## What are Keys in DBMS?

**KEYS in DBMS** is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables.

RDB - Relational Databases

RDBMS - Relational Database Management System

- eg : Oracle, Postgres, SQLServer, MySql etc

databases, tables, rows/records, columns

student\_details(table)

stud\_id stud\_name stud\_mark

101 ABC 60

102 XYZ 50

SQL - Structured Querying Language,

- compiled and executed

ANSII SQL - standard followed by all sql DB software vendors

dialect – minor changes in queries with different types of software or database

ORM - Object Relational Mapping (don’t have to worry about this in week2) -Spring Data, Hibernate, Ibatis

References

https://www.postgresqltutorial.com/

https://www.postgresql.org/docs/current/tutorial-table.html

https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql#:~:text=Normalization%20is%20the%20process%20to,data%20from%20the%20relational%20tables.

So, in summary, the different types of database keys are:

Natural key: an attribute that can uniquely identify a row, and exists in the real world.

Surrogate key: an attribute that can uniquely identify a row, and does not exist in the real world.

Composite key: more than one attribute that when combined can uniquely identify a row.

Primary key: the single unique identifier for the row.

Candidate key: an attribute that could be the primary key.

Alternate key: a candidate key that is not the primary key.

Unique key: an attribute that can be unique on the table. Can also be called an alternate key.

Foreign key: an attribute that is used to refer to another record in another table.

student\_details

student\_id(PK)

101

102

103

course\_details

course\_id(PK)

501

502

503

student\_course\_details

stud\_id course\_id id🡨-----surrogate key

101 502 1

101 503 2

102 501 3

102 502 4

103 501 5

103 503 6

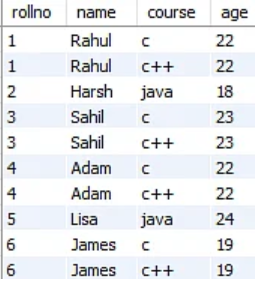
stud\_id course\_id

1st Normal Form (NF)

* A table is referred to as being in its First Normal Form if atomicity of the table is 1.
* Here, atomicity states that a single cell cannot hold multiple values. It must hold only a single-valued attribute.

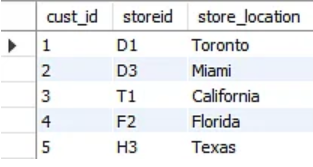


So we have to change above table to below table because above table contains multiple values in column to make it 1st NF.



2nd normal form

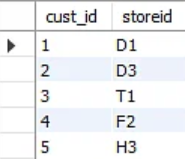
Partial dependency means when we have composite primary key and then non-key values partial dependent on composite primary key is called partial dependency.



Composite Primary key = cust\_id and storeid

Non-key = store\_Loation

Now store location is only depends on storeid not on cust\_id so it is partial depended so now we have to break the table to make fully depended on primary key



Another example for 2nd normal form

employee\_project\_details

emp\_id(PK) , proj\_id(PK), emp\_name, proj\_description, proj\_rate

project rate depends on project\_id(what kind of project is that) as well as employ\_id( manager, dev, tester) so new table below proj\_rate is fully depends on composite primary key

emp\_id(PK), proj\_id(PK), proj\_rate

emp\_name is depends on emp\_id not on proj\_id so new table below

emp\_id, emp\_name

proj\_description depends on proj\_id not on emp\_id so new table below

proj\_id, proj\_description

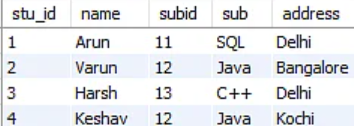
3rd normal form

No Transitive dependency

A🡪B and B🡪C

A🡪C indirectly

So we have to remove transitive dependency to make 3rd NF

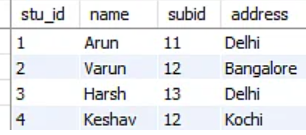
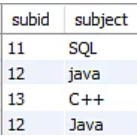


In the aove table stu\_id is PK and we can find subid with stu\_id (means A🡪B)

And subid can find sub (B🡪C)

And sub is not dependent on stu\_id (PK)

So we have to remove transitive dependency



**BCNF boyce codeNormal Form**

Boyce Codd Normal Form is also known as 3.5 NF. It is the superior version of 3NF a

every Right-Hand Side (RHS) attribute of the functional dependencies should depend on the super key of that particular table.

Joins

When we are dividing data or create smaller tables to make it normalization form then it would become difficult to fetch all data related to one particular row. As a result of which, we are using joins. But too many joins also reduce performance of query.

We are also renormalize to reduce number of small table so it can’t lower performance of join query as a result we compromise with normalization not with performance of join query. Means we go with denormalization

HomeWork

2 types of functions

scalar function

- work on individual record

aggregate function

- works on all the record in one go

4 things that a client to contact the server

- protocol – rules or English language two way so they can understand each other

- ip address

- port

- credententials if it is secured

- network

meta commands in postgres

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

\l - lists all the databases

\dt - list all the tables in the current database that you are connected to

\c - connect to a particular database

GENERATED ALWAYS AS IDENTITY- used to generate automatically primary key

Create database sms

\c sms - to connect to sms database

CREATE TABLE product\_details(product\_id int PRIMARY KEY GENERATED ALWAYS AS IDENTITY, product\_name varchar(25), product\_description varchar(200), product\_cost int, product\_image\_url varchar(255));

Select \* from product\_details;

INSERT INTO product\_details(product\_name, product\_description, product\_cost) VALUES(‘Apples’, ‘Juicy Apples’, 15, ‘ ’);

Select \* from product\_details;

INSERT INTO product\_details(product\_name, product\_description, product\_cost) VALUES(‘Bananas’, ‘Delicious Bananas’, 17, ‘’);

Primary key - 2 types to use column constraint and table constraint

* + - 1. Can’t be null
      2. Must be unique
      3. When you make any column as primary key that column is automatically indexed. Which would take extra memory to store indexed files which slows down the speed and process of insertion. Because they have to added to index too.

Disable contraints – when we take a back up of database because constraints slows down the speed of execution as its just copy functions. After copy DB enable it again

\c sample

\dt

CREATE TABLE employee\_deatils(emp\_id int, emp\_name VARCHAR(20), emp\_salary int, PRIMARY KEY (emp\_id));

CREATE TABLE department\_details(dept\_id int, dept\_name VARCHAR (20), PRIMARY KEY (dept\_id));

DROP TABLE employee\_details;

CREATE TABLE employee\_details(emp\_id INT, dept\_id INT, emp\_name VARCHAR(20), emp\_salary INT, PRIMARY KEY (emp\_id), FOREIGN KEY (dept\_id) REFERENCES department\_details(dept\_id));

INSERT INTO department\_details VALUES (501, ‘HR‘ );

INSERT INTO department\_details VALUES (502, ‘IT Development‘ );

INSERT INTO department\_details VALUES (503, ‘QA‘ ) (504, ‘Tester‘ );

INSERT INTO employee\_details values (101,501,'abc',7000),(102,502,'qwerty',7500),(103,503,'pqrs',8450),(104,504,'zinga la la',4578);

**JOINS**

If I want to fetch all data with emp\_id(PK) then with the help of foreign key, we are able to fetch department name with joins. We can fetch all data from multiple tables

Cartesian product

If we have two tables

(A, B , C)

(K, L)

Then Cartesian product is below

(A,K) (A,L) (B,K) (B,L) (C,K) (C,L)

Depending on joining condition and join type that you specify that combination will be fetched

two styles to write joins

- theta style - old

- join style - new

SELECT emp\_id, emp\_name, emp\_salary, dept\_name FROM employee\_details,department\_details;

emp\_id | emp\_name | emp\_salary | dept\_name

--------+-------------+------------+----------------

101 | abc | 7000 | HR

102 | qwerty | 7500 | HR

103 | pqrs | 8450 | HR

104 | zinga la la | 4578 | HR

101 | abc | 7000 | IT Development

102 | qwerty | 7500 | IT Development

103 | pqrs | 8450 | IT Development

104 | zinga la la | 4578 | IT Development

101 | abc | 7000 | QA

102 | qwerty | 7500 | QA

103 | pqrs | 8450 | QA

104 | zinga la la | 4578 | QA

101 | abc | 7000 | Scrum Master

102 | qwerty | 7500 | Scrum Master

103 | pqrs | 8450 | Scrum Master

104 | zinga la la | 4578 | Scrum Master

(16 rows)

SELECT emp\_id, emp\_name, emp\_salary, dept\_name FROM employee\_details,department\_details where employee\_details.dept\_id=department\_details.dept\_id;

emp\_id | emp\_name | emp\_salary | dept\_name

--------+-------------+------------+----------------

101 | abc | 7000 | HR

102 | qwerty | 7500 | IT Development

103 | pqrs | 8450 | QA

104 | zinga la la | 4578 | Scrum Master

(4 rows)

SELECT emp\_id, emp\_name, emp\_salary, dept\_name FROM employee\_details emp INNER JOIN department\_details dept ON emp.dept\_id=dept.dept\_id;

emp\_id | emp\_name | emp\_salary | dept\_name

--------+-------------+------------+----------------

101 | abc | 7000 | HR

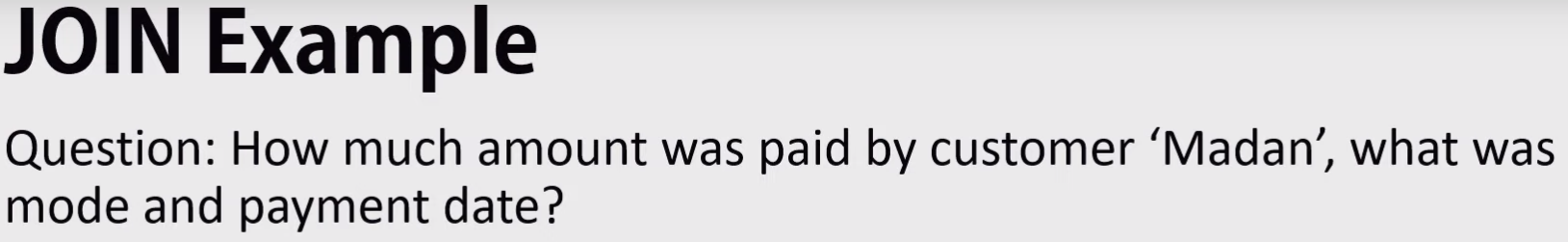
102 | qwerty | 7500 | IT Development

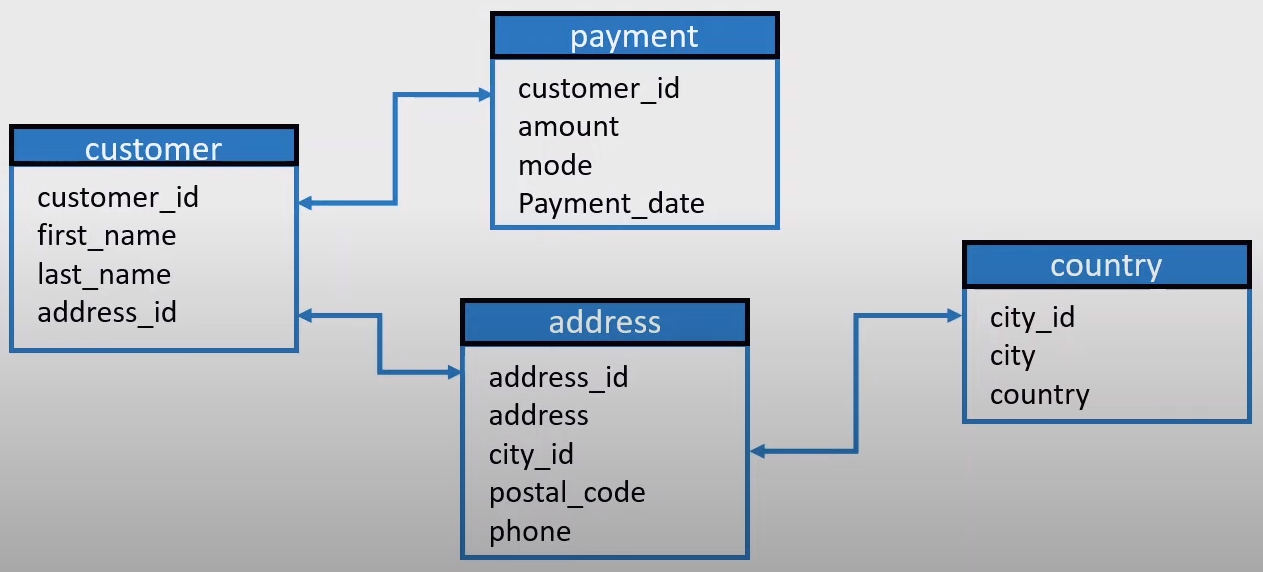
103 | pqrs | 8450 | QA

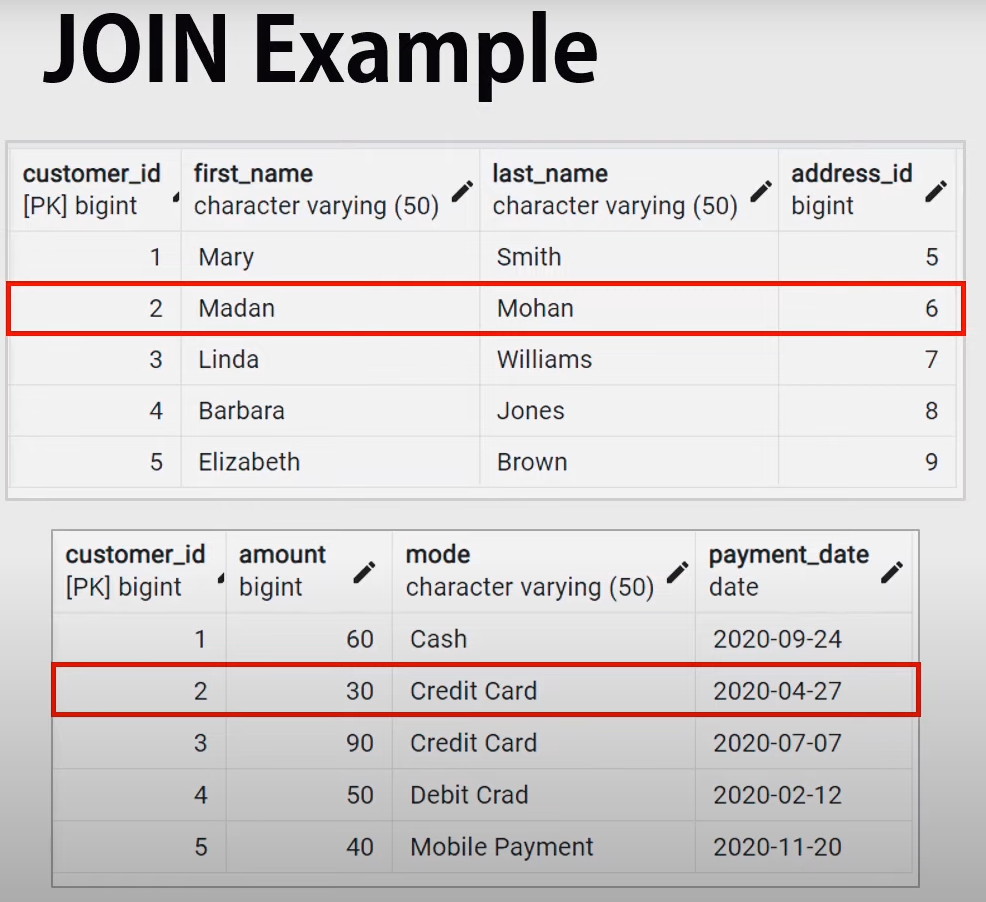
104 | zinga la la | 4578 | Scrum Master

(4 rows)

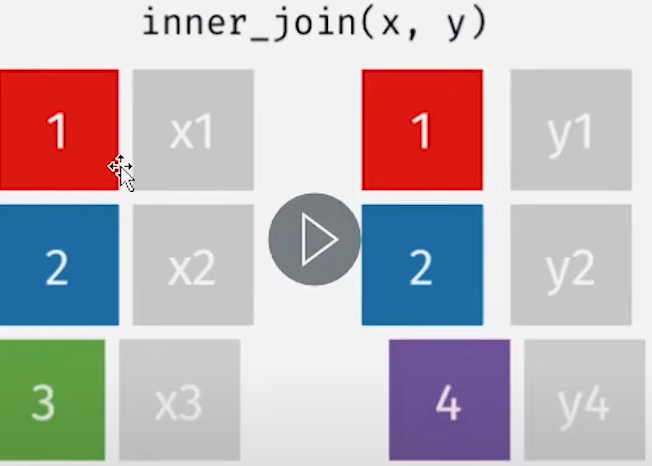
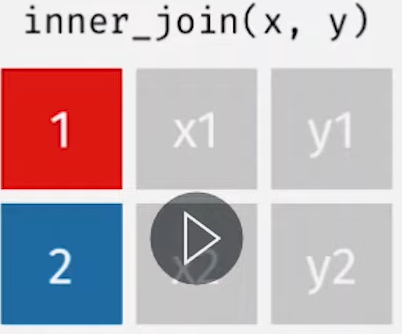
Where emp is alisis

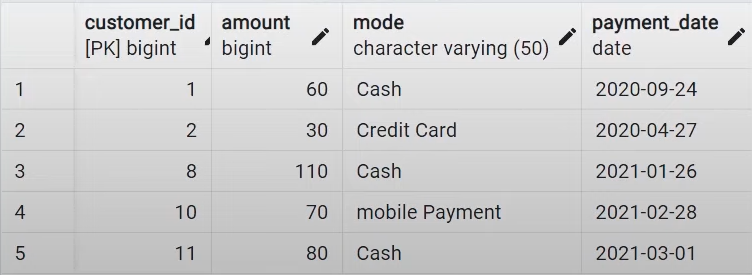
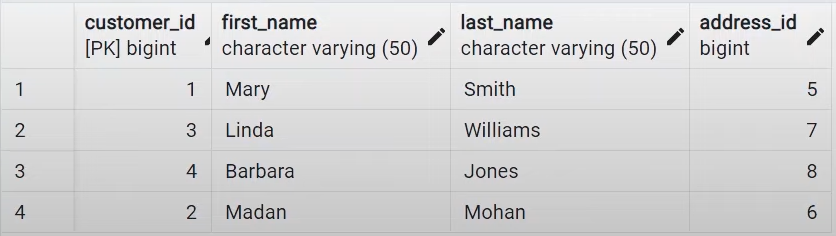


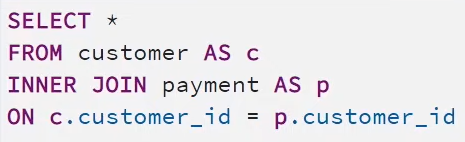




Inner Join

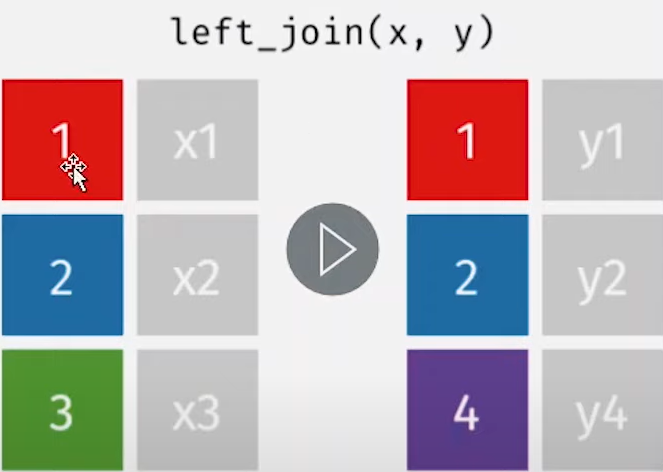






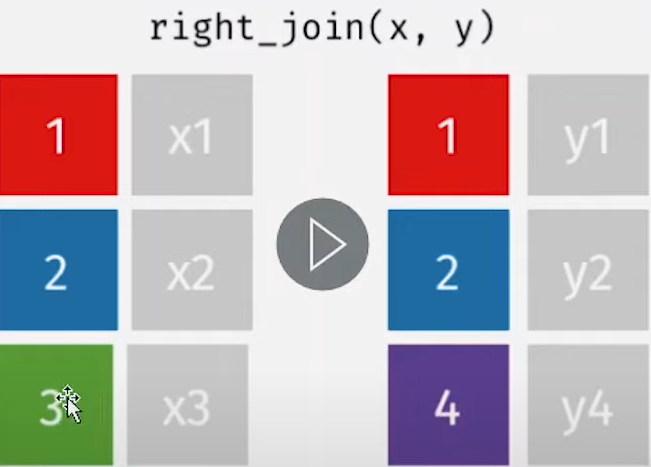
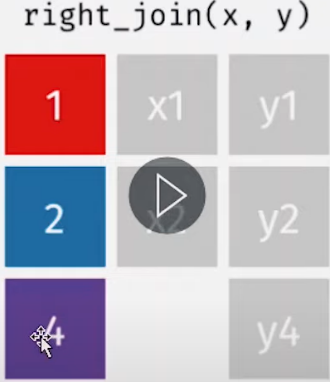


Left Join:

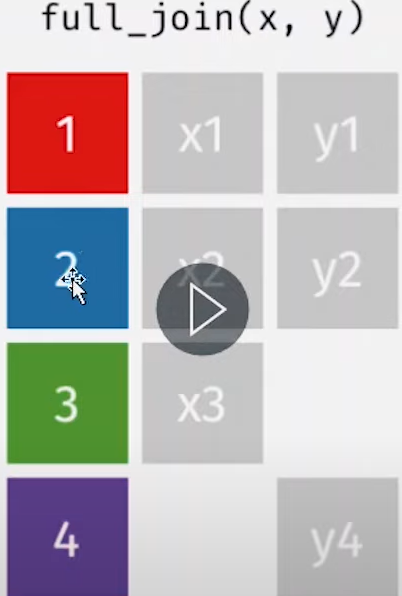


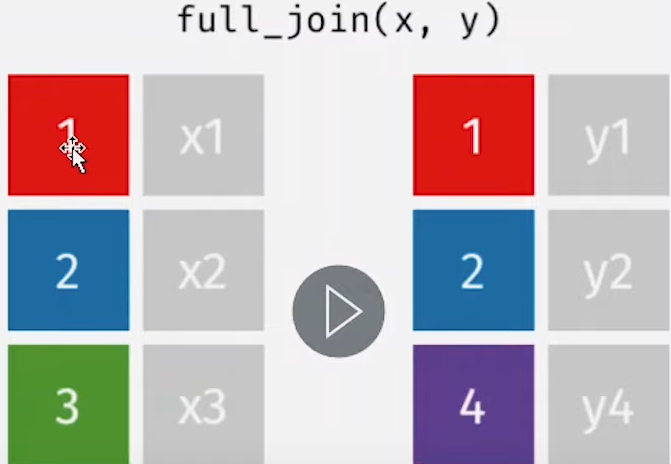


Right Join

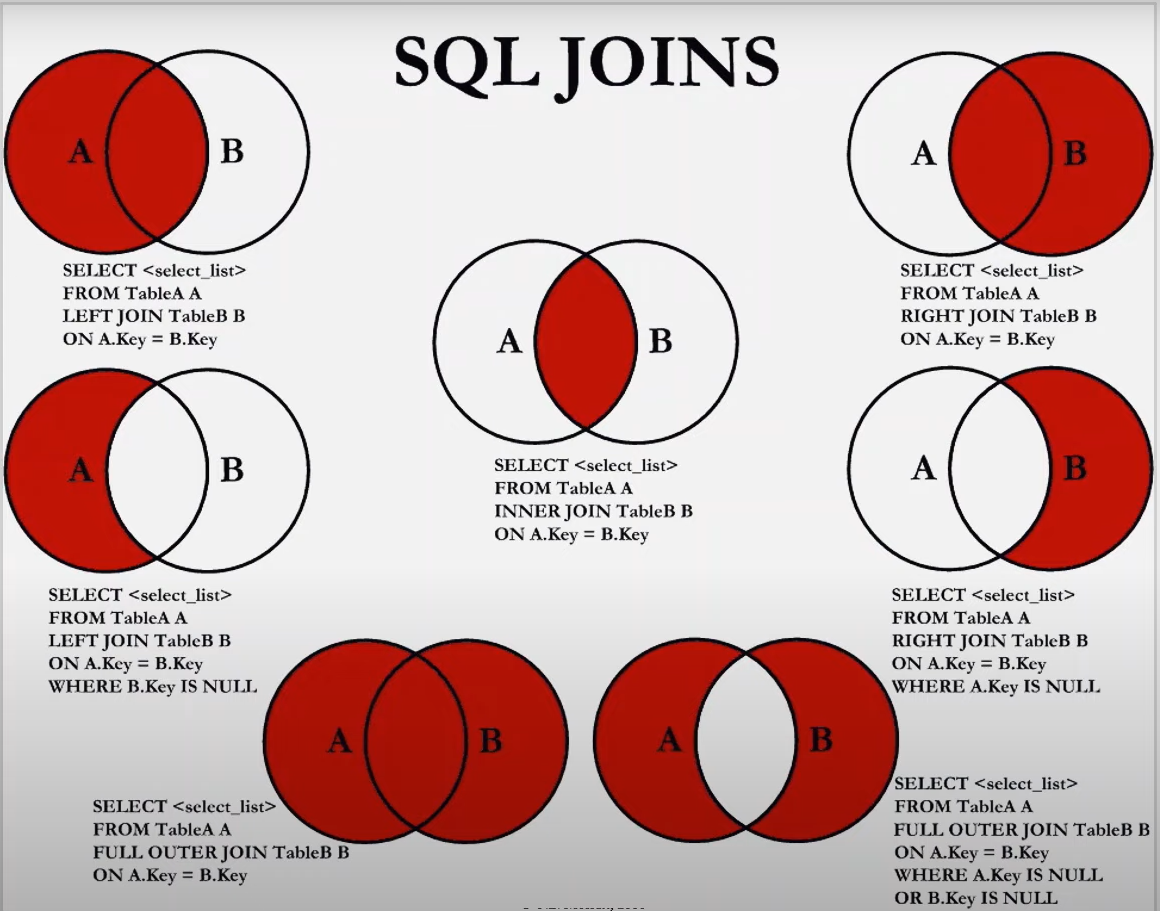












Self join example

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

write a query to display employe names and the names of their manager

CREATE TABLE emp\_manager\_details(emp\_id INT, emp\_name VARCHAR(20), emp\_salary INT, mgr\_id INT, PRIMARY KEY(emp\_id), FOREIGN KEY(mgr\_id) REFERENCES emp\_manager\_details(emp\_id));

INSERT INTO emp\_manager\_details VALUES(101,'ABC', 10000, 101);

INSERT INTO emp\_manager\_details VALUES(102,'XYZ', 8000, 101);

INSERT INTO emp\_manager\_details VALUES(103,'LMN', 8000, 101);

INSERT INTO emp\_manager\_details VALUES(104,'QWE', 8000, 102);

select \* from emp\_manager\_details;

SELECT emp1.emp\_name, emp2.emp\_name manager\_name FROM emp\_manager\_details emp1 INNER JOIN emp\_manager\_details emp2 ON emp1.mgr\_id=emp2.emp\_id;

emp\_id emp\_name mgr\_id

101 ABC 103

102 XYZ 103

103 LMN 104

104 QWE 104

emp\_id emp\_name mgr\_id

101 ABC 103

102 XYZ 103

103 LMN 104

104 QWE 104

boolean result=stmt.execute(query); - for DDL

int rows=stmt.executeUpdate(query); - for DML

ResultSet resultSet=stmt.executeQuery(query); - for DQL

\

working with statement

Statement stmt = conn.createStatement();

String query = "INSERT INTO product\_details(product\_name, product\_description, product\_cost, product\_image\_url) VALUES ('"+productPojo.getProductName()+"', '"+productPojo.getProductDescription()+"', "+productPojo.getProductCost()+", '')";

int rowsAffected = stmt.executeUpdate(query); // 1 Hit, 1 compilation, 1 execution

// total - 1H, 1C, 1E

// for 100 product insertions, analysis - 100 H, 100 compilation, 100 excecutions

working with prepared statement

- pre-compiled statements

- compiled once and executed many times

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

PreparedStatement pstmt = con.prepareStatement("INSERT INTO product\_details(product\_name, product\_description, product\_cost, product\_image\_url) VALUES (?, ?, ?, ?)");// 1 Hit,1 Compilation

pstmt.setString(1, productPojo.getProductName());

pstmt.setString(2, productPojo.getProductDescription());

pstmt.setInt(3, productPojo.getProductCost());

pstmt.setString(4, "");

int rowsAffected = pstmt.executeUpdate();// 1 Hit, 1 Execution

// total - 2H, 1C, 1E

// for 100 product insertions, analysis - 101 H, 1 compilation, 100 excecutions

5 Categories of SQL queries

- DDL (Data Definition Language)

- deal with the structure of the table

- create, alter, drop, truncate

- DML (Data Manipulation La nguage)

- deal with the data in the table

- insert, update, delete

- TCL (Transaction Control language)

- commit, rollback, savepoints

Commands in PSQL – begin, commit, rollback.

conn.setAutoCommit(**false**); // marks the begining of the transaction

**int** rowsAffected1 = stmt.executeUpdate(query1);

// anything could happen here, maybe DB server is down or network is down

**int** rowsAffected2 = stmt.executeUpdate(query2);

conn.commit(); // makes the temporary changes permanent, marks the end of a successfull transaction

- DCL (Data Control language)

- granting and revoking priveliges to the users

- grant, revoke

Constraints are rules for data in tables

Normalization is a way to split raw data into multiple tables so we have consistent data in our hands

ACID properties for transactions -very strong interview question

ACID

A-antomicity

C-consistency

I-Isolation

PL/SQL - stored procedure, functions, triggers

Two types of functions in PSQL

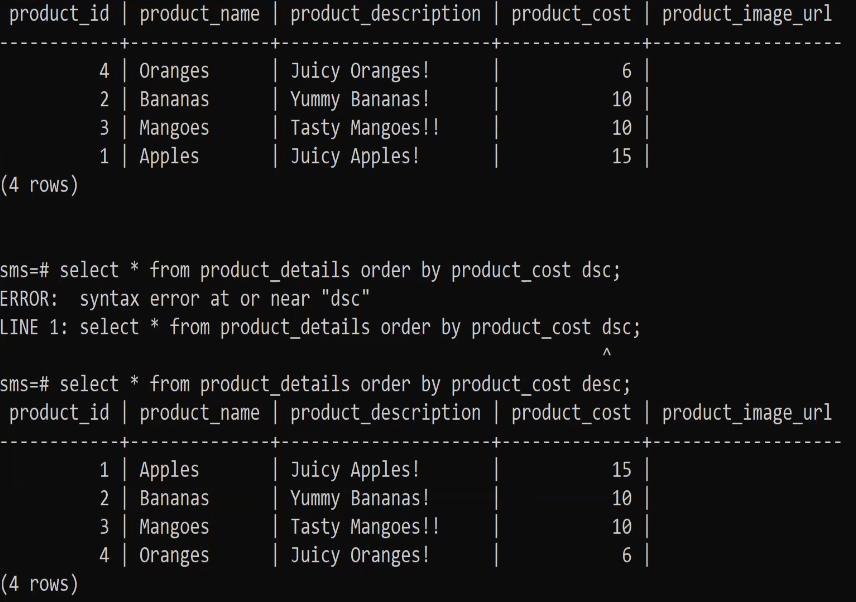
Scaller Funtions

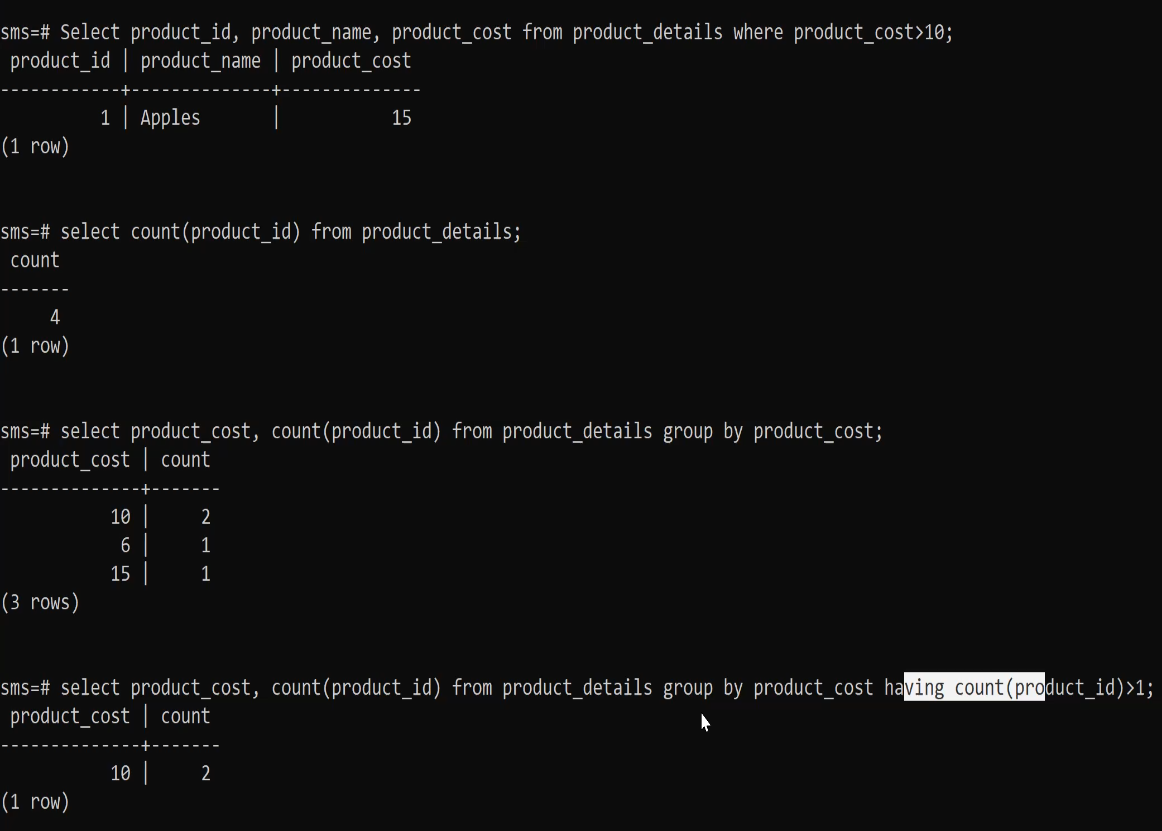
* Works on single row/tupple/record
* Moves rows/tuples one by one

Aggaregate Functions

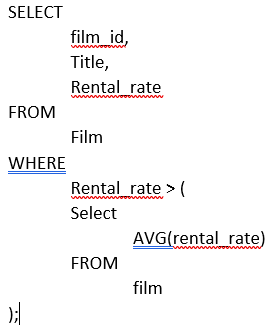
* Works on multiple records
* - DQL (Data Querying Language)
* - select
* clause
* - projection
* - from clause
* - restriction - where clause
* - grouping
* - grouping restriction - having clause
* - order by clause

Projection means what you want to fetch





SubQuery – when a query works on an unknown value/ computed value, we use subquery to compute that value

* Inner query (sub query) and outer query
* First executes the subquery
* Secondly, gets the result and passes it to the outer query
* Third, executes the outerquery.

For example

SELECT

film\_id,

Title,

Rental\_rate

FROM

Film

WHERE

Rental\_rate > 2.8

Look For CORELLATED SUBQUERY and view

* For every record of the outer query, the inner query works

View – is giving name or reference to complex queries. So we can use that view name instead of wring complex queries again and again. It just kind for reference and It’s virtual

* Create View, Alter View and Drop View

Index-

* Used generally column which are frequently used in the where clause
* Will extra overhead but also improve efficiency
* Can create indexes on more than one column(may be where clause frequently works with those 2 columns together)
* Will takes extra memory. Its similar type od indexing like books topic name and page number
* Create Index, Alter Index and Drop Index

Clients

* pgAdmin
* db work