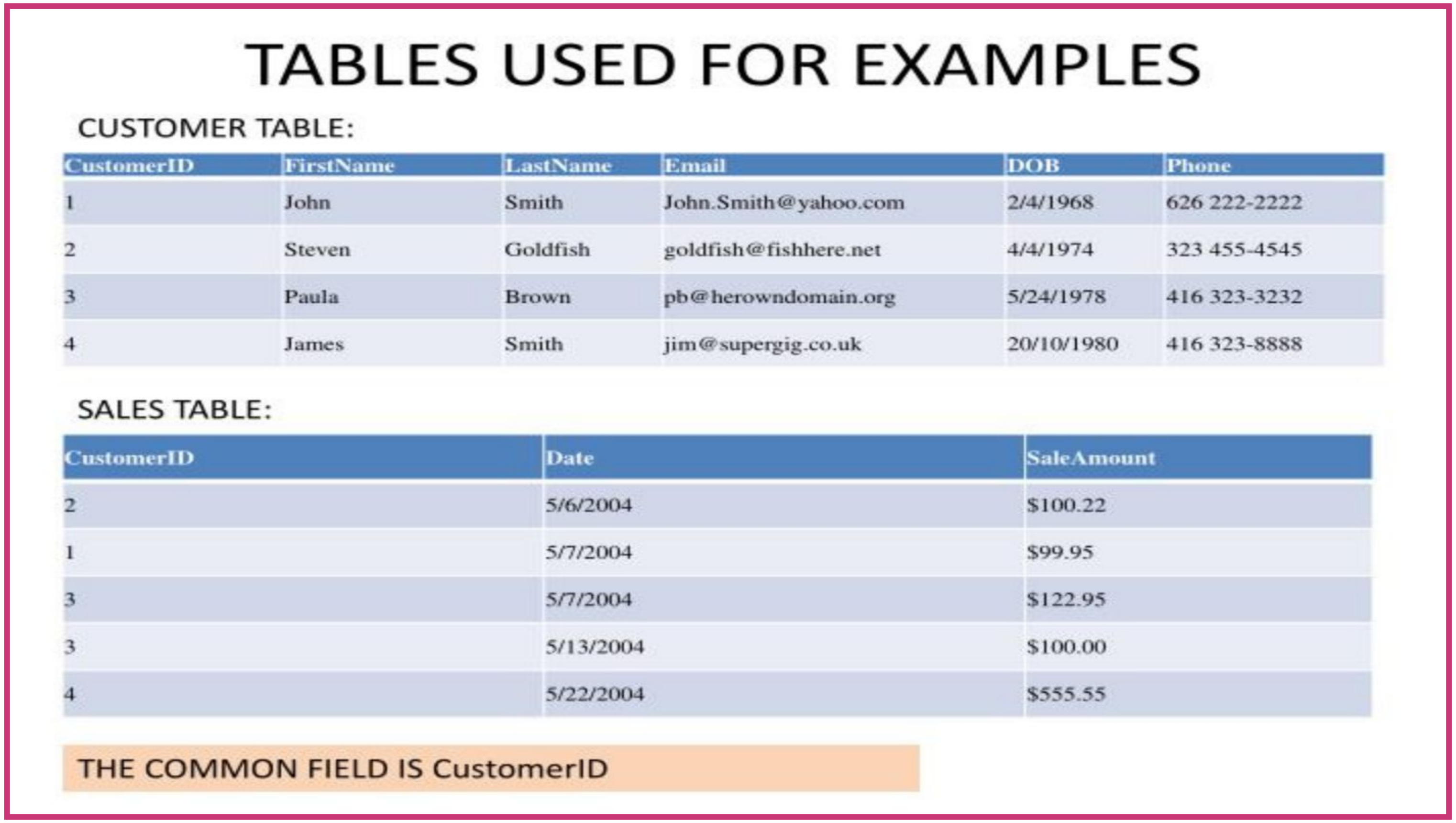
```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

DIFFERENCE BETWEEN HAVING CLAUSE AND GROUP BY CLAUSE :

S.No.	Having Clause	GroupBy Clause
1.	It is used for applying some extra condition to the query.	The groupby clause is used to group the data according to particular column or row.
2.	Having can be used without groupby clause,in aggregate function,in that case it behaves like where clause.	groupby can be used without having clause with the select statement.
3.	The having clause can contain aggregate functions.	It cannot contain aggregate functions.
4.	It restrict the query output by using some conditions	It groups the output on basis of some rows or columns.

Why Use a Join?

- The SQL JOIN clause is used whenever we have to select data from 2 or more tables.
- Join clause used to combine rows from two or more tables based on a related column between them.



TABLES USED FOR EXAMPLES

CUSTOMER TABLE:

CustomerID	FirstName	LastName	Email	DOB	Phone
1	John	Smith	John.Smith@yahoo.com	2/4/1968	626 222-2222
2	Steven	Goldfish	goldfish@fishhere.net	4/4/1974	323 455-4545
3	Paula	Brown	pb@herowndomain.org	5/24/1978	416 323-3232
4	James	Smith	jim@supergig.co.uk	20/10/1980	416 323-8888

SALES TABLE:

CustomerID	Date	SaleAmount
2	5/6/2004	$100.22
1	5/7/2004	$99.95
3	5/7/2004	$122.95
3	5/13/2004	$100.00
4	5/22/2004	$555.55

THE COMMON FIELD IS CustomerID

Primary Key

- A primary key is a column or a group of columns in a table that uniquely identifies the rows in that table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

PRIMARY KEY on CREATE TABLE:

For single column:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

For multiple columns:

```
    CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
  CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

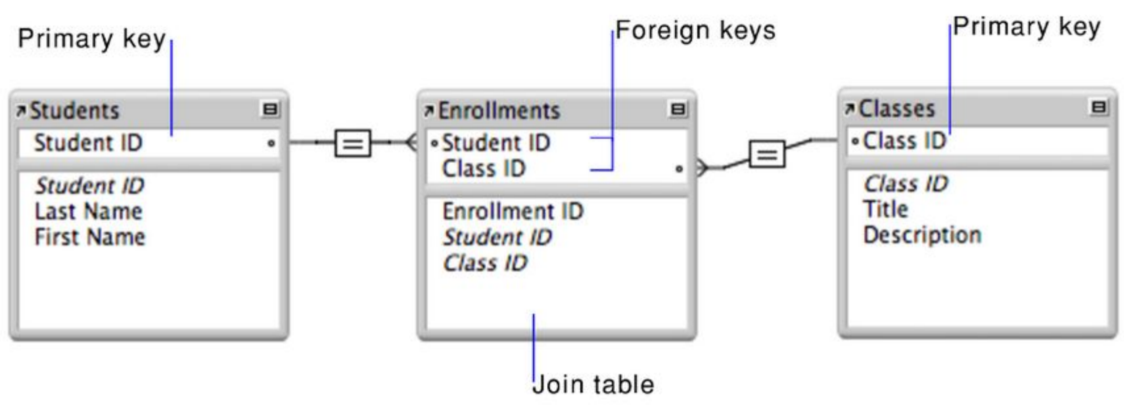
FOREIGN KEY

- A FOREIGN KEY is a column or group of columns in a table, that refers to the PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables. Which means Foreign key don’t allow values other than the values in the parent table.

Specifying foreign key:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

Examples:



Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Joins

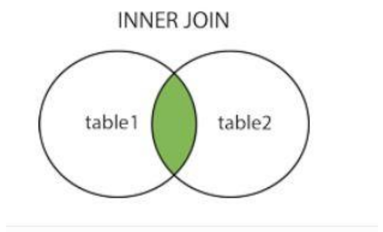
- Join clause used to combine rows from two or more tables. based on a related column between them.
- Using a Primary key and foreign keys which has a common relationship between two tables.

Inner Join:

Returns records that have matching values in both tables.

Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

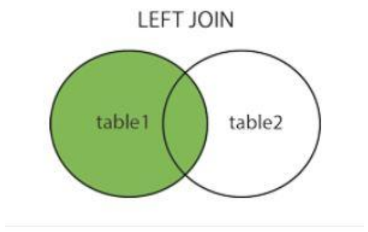


LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

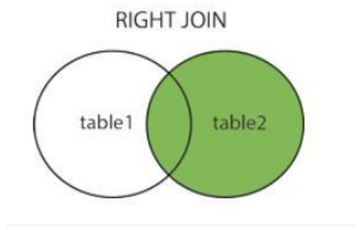


RIGHT JOIN

The RIGHT JOIN returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

Syntax:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

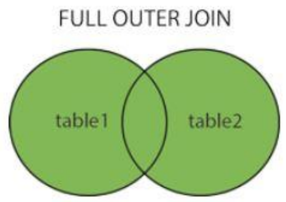


FULL OUTER JOIN

- The FULL OUTER JOIN returns all records when there is a match in left (table1) or right (table2) table records.
- FULL OUTER JOIN and FULL JOIN are the same.

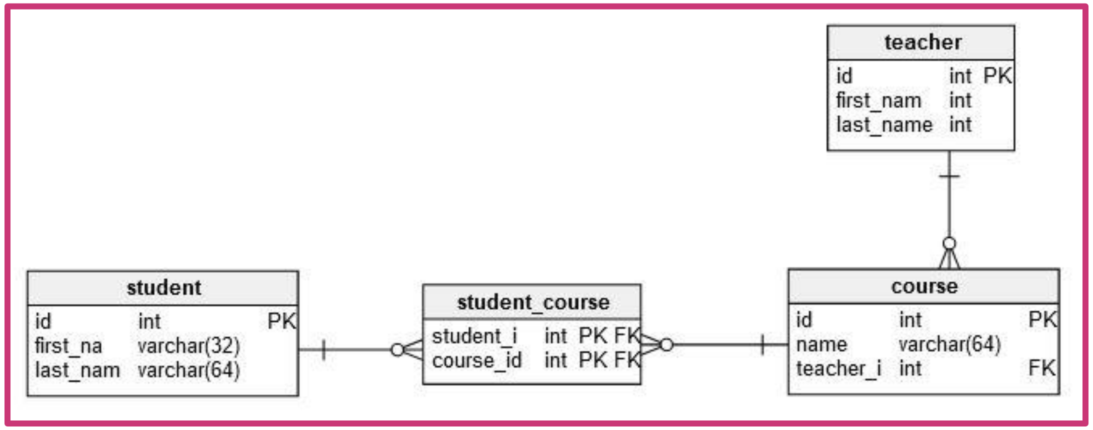
Syntax:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



THREE TABLES JOIN EXAMPLE:

```
SELECT
  student.first_name,
  student.last_name,
  course.name
FROM student
JOIN student_course
  ON student.id = student_course.student_id
JOIN course
  ON course.id = student_course.course_id;
```



3. Practice Questions

1. What is the movie(s) that was rented the most.
2. Display each movie and the number of times it got rented.
3. Show the number of movies each actor acted in.
4. Display the names of the actors that acted in more than 20 movies.
5. For each store, display the number of customers that are members of that store.
6. What is the highest total_payment done.
7. What is the name of the customer who made the highest total payments.
8. How many actors have 8 letters only in their first_names.
9. Display the movies offered for rent in store_id 1 and not offered in store_id 2.
10. Display the movie title for the most rented movie in the store with store_id 1.

Chapter 4: SQL Joins and SubQueries

SELF JOIN

- A self join allows you to join a table to itself. It helps query hierarchical data or compare rows within the same table.
- A self join uses the inner join or left join clause. Because the query that uses the self join references the same table, the table alias is used to assign different names to the same table within the query.

Self join syntax:

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

CROSS JOIN

The CROSS JOIN is used to generate a paired combination of each row of the first table with each row of the second table. This join type is also known as Cartesian join.

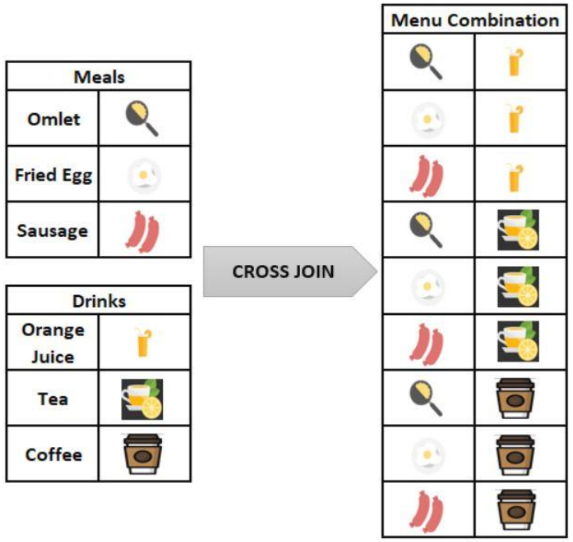
What is the Cartesian Product?

The Cartesian Product is a multiplication operation in the set theory that generates all ordered pairs of the given sets. Suppose that, A is a set and elements are {a,b} and B is a set and elements are {1,2,3}. The Cartesian Product of these two A and B is denoted AxB and the result will be like the following.

AxB ={(a,1), (a,2), (a,3), (b,1), (b,2), (b,3)}

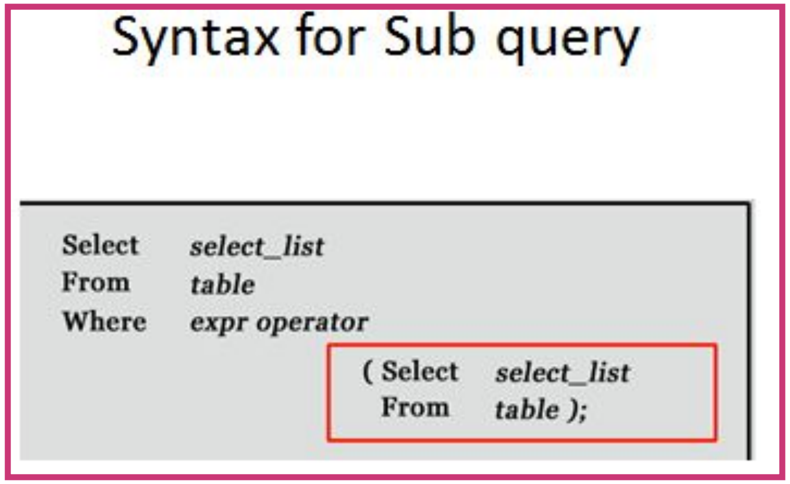
Syntax:

```
SELECT ColumnName_1, ColumnName_2,     ..
FROM Table_1
CROSS JOIN Table_2
```



SUB QUERY

- Definition-- A sub query or Inner Query is a query within Another Query and embedded within Where clause
- It is also called as Nested query.
- Sub query is used to return data that will be used in main query to further restrict data retrieve.
- In PostgreSQL subquery can be nested inside a SELECT, UPDATE, INSERT, DELETE statements along with the expression operators like =, <, >, >=, <=, IN, BETWEEN, etc
- The first statement is known as the outer query, the second is known as the inner query or sub query. The inner query or sub query is normally executed first.
- The sub query can be placed in the following SQL clauses they are WHERE clause, HAVING clause, FROM clause.
- The sub query (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the sub query result.


Syntax for Sub query

```
Select    select_list
From      table
Where     expr operator
                    ( Select   select_list
                      From     table );
```

RULES TO WRITE SUB QUERY

1. Subqueries must be enclosed within parentheses.
2. A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
3. An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
4. Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
5. A subquery cannot be immediately enclosed in a set function.
6. The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.
7. A subquery is always placed on the right side of the comparison operators.
   - Subqueries will with ALL, ANY, IN, or SOME
   - Subqueries will with EXISTS or NOT EXISTS

EXAMPLES OF SUBQUERY

1. Suppose we want to find the films whose rental rate is higher than the average rental rate. We can do it in two steps:

   a. First Find the average rental rate by using the SELECT statement and average function (AVG).

   select * from film;

```
SELECT
    AVG (rental_rate)
FROM
    film;                    Ans :- avg 2.98
```

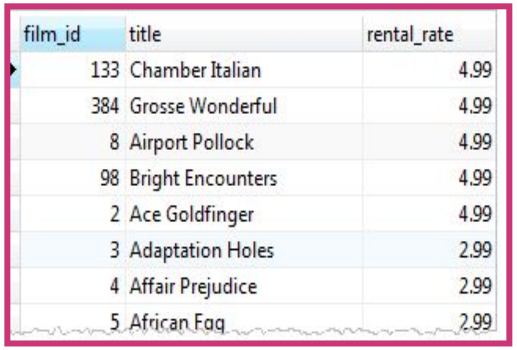
   Means The average rental rate is 2.98

   b. Second Use the result of the first query in the second SELECT statement to find the films that we want.

   Now we can get those films whose rental rate is higher than average rental rate

```
SELECT
    film_id,
    title,
    rental_rate
FROM
    film
WHERE
    rental_rate > 2.98;
```

Answer☐



film_id	title	rental_rate
133	Chamber Italian	4.99
384	Grosse Wonderful	4.99
8	Airport Pollock	4.99
98	Bright Encounters	4.99
2	Ace Goldfinger	4.99
3	Adaptation Holes	2.99
4	Affair Prejudice	2.99
5	African Egg	2.99

Lets try new query for sub query

As earlier mentioned subquery is a query nested inside another query such as SELECT, INSERT, DELETE and UPDATE. In this tutorial, we are focusing on the SELECT statement only.

To construct a subquery, we put the second query in brackets and use it in the WHERE clause as an expression:

```
SELECT
  film_id,
  title,
  rental_rate
FROM
  film
WHERE
  rental_rate > (
          SELECT
                  AVG (rental_rate)
          FROM
                  film
  );
```

The query inside the brackets is called a subquery or an inner query. The query that contains the subquery is known as an outer query.

PostgreSQL executes the query that contains a subquery in the following sequence:

a. First, executes the subquery.

b. Second, gets the result and passes it to the outer query.

c. Third, executes the outer query.

Types of sub queries

1. Scalar Subquery: Is a Single Row Subquery. Which will return one row and one column in the result.

EX:

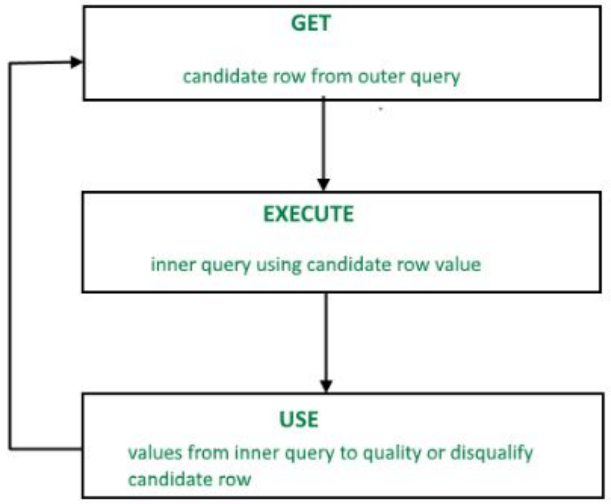
```
SELECT film_id, title, rental_rate --outer query/main query

FROM film

WHERE rental_rate > ( SELECT AVG (rental_rate FROM film );
```

2. Multiple Row Subquery : There are two types of multiple row subquery.

a. Subquery which returns multiple column and multiple row

b. Subquery which returns only 1 column and multiple rows.

3. Correlated Subquery: Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query. Multiple Column Subqueries. Returns one or more columns.


GET
candidate row from outer query
EXECUTE
inner query using candidate row value
USE
values from inner query to quality or disqualify candidate row

A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

Syntax:

```
SELECT column1, column2, ....
FROM table1 outer
WHERE column1 operator
                    (SELECT column1, column2
                     FROM table2
                     WHERE expr1 = outer.expr2);
```

PostgreSQL subquery with EXISTS operator

- The EXISTS operator is a boolean operator that tests for existence of rows in a subquery.
- The following illustrates syntax of the EXISTS operator: EXISTS (subquery)
- A subquery can be an input of the EXISTS operator. If the subquery returns any row, the EXISTS operator returns true. If the subquery returns no row, the result of EXISTS operator is false.
- The EXISTS operator is often used with the correlated subquery.
- The result of EXISTS operator depends on whether any row returned by the subquery, and not on the row contents. Therefore, columns that appear on the SELECT clause of the subquery are not important.

Syntax:

```
SELECT column1 FROM table_1
 WHERE EXISTS( SELECT 1
                      FROM table_2
                      WHERE table_2.column_2 = table_1.column_1);
```

Note that if the subquery returns NULL, the result of EXISTS is true.

Advantages Of Sub query

Sub-queries can be very useful to select rows from a table with a condition that depends on the data in the same or another table.

- Sub queries divide the complex query into isolated parts so that a complex query can be broken down into a series of logical steps.
- It is easy to understand and code maintenance is also at ease.
- Sub queries allow you to use the results of another query in the outer query.
- In some cases, subqueries can replace complex joins.

Disadvantages of Sub query

- We cannot modify a table and select from the same table within a subquery in the same SQL statement.
- In many cases sub query will execute more efficiently if we rewrite the sub-query as a Join

Query optimization:

SQL Query optimization is defined as the iterative process of enhancing the performance of a query in terms of execution time, the number of disk accesses, and many more cost measuring criteria.

- It is achieved using
  - I. Define requirement
  - II. Reduce table size
  - III. Simplify joins
  - IV. Use SELECT Fields FROM Instead of SELECT * FROM– it reduces data fetching
  - V. Use EXISTS() Instead of COUNT()

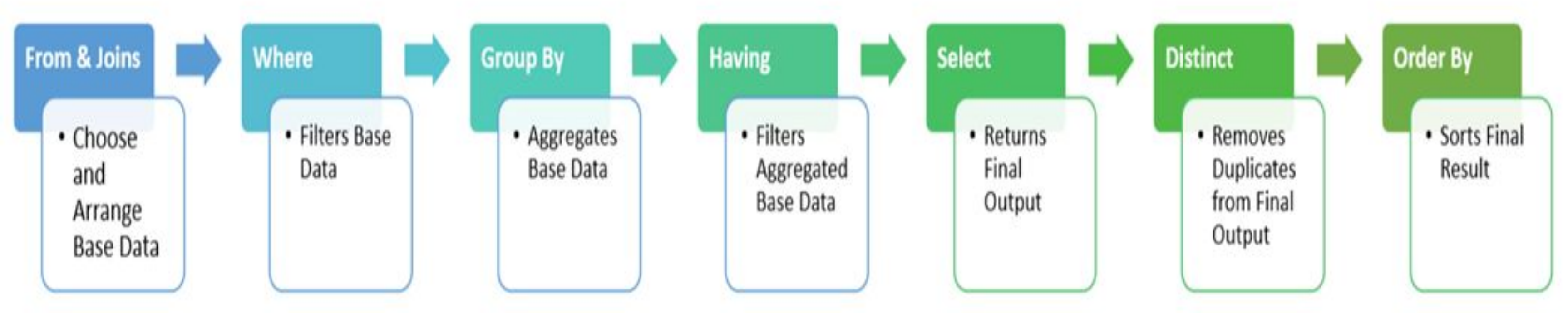
Conclusion :

- A sub query is easier to write, but a join might be better optimized by the server.
- E.g a Left Outer join typically works faster because servers optimize it.

SQL Query Order Of Execution

The SQL order of execution defines the order in which the clauses of a query are evaluated. Some of the most common query challenges people run into could be easily avoided with a clearer understanding of the SQL order of execution, sometimes called the SQL order of operations. Understanding SQL query order can help you diagnose why a query won't run, and even more frequently will help you optimize your queries to run faster.

This order is:



4. Practice Questions

1. Find the names of the customers had bought DVD for rent for more than 5 days?
2. Find the city with maximum number of Staff ?
3. Find the Staff Names in a city "Barcelona"?
4. List all the stores with their address.
5. Find the films which were not rented ?
6. Find the film which has maximum number of inventory ?
7. Find the name of the store which has maximum inventory ?
8. Find the actors who have not acted in a film ?
9. Show the number of rented movies under each rating.

Chapter 5: Views

Views

- Definition-- Views are pseudo-tables.
- A view can represent a subset of a real table, selecting certain columns or certain rows from an ordinary table.
- A view can contain all rows of a table or selected rows from one or more tables.
- A view can be created from one or many tables, which depends on the written PostgreSQL query to create a view.
- The basic CREATE VIEW **syntax** is as follows –

```
CREATE [TEMP | TEMPORARY] VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

- We can include multiple tables in our SELECT statement in very similar way as you use them in normal PostgreSQL SELECT query. If the optional TEMP or TEMPORARY keyword is present, the view will be created in the temporary space. Temporary views are automatically dropped at the end of the current session.
- So if we not use TEMP keyword, the view be created in permanent space and it will be available until we cant? drop it by writing Drop query.

Examples:

1. A simple trial query to create view

```
Create  view Hello As select ‘Hello world’;

select * from Hello;
```

Answer: Hello world

1. Just create a view from Employee table

```
CREATE VIEW  EMP _VIEW AS SELECT emp_id, salary FROM employee;
```

This will create the view for table Employee.

SELECT * FROM COMPANY_VIEW; ---. This query will give same table view with selected columns

1. Create view with Where condition

```
select * from employee;

create view emp_view
As select emp_id, emp_name from employee
where emp_id = 110;

select * from emp_view;
```

View with Join

```
CREATE or REPLACE VIEW current_inventory AS
  SELECT product_name, quantity, category_name
  FROM products
  INNER JOIN categories
  ON products.category_id = categories.category_id
  WHERE quantity > 0;
```

CAUTION:

The CREATE OR REPLACE VIEW statement will work if you are adding columns to the view at the end of the list. However, it will error if you are adding new columns within the existing columns (ie: start or middle of the existing list). In this case, do not use the CREATE OR REPLACE VIEW statement. It is better to drop the view and use the CREATE VIEW statement!

DROP VIEW [IF EXISTS] view_name

view_name

The name of the view that you wish to drop.

IF EXISTS

Optional. If you do not specify this clause and the VIEW does not exist, the DROP VIEW statement will return an error.

E.g. DROP VIEW EMP_VIEW;

5. Practice Questions

1. Create view on table film on columns film_id and title.
2. Create a view to locate the rental_rate is 4.99 .
3. Drop the view for the table film.

Chapter 6: Index

INDEX

- An Index is the structure or object by which we can retrieve specific rows or data faster.
- Index is a data structure which we will built and assign upon existing table that basically look through table and try analyze and summarize to create short cuts to retrieve data.
- There are various types of indexes in SQL server:
    - B-Tree
    - Hash
    - GiST
    - SP-GiST
    - GIN
    - BRIN

The basic **syntax** of CREATE INDEX is as follows –

```
CREATE INDEX index_name ON table_name;
```

There are some ways to create Index

1. Single-Column Indexes

Syntax:

```
CREATE INDEX index_name
   ON table_name (column_name);
```

2. Multicolumn Indexes

Syntax:

```
CREATE INDEX index_name ON table_name (column1_name, column2_name);
```

3. Unique Indexes

Syntax:

```
CREATE UNIQUE INDEX index_name
on table_name (column_name);
```

4. Partial Indexes

Syntax:

```
CREATE INDEX index_name on table_name (conditional_expression);
```

E.G1. select * from customer;

[select table]

2. CREATE INDEX customer_id_index ON customer (customer_id);

[This will create Index on customer_id column]

We can drop Index by using Drop command

Syntax:

```
DROP INDEX index_name;
```

Index with where condition-□

Create Index emp_id_index ON employee(emp_id)

where emp_id=102;

6. Practice Questions

1. Create index on **‘film’** table.
2. Create index on the on the **‘customer’** table using the first_name and the last_name.
3. Write a Query to drop the indexes.

Chapter 7: Trigger

- PostgreSQL Triggers are database callback functions, which are automatically performed/invoked when a specified database event occurs.There are 2 Levels of trigger
  A. A trigger that is marked FOR EACH ROW is called once for every row that the operation modifies.
  B. And for table/statement trigger will execute only once doesn't matter how many rows are there.
- Three types of triggers:
  - DML (Data Manipulation Language) Triggers
  - DDL (Data Definition Language) Triggers
  - Logon Triggers

The basic **syntax** of creating a trigger is as follows –

```
CREATE  TRIGGER /Replace Trigger trigger-name
[Before/After]
[Insert/Update/Delete]
ON table_name
[For each row /for a table]
[Declare statement
Begin
Executable statement(-- Trigger logic goes here....)
End ];
```

Enabling Triggers

Syntax

ENABLE TRIGGER[TRIGGER_NAME|ALL] ON[OBJECT_NAME | DATABASE | ALL SERVER]

Disabling Triggers

Syntax

DISABLE TRIGGER[TRIGGER_NAME|ALL] ON[OBJECT_NAME | DATABASE | ALL SERVER]

Purpose of Triggers

The main purpose of triggers is to automate execution of code when an event occurs.

Triggers are used for several purposes:

Produce additional checking during insert, update or delete operations on the affected table.

They allow us to encode complex default values that cannot be handled by default constraints.

Implement referential integrity across databases.

They allow us to control what actually happens when one performs an insert, update, or delete on a view that accesses multiple tables.

You can calculate aggregated columns in a table using triggers.

7. Practice Questions

1. Create a trigger function while performing insert on the ‘film’ table.
2. Delete the trigger.

Chapter 8: Stored Procedures

What is Stored procedure?

- A stored procedure is block of SQL Statements. We can give name to that block of statements or code and stored in a database.
- We can save stored procedure and can be reuse multiple times.
- So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
- We can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

What is a Purpose of using a procedure?

- Procedures are introduced to give more power to SQL language like a user defined function.
- We use Procedures when we have a requirement which is not possible to achieve just by using SQL queries.

MS SQL Syntax:

```
CREATE PROCEDURE procedure_name
    AS
 sql_statement

GO;
```

PostgreSQL stored procedure Syntax:

```
CREATE OR REPLACE PROCEDURE PROCEDURE_NAME(PARAMETER_NAME, DATATYPE)
LANGUAGE  plpgsql
AS  $$
DECLARE
 -- variable declaration
Begin
   --PROCEDURE body – all logics
END;
$$
```

- Language : PostgreSQL supports to write stored procedure in several different languages we need to specify language. Ex: Python, C, SQL etc.
- Plpgsql : stands for Procedural Language PostgreSQL
- $$: To replace multiple single quotation marks and print as it is what ever we write between $$ $$ symbol.

  Ex: select ‘ I’m John’;
- Declare: where we can declare all your variables.
- Begin: inside begin we can write procedure body and logic

Calling a stored procedure

- To call a stored procedure, you use the CALL statement as follows:
- call stored_procedure_name(argument_list);

Advantages of using stored procedures:

- The stored procedures bring many advantages as follows:
- Reduce the number of round trips between applications and database servers. All SQL statements are wrapped inside a function stored in the PostgreSQL database server so the application only has to issue a function call to get the result back instead of sending multiple SQL statements and wait for the result between each call.
- Increase application performance because the user-defined functions and stored procedures are pre-compiled and stored in the PostgreSQL database server.
- Reusable in many applications. Once you develop a function, you can reuse it in any applications.

Disadvantages of using PostgreSQL stored procedures:

- Slowness in software development because stored procedure programming requires specialized skills that many developers do not possess.
- Difficult to manage versions and hard to debug.
- May not be portable to other database management systems e.g., MySQL or Microsoft SQL Server.

8. Practice Questions

1. Create a table. Write a stored procedure to insert data in to the created table
2. Write a stored procedure to select the customers who rented from

store_id 2.

RESOURCES

- https://www.postgresqltutorial.com/
- https://www.w3schools.com/sql/
- https://join.codecademy.com/learn/learn-sql/
- https://www.youtube.com/watch?v=C93Ed8b8Mhc
- https://www.youtube.com/watch?v=9jLjUIkp78Y