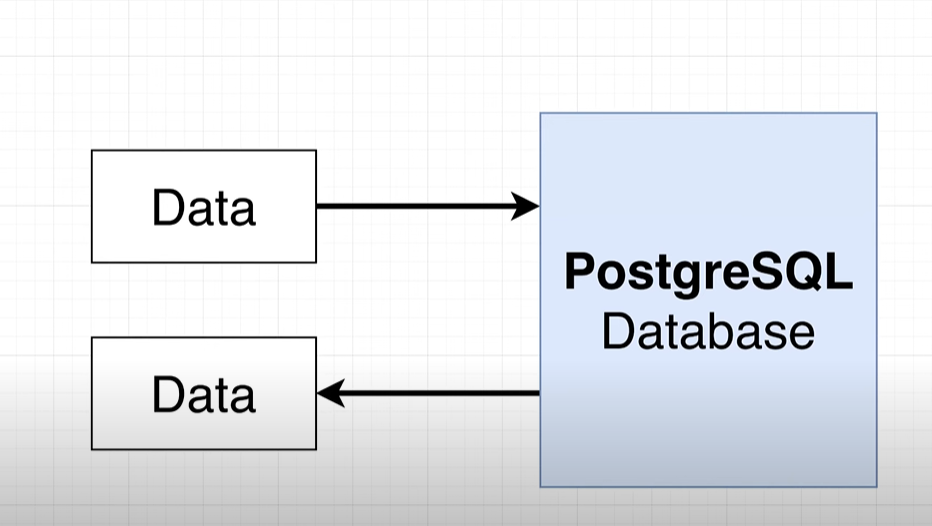
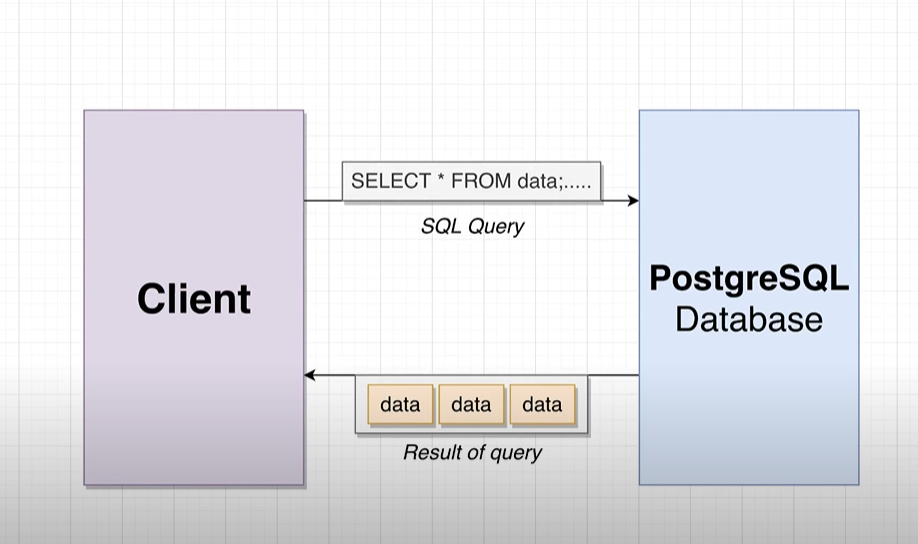
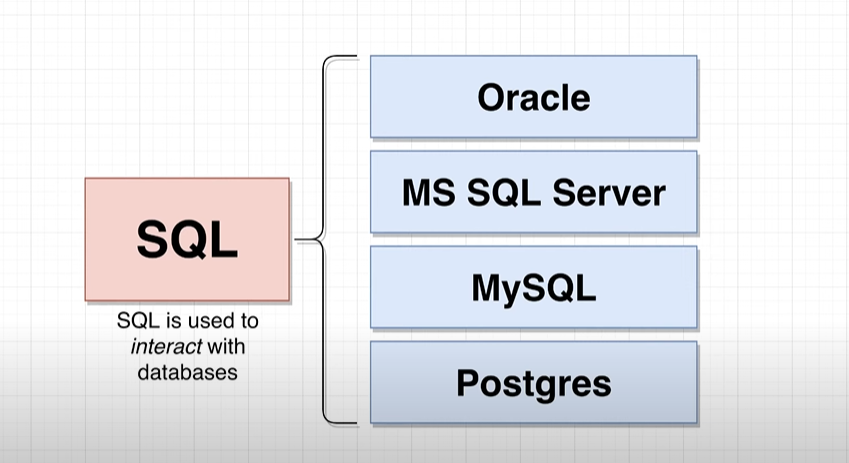
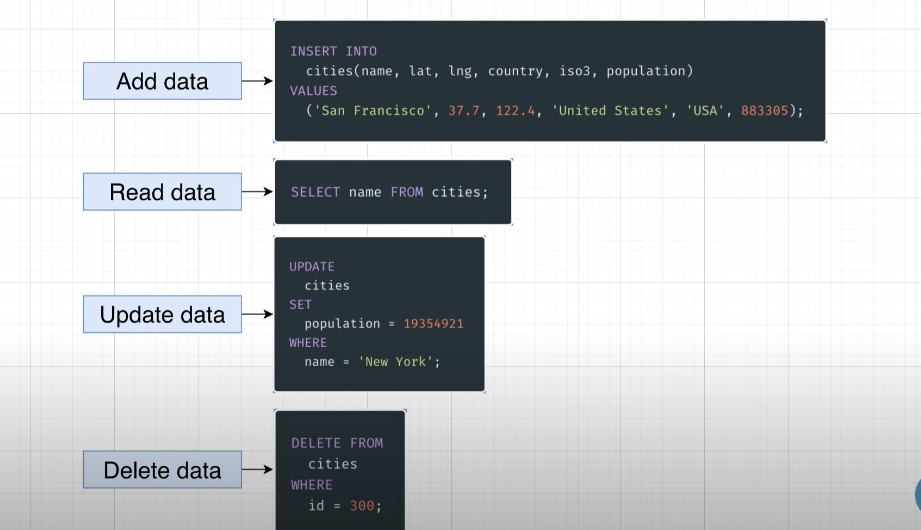
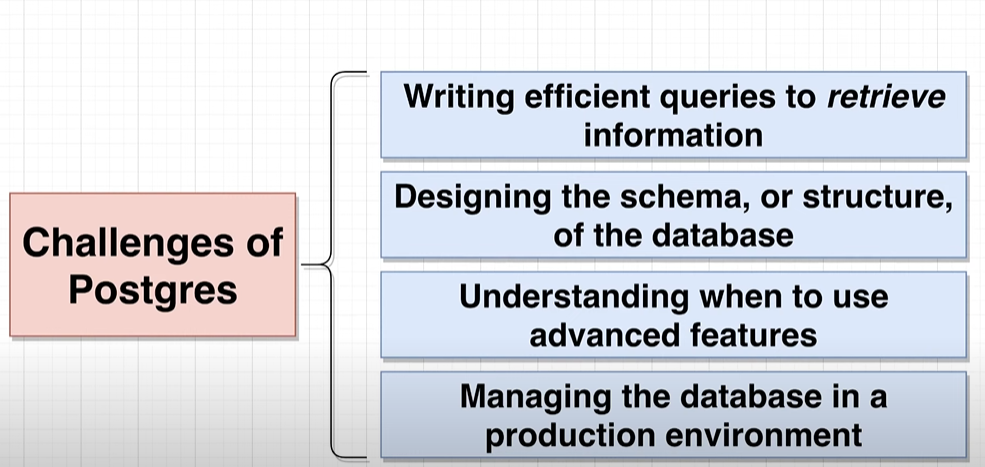
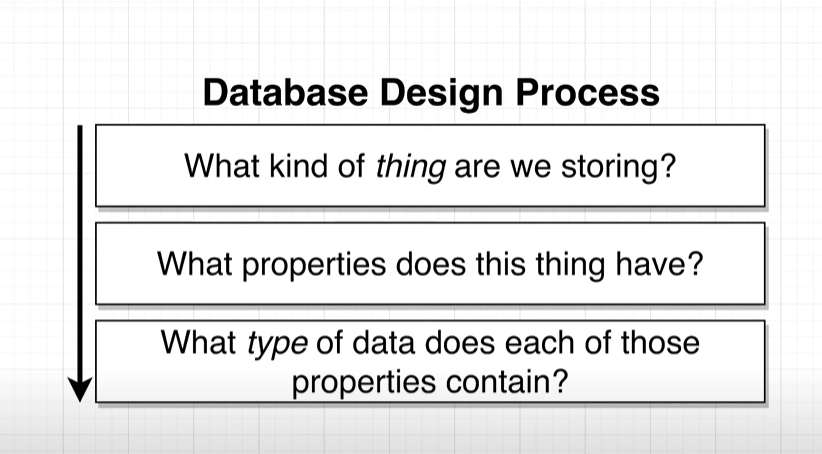
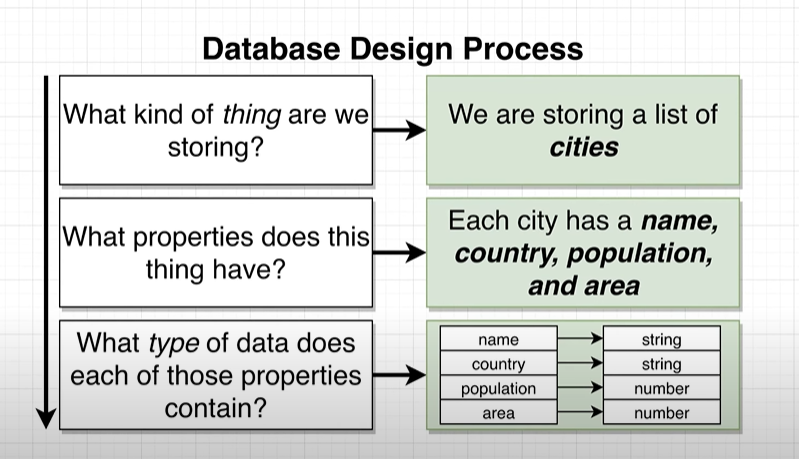
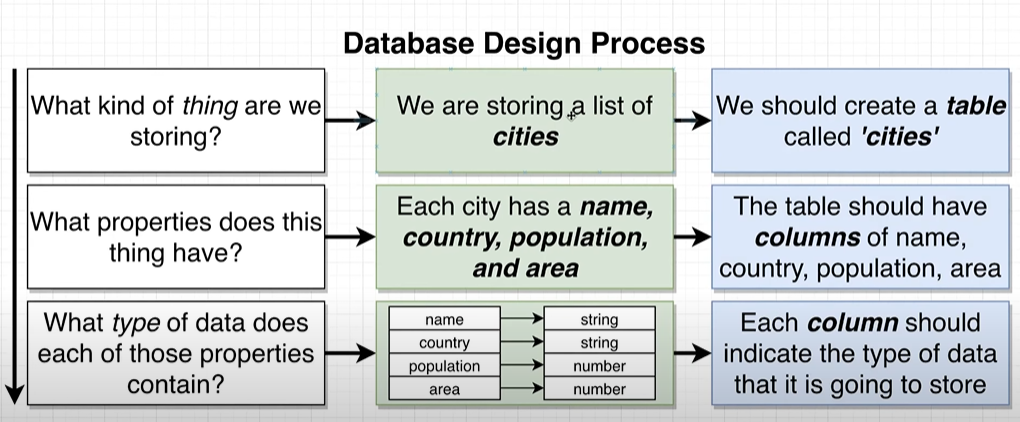
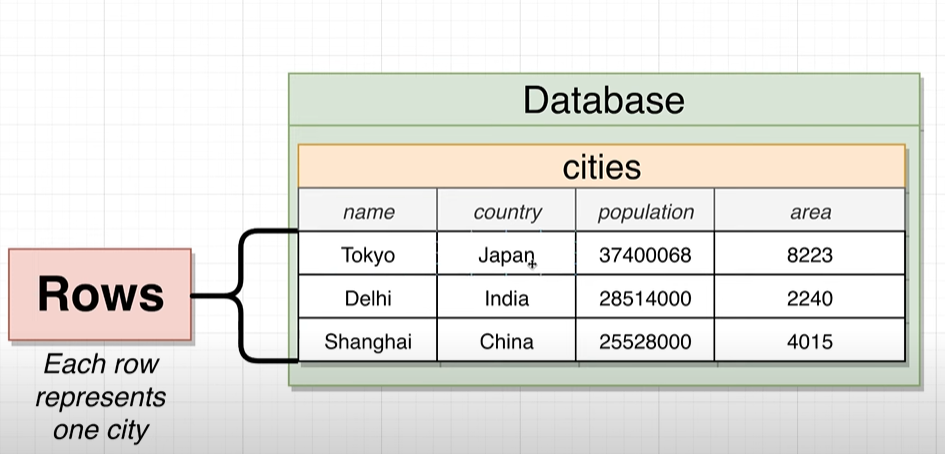
**Basic Postgres SQL :** is a db we use to store the information and retrieve the information.

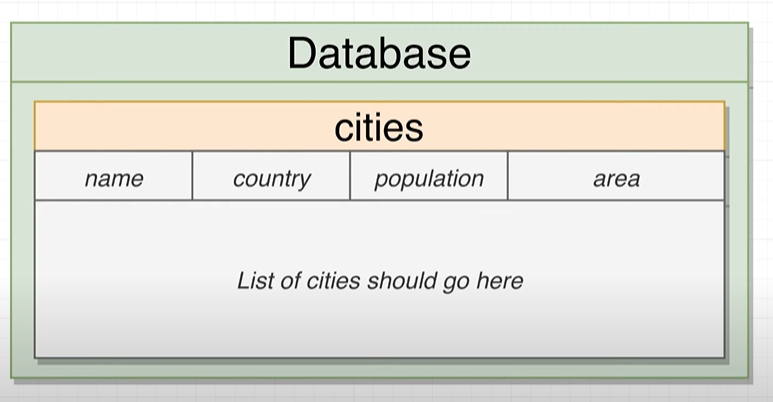
**# To work with postgres we connect to client,** client can we utility server or a program which access and store the data with some purpose.  **  
# we connect client to db using SQL(STRUCTURED QUERY LANG):  
  
  
Note :** we use SET in postgres to update the data into db.****  
# when u write the query think following point 1st for facing challenges:  
****  
Here we will focus on 2 challenges :

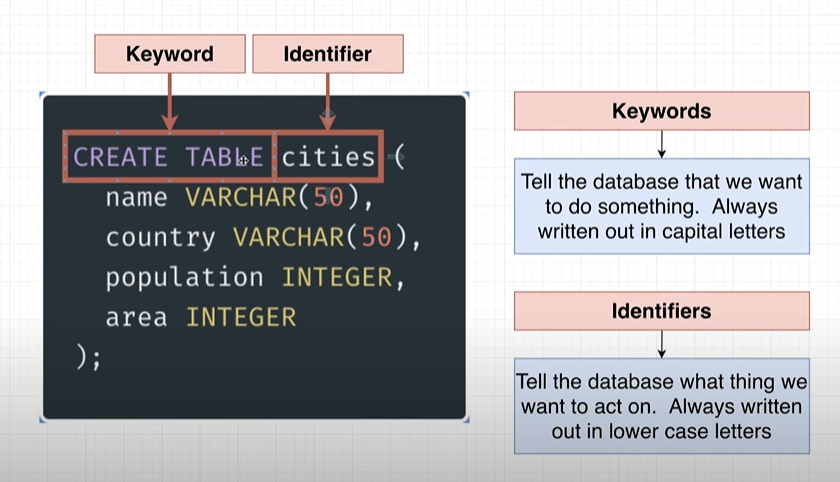
1. Writing efficient query.
2. Designing the structure of the db.

**# Before designing the db we follow the following process:  
  
  
assume here we are taking example of populated city in the world:  
  
Than we should think in this way:  
  
than we create a table like following:  
  
Now, we will write the query to create a table:**   
CREATE TABLE cities(  
name VARCHAR(50),

Country VARCHAR(50),

Population INTEGER,

Area INTEGER  
); **After running this query following structre will going to create in ur db:  
**

**Note:  
  
Now will insert the value into the table:**INSERT INTO cities (name,Country,Population,Area)

VALUES ('begaluru','india',45783783,6876);

Inset multiple rows in single column:

INSERT INTO cities (name,Country,Population,Area)

VALUES ('begaluru','india',45783783,6876),

('tokyo','japan' ,773783,676),

('shanghai','china',6783783,66876),

('Soa\_Paul','Brazil',5483783,23876);

To directly fetch the data from the table we can use :  
 SELECT \* FROM cities;

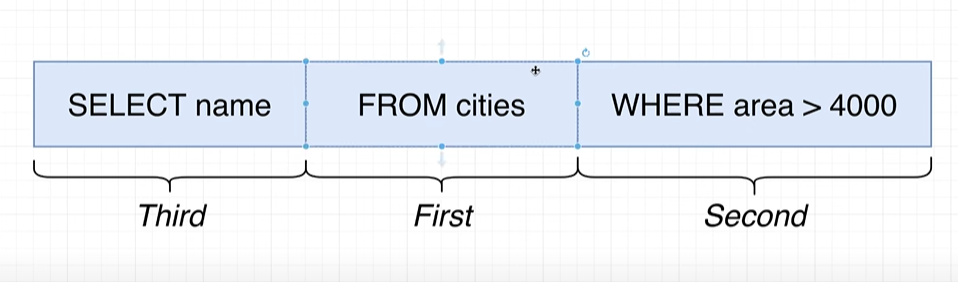
If u want specific column than :

SELECT name , Country FROM cities;

# Except pulling data we can perform operation and transform the data before we retrieve, for example I want the density of the city per square km so:

Density= population/area

SELECT name , Population/Area AS Density FROM cities;

# Same like that we can filter the query result :  
 SELECT name , Country FROM cities where area>4000;  
 it execute in following way:  
  
Some operator we use with where:

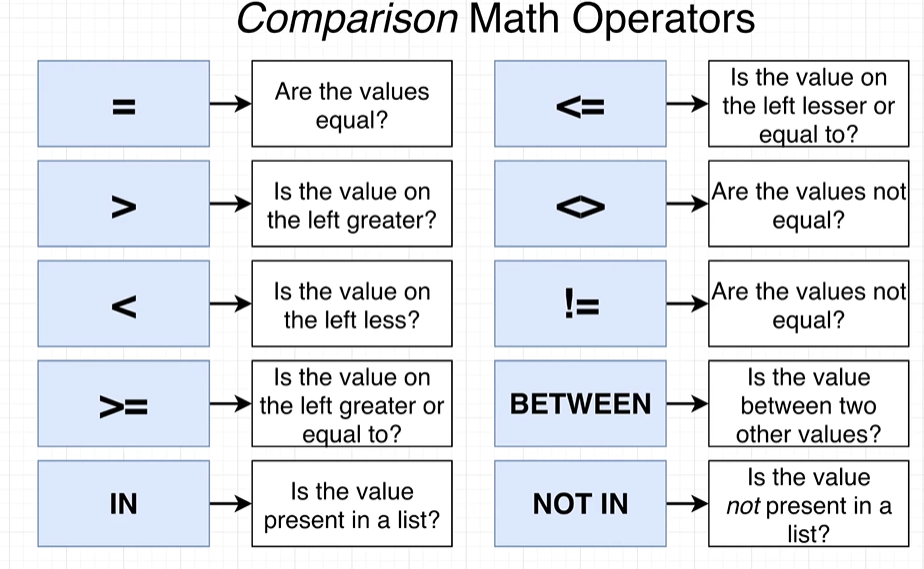
SELECT name , Country FROM cities where area>700;

SELECT name , Country FROM cities where area BETWEEN 100 and 4000;

SELECT name , Country FROM cities where area IN (600,66876,6876);

SELECT name , Country FROM cities where area NOT IN (600,66876,6876);

Note: In between oprerator it included the value as well to check.



Note: If u try with following way it will give u a error:  
 SELECT name , Population/Area AS density FROM cities WHERE density>6000; because

density is not a column in the table.

SELECT name , Population/Area AS density FROM cities WHERE Population/Area >6000;

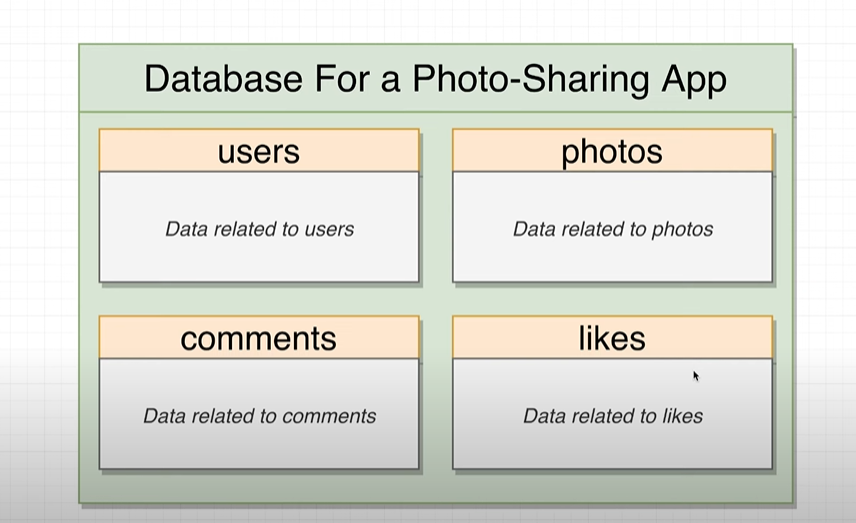
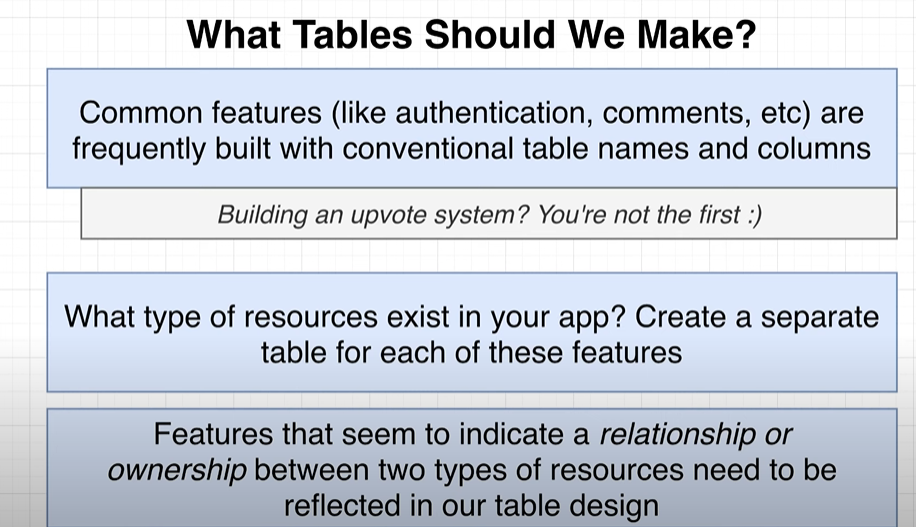
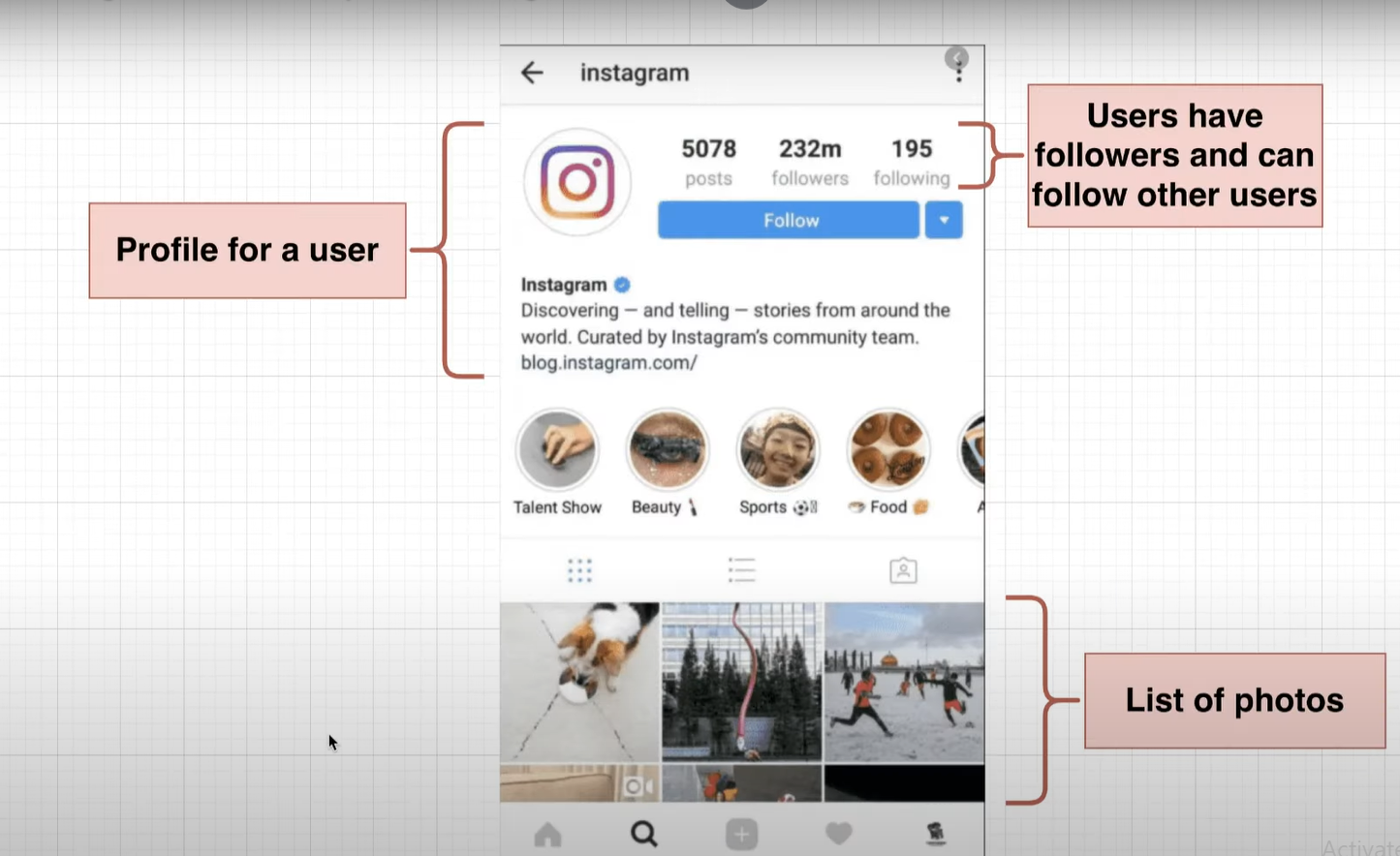
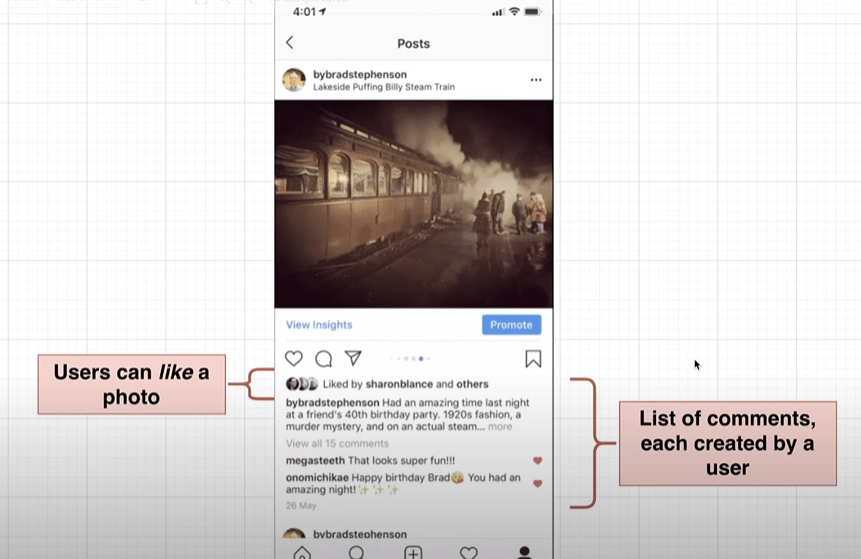
Is a correct way.

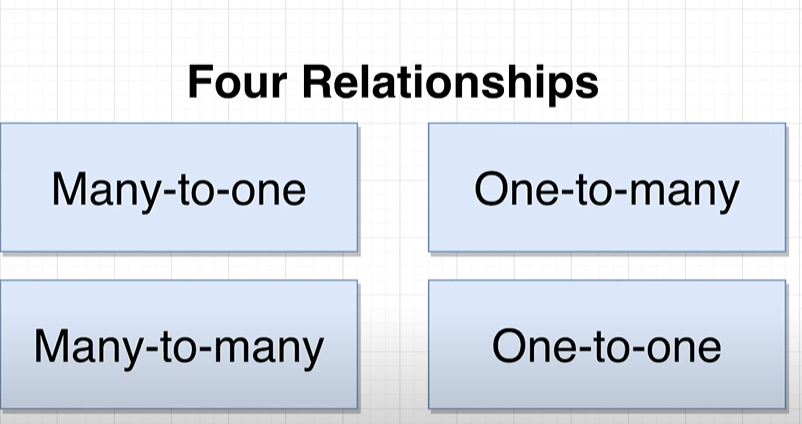
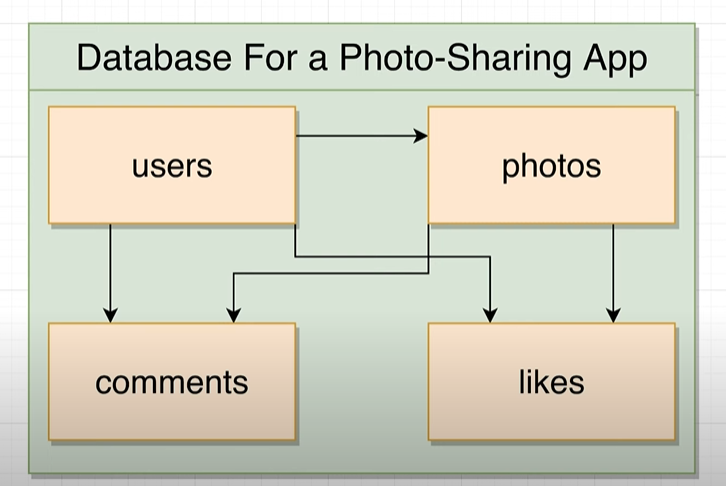
#**Now we will use the update to update the existing data into the table:  
 UPDATE cities**

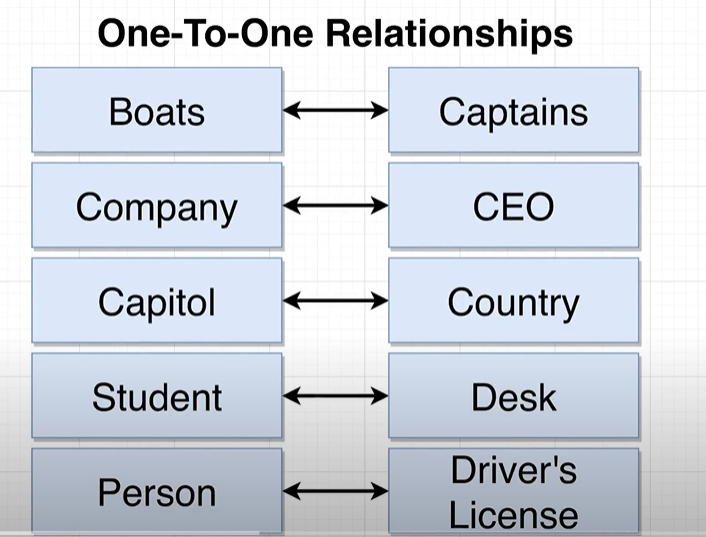
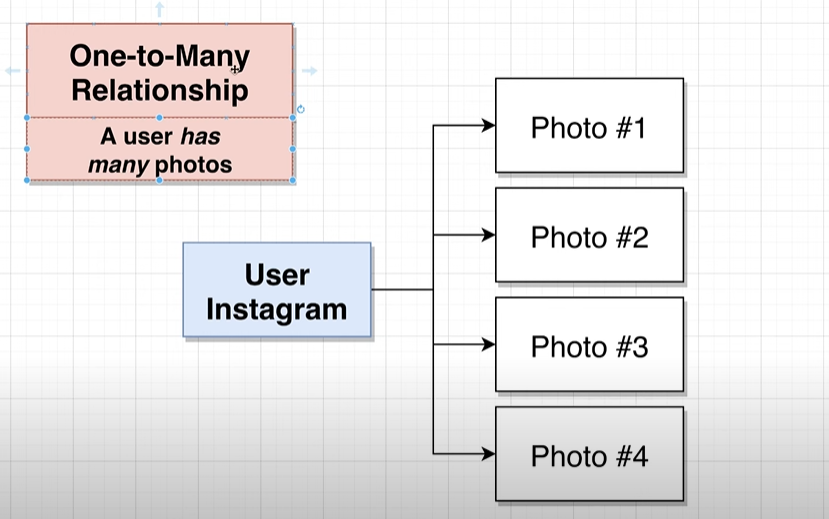
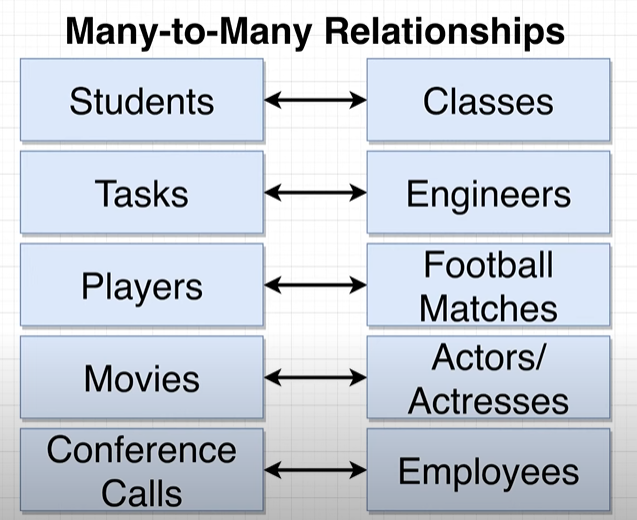
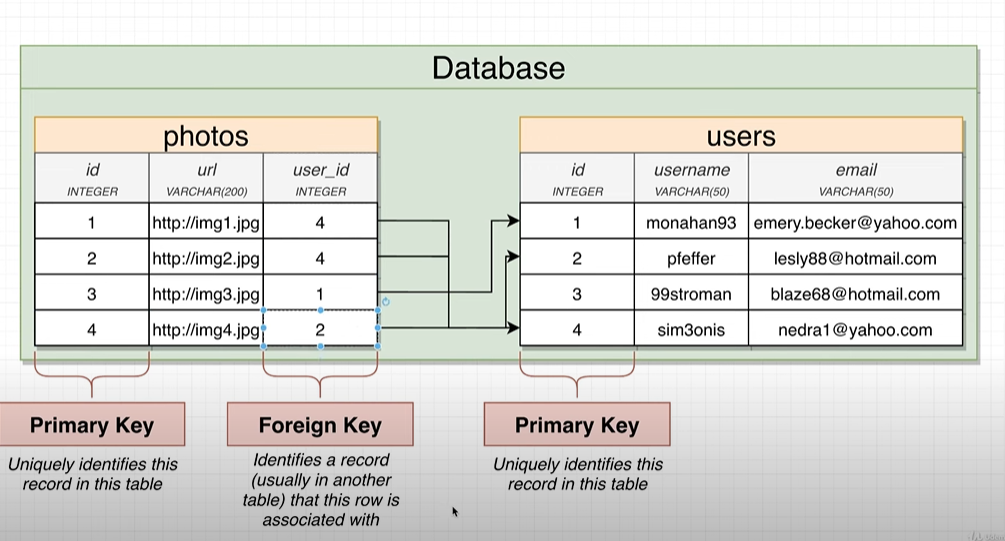
**SET Population = 39550500**

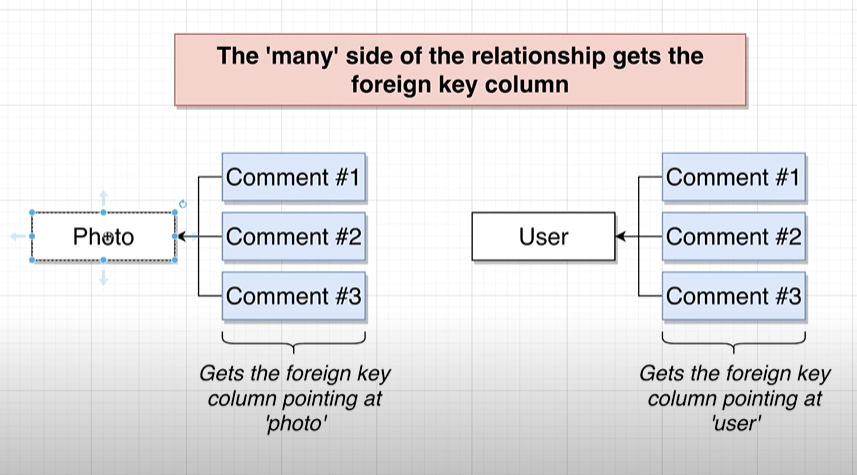
**WHERE name = 'Tokyo';**

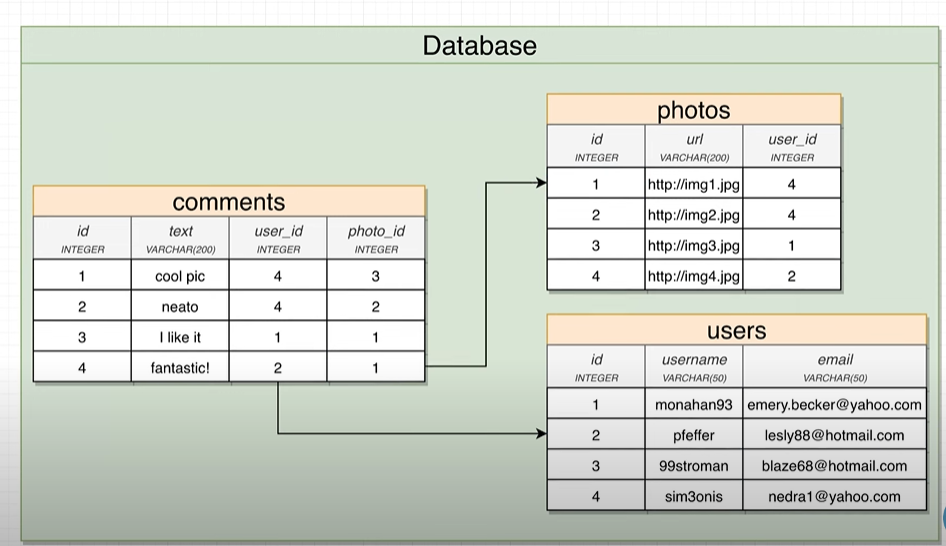
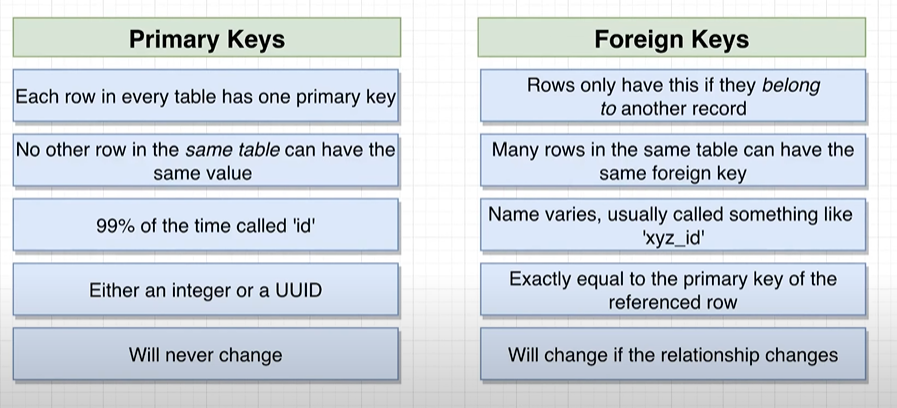
**Delete keyword:  
 DELETE FROM** cities **WHERE** name **=**’tokyo’**;**

IN Reality we have many table in one db like following :  
  
  
Lets take a example of insta where we have many sections, this is the user profile page :  
  
Here in common page we have some post on it:  
  
**From there we can understand there is a relationship between users and there comments:  
In RDBM we have 4 type of relationship:  
 1. 1:1 (one to one)  
 2. 1:M(one to many )**

**3.M:M (many to many).  
4.M:1 (many to one)  
** **One to one relationship(1:1):**

**  
  
One to many relationship 1:M:🡪 has one**one insta user can post mant photos.  
**Many to one (M:1): 🡪 Has Many**one photo has many comment.  
  
 Note: One to many and many to one both are vise versa only perspective we have to change.  
 1:M <-> M:1  
  
# Many to many relationship M:M:  
  
Note: To set the relationship between table we have following keys:  
1.Primary keys(Unique row and not null)  
2. Foreign key (foreign key can be null)  
we relate the relationship between foreignkey and primany key of another table.  


# many side of the table get the foreign key table :  
  
Example:

  
Note: difference between foreign and primary key:  


Let’s write the query for user table:

CREATE TABLE users(

Id SERIAL PRIMARY KEY NOT NULL,

Username VARCHAR(50)

);  
Note: SERIAL will automatically increase a id column value by 1, it’s Postgres functionality.  
  
Now, Let’s insert some value into the table:  
Note :here we are not passing the id because SERIAL will automatically handle.

INSERT INTO users (username)

VALUES('Ravi'),('james'),('abdul'),('sidhu');

# Lets create a photos table :

CREATE TABLE photos(

Id SERIAL PRIMARY KEY,

url VARCHAR(200),

user\_id INTEGER REFERENCES users (id)

);

Than insert some data into photos table:

INSERT INTO photos (url , user\_id)

VALUES('hhtkjhsjhjkj',4),

('hhtkjhsjhjkj',4),

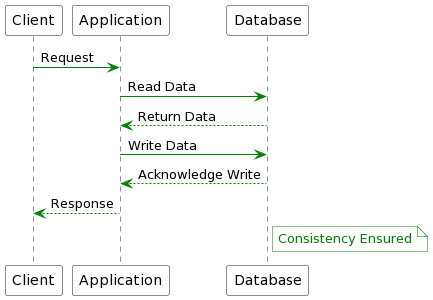
('hhtkjhsjhjkj',1),

# ('hhtkjhsjhjkj',2); \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Note: Consistency in DBMS:-

Data integrity and reliability are key in the domain of Database Management Systems (DBMS). Consistency, one of the core principles in DBMS, ensures that every transaction is made according to predefined rules and limits thus preserving the accuracy and authenticity of data kept within. The change to the database must take it from one consistent state into another.

Consistency, in DBMS, requires that any modification to a single piece of data be reflected uniformly across all linked tables as well as entities. For example, suppose you have a driver’s license database. Updating a driver’s house address should consistently appear in all relevant tables just to avoid mismatching data.

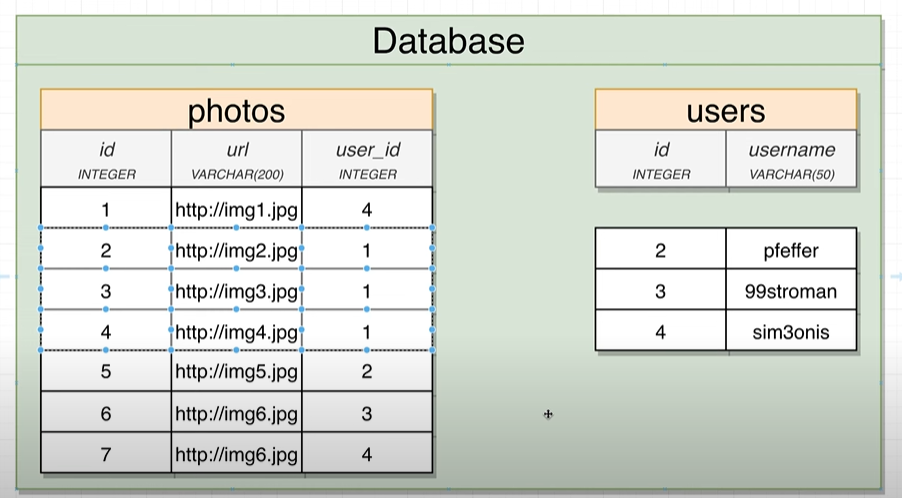
It is not enough for consistency in DBMS since it may not result in transactional correctness at all times but it plays an important role in shielding against programming errors that violate set up database constraints. To make sure that data remains reliable and intact, RDBMS go through the process of enforcing consistency to create a firm foundation for robust and trusted applications based on stored data.



*Consistency in DBMS*

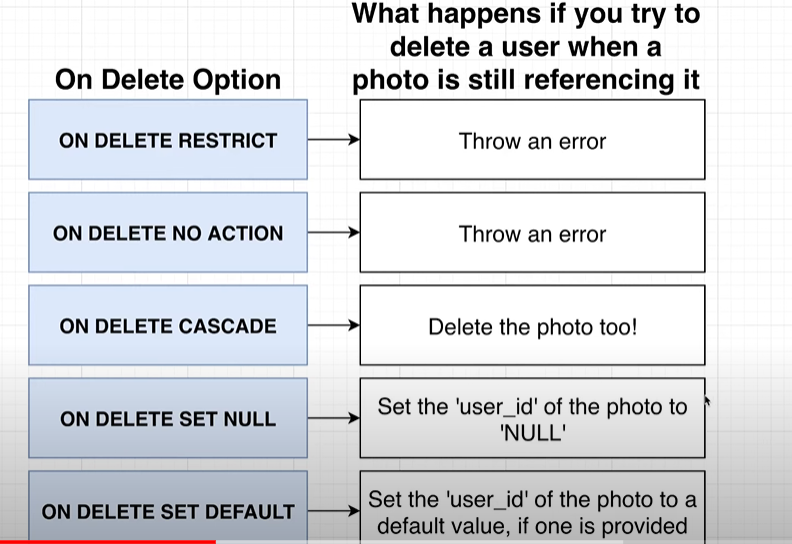
**Why is Data Consistency in DBMS Important?**

In a database, maintaining accuracy and usability are vital to ensuring consistency. Mismatching data can destabilize or corrupt systems, thus undermining the integrity of the database. For that reason, all users must input data in a way that is consistent with current database records; this is normally accomplished by isolating data fields to avoid conflicting transactions.



But now problem here is if user got delete (or user has deleted his profile picture from inta than what about the uploaded picture and all information) in this case remaining garbage data will increase so we follow the following solution:

For that we can give the ON DELETE CONSTAINS which should restrict while deletion:

  
the above picture is referring the photos and user table.   
note :-

 **DROP TABLE**: Cannot be rolled back if not executed within a transaction that is subsequently rolled back.

 **DELETE**: Can be rolled back if executed within a transaction.

For setting all the constrain let me drop the table and re-create all the tables again:

DROP TABLE photos; 🡪 This will drop entire table with structure.

Now let me set ON DELETE CASCADE: what is do basically whenever you delete the user it will delete the REFERENCES photos row from photos table too like in below example user 1 has deleted from user table so all the data got deleted from photos table which is referring to users id 1.

CREATE TABLE photos(

Id SERIAL PRIMARY KEY,

url VARCHAR(200),

user\_id INTEGER REFERENCES users (id) ON DELETE CASCADE

);

Now inserting some data into the table :

INSERT INTO photos (url , user\_id)

VALUES('hhtkjhsjhjkj',1),

('hhtkjhsjhjkj',4),

('hhtkjewerehsjhjkj',1),

('sdfd',2),

('hgfddf',3),

('sdfeeee',2);  
  
now deleted the user 1 from user table :

DELETE FROM users

WHERE id =1;

Than,  
SELECT \* FROM photos;

SELECT \* FROM users;

