**Learn PostgreSQL Tutorial - Full Course for Beginners**

YouTube : https://www.youtube.com/watch?v=qw--VYLpxG4

GitHub: https://github.com/amigoscode/sql-postgres-course

**What is Database**

- Store

- Manipulate

- Retrieve

**What is SQL And Relational Database**

- Postgres is the Database

- SQL =**S**tructured **Q**uery **L**anguage

- SQL is a programming language

* e.g. **SELECT** first\_name **FROM** person
* Manage data held in a **relational** database
* Easy to learn
* Very powerful
* 1974
* Used all over the internet

- How data is stored

* Stores data in tables
* Columns
* Rows

- Relational database

* it simply a relation between one ore more tables and data might be structured inside of a relational database.

**What is PostgreSQL**

- PostgreSQL : The World's Most Advance Open Source Relational DB with over 30 years of active development. It doesn't have to pay for a license. This is one of the advantages over your know, Oracle.

* Object-relational database management system
* Modern
* Open Source

**PostgreSQL Installation (Windows)**

- Download URL: https://www.postgresql.org/download/windows/ and download current version (14.2-2 x64 version )

- install PostgreSQL (password: Chan\*2022)

- Connecting the DB server

* GUI Client
* Terminal / CMD
* Application

**GUI Clients vs Terminal/CMD Clients**

- **DataGrip** ([DataGrip: The Cross-Platform IDE for Databases & SQL by JetBrains](https://www.jetbrains.com/datagrip/)) : Support lots of databases such as PostgreSQL, MySQL, SQL Server, Azure and SQLite, so on and so forth. But you need to buy a license.

- **Postico - A Modern PostgreSQL Client for the Mac (**https://eggerapps.at/postico/**)** and it's free

- **pgAdmin** for window users (https://www.pgadmin.org/)

**Setup PSQL (Windows) (**0:25:22**)**

*- Maybe to add "****C:\Program Files\PostgreSQL\14\bin"*** *to the SYSTEM Path*

- Introduce two tools - pgAdmin 4 and psql

**How to Create Database**

- Statement :

**CREATE DATABASE** database \_name;

- Example as below :-

CREATE DATABASE test; <- need to use ';' to execute the

\l to list the databases

**How to Connect to Database**

There are two methods to connect the DB

1. **psql -h localhost -p 5432 -U postgres -d test** or

2. Under psql command, then type **\c test**

- \l to list the databases

Note :\l -> list all databases

**A Very Dangerous Command**

- Statement :

**DROP DATABASE** database\_name;

- Example as below :-

**DROP DATABASE test;** <- need to use ';' to execute the

\l to list the databases

**How to Create Tables**

- Statement :

**CREATE TABLE** table\_name (

Column name + data type + constraints if any

);

- Example as below :-

**CREATE TABLE** person (

**id** int,

**first\_name** VARCHAR(50),

**last\_name** VARCHAR(50),

**gender** VARCHAR(7),

**date\_of\_birth** TIMESTAMP

)

**Creating Tables Without Constraints**

- Example as below :-

**CREATE TABLE** person (

**id** int,

**first\_name** VARCHAR(50),

**last\_name** VARCHAR(50),

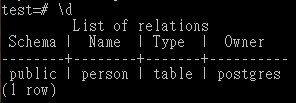
**gender** VARCHAR(7),

**date\_of\_birth** DATE

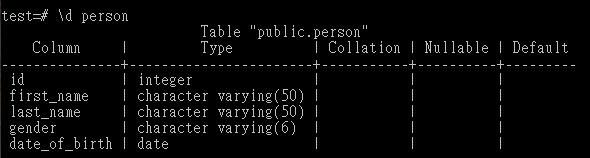
)

Note: use DATE instead of TIMESTAMP

- **\d** to list the tables

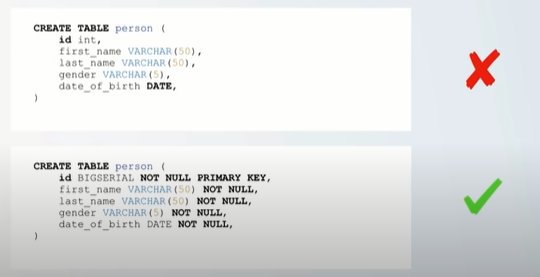


- **\d table\_name** to list the fields



**Creating Tables Without Constraints**

- Example as below :-

****

NEW SQL

**CREATE TABLE** person (

**id** BIGSERIAL **NOT NULL PRIMARY KEY**,

**first\_name** VARCHAR(50) **NOT NULL**,

**last\_name** VARCHAR(50) **NOT NULL**,

**gender** VARCHAR(7) **NOT NULL**,

**date\_of\_birth** DATE **NOT NULL,**

**email** VARCHAR(150) );

**Insert Into**

- Example as below :-

**INSERT INTO** person (

first\_name,

last\_name,

gender,

date\_of\_birth)

**VALUES** ('Anne', 'Smith', 'FEMALE', DATE '1988-01-09');

**INSERT INTO** person (

first\_name,

last\_name,

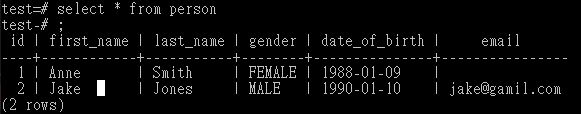
gender,

date\_of\_birth,

email)

**VALUES** ('Jake', 'Jones', 'MALE', DATE '1990-01-10','jake@gamil.com');

**SELECT** \* **FROM** person



**Generate 1000 Rows with Mockaroo (1:02:36)**

- Mockaroo to generate fake data : [https://www.mockaroo.com/](https://www.mockaroo.com/%20)

- Use the VS Code (<https://code.visualstudio.com/>) for working SQL files. For the server backend side, we can use IntelliJ

- We can use other way - atom (https://atom.io/) or you can use sublime.

- download the SQL and click "CREATE TABLE" option

- need to modify to below SQL command

create table person (

    first\_name VARCHAR(50) NOT NULL,

    last\_name VARCHAR(50) NOT NULL,

    email VARCHAR(150),

    gender VARCHAR(7) NOT NULL,

    date\_of\_birth DATE NOT NULL,

    country\_of\_birth VARCHAR(50)

);

- save the file and execute the file under PSQL command file. The command is as below:-

\i C:/Users/chan/Downloads/person.sql

- need to add id to the SQL command and the new version is as below:-

drop table person;

create table person (

    id BIGSERIAL NOT NULL PRIMARY KEY,

    first\_name VARCHAR(50) NOT NULL,

    last\_name VARCHAR(50) NOT NULL,

    email VARCHAR(150),

    gender VARCHAR(7) NOT NULL,

    date\_of\_birth DATE NOT NULL,

    country\_of\_birth VARCHAR(50)

);

- then execute it again.

**Select From**

- test=# **select from person;**

Output

--

(1000 rows)

**Order By**

- SELECT \* FROM person ORDER BY country\_of\_birth ;

- SELECT \* FROM person ORDER BY country\_of\_birth DESC;

- SELECT \* FROM person ORDER BY date\_of\_birth, country\_of\_birth;

**Distinct**

- SELECT DISTINCT country\_of\_birth FROM person;

- SELECT DISTINCT country\_of\_birth FROM person ORDER BY country\_of\_birth;

**Where Clause and AND**

- SELECT \* FROM person WHERE gender='Female';

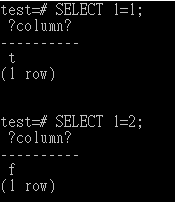
- SELECT \* FROM person WHERE gender='Male' AND country\_of\_birth='Poland';

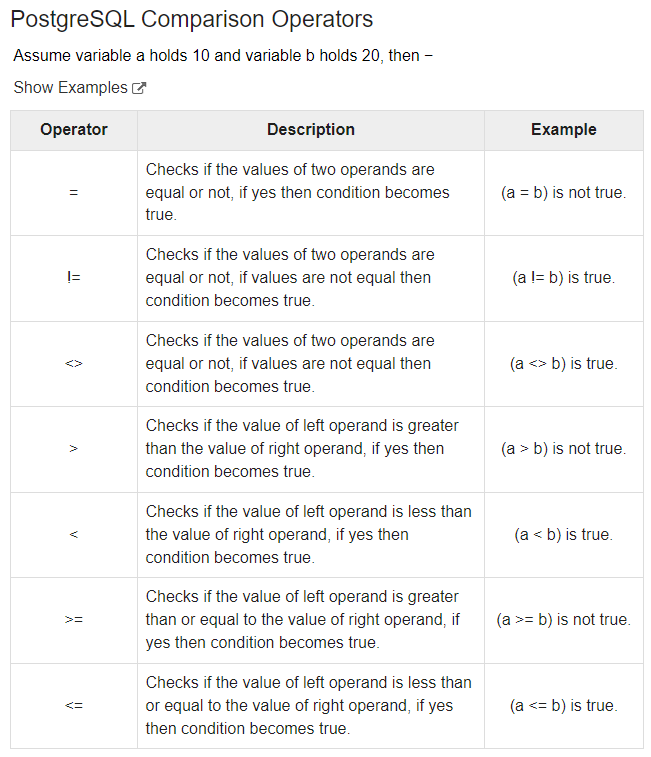
- SELECT \* FROM person WHERE gender='Male' AND (country\_of\_birth='Poland' OR country\_of\_birth='China');

- SELECT \* FROM person WHERE gender='Male' AND (country\_of\_birth='Poland' OR country\_of\_birth='China') AND last\_name='Gleave';

- SELECT \* FROM person WHERE gender='Female' AND (country\_of\_birth='Poland' OR country\_of\_birth='China') AND last\_name='Pietersmo';

**Comparison Operators**





**Limit, Offset & Fetch**

- SELECT \* FROM person LIMIT 10;



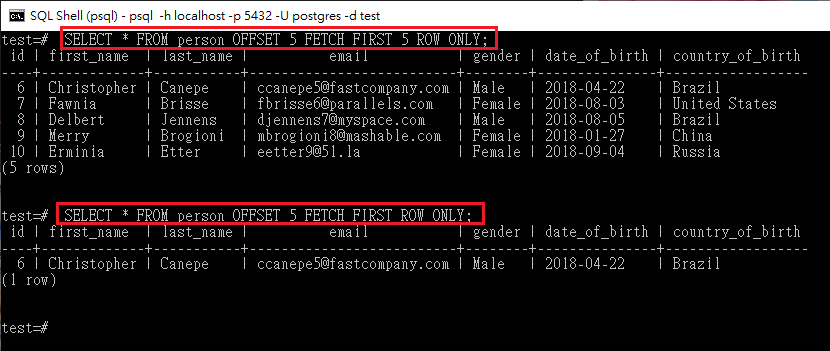
- SELECT \* FROM person OFFSET 5 LIMIT 5;



- SELECT \* FROM person OFFSET 5

- SELECT \* FROM person OFFSET 5 FETCH FIRST 5 ROW ONLY;

- SELECT \* FROM person OFFSET 5 FETCH FIRST ROW ONLY;



- Note :it's similar to LIMIT but it's in order.

**IN**

- SELECT \* FROM person WHERE (country\_of\_birth in ('China', 'France', 'Brazil'));

- SELECT \* FROM person

WHERE (country\_of\_birth in ('China', 'France', 'Brazil', 'Mexico','Portugal','Nigeria')) ;

- SELECT \* FROM person

WHERE (country\_of\_birth in ('China', 'France', 'Brazil', 'Mexico','Portugal','Nigeria'))

ORDER BY country\_of\_birth ;

**Between**

- SELECT \* FROM person

WHERE date\_of\_birth BETWEEN DATE '2017-01-01' AND '2018-01-01';

**Like and iLike**

- SELECT \* FROM person WHERE email LIKE '**%**.com';

- SELECT \* FROM person WHERE email LIKE '**%**bloomberg.com';

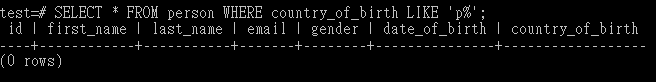
- SELECT \* FROM person WHERE email LIKE '**%**google.**%**';

- SELECT \* FROM person WHERE email LIKE '**\_\_\_\_\_\_\_\_**@**%**';

- SELECT \* FROM person WHERE email LIKE '\_\_\_\_\_\_7@%';



- SELECT \* FROM person WHERE country\_of\_birth LIKE 'p%';



- SELECT \* FROM person WHERE country\_of\_birth LIKE 'P%';

- SELECT \* FROM person WHERE country\_of\_birth iLIKE 'p%';



**Wild cards**

There are only two wildcards that can be used together with

- Percent sign (%) : to represent zero, one, or many characters or numbers.

- Underscore (\_) : to represent one character or number.

The PostgreSQL **ILIKE** operator is used query data using pattern matching techniques. Its result include strings that are **case-insensitive** and follow the mentioned pattern.

**Group By**

- SELECT country\_of\_birth, COUNT(\*) FROM person GROUP BY country\_of\_birth;

- SELECT country\_of\_birth, COUNT(\*) FROM person GROUP BY country\_of\_birth ORDER BY country\_of\_birth;

**Group By Having**

- SELECT country\_of\_birth, COUNT(\*) FROM person GROUP BY country\_of\_birth HAVING COUNT(\*) >=40 ORDER BY country\_of\_birth;



- SELECT country\_of\_birth, COUNT(\*) FROM person GROUP BY country\_of\_birth HAVING COUNT(\*) <=40 ORDER BY country\_of\_birth;



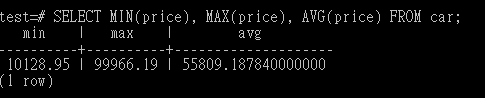
Note : [PostgreSQL: Documentation: 9.5: Aggregate Functions](https://www.postgresql.org/docs/9.5/functions-aggregate.html)

**Adding New Table And Dat Using Mockaroo**

- \i C:/Users/chan/Downloads/car.sql;

**Calculating Min, Max & Average**

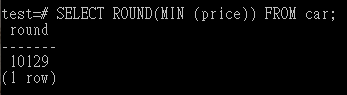
- SELECT MIN(price), MAX(price), AVG(price) FROM car;



- SELECT ROUND(AVG(price)) FROM car;

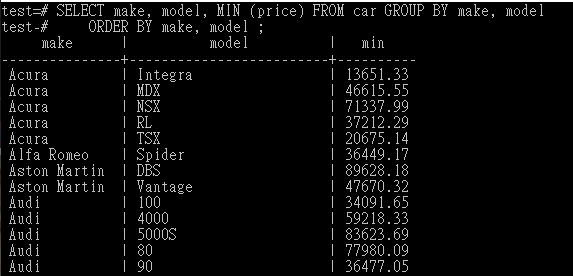


- SELECT ROUND(MIN (price)) FROM car;



- SELECT make, model, MIN (price) FROM car GROUP BY make, model

ORDER BY make, model ;

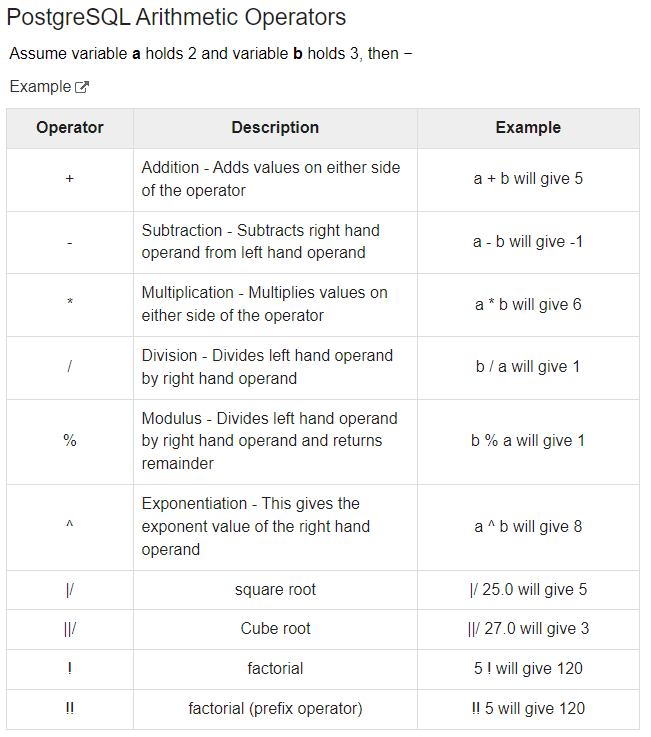


**Sum**

- SELECT SUM(price) FROM car;

- SELECT make, SUM(price) FROM car GROUP BY make;

**Basics of Arithmetic Operators**



**Arithmetic Operators - Round**

- SELECT id, make, model, price, price \* 0.1 FROM car;

- SELECT id, make, model, price, ROUND(price \* 0.1,2) FROM car;

- SELECT id, make, model, price, ROUND(price \* 0.1,2),

price - ROUND(price \* 0.1,2) FROM car;

**Alias**

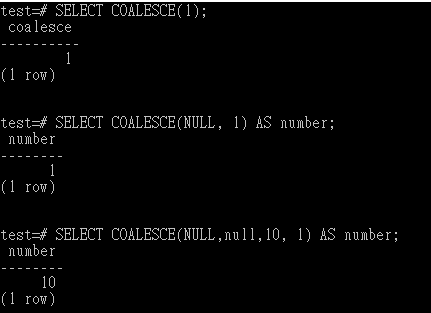
- SELECT id, make, model, price AS original\_price,

ROUND(price \* 0.1,2) AS ten\_percent\_value,

price - ROUND(price \* 0.1,2) AS promotion\_price FROM car;

**Coalesce**

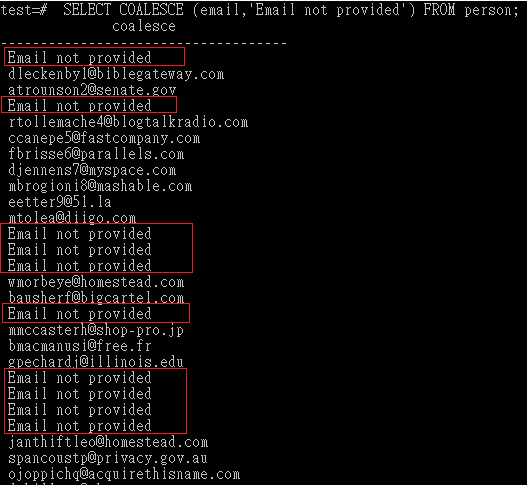
- The **COALESCE** function accepts an unlimited number of arguments. It returns the first argument that is not null. If all arguments are null, the **COALESCE** function will return null.



- SELECT email FROM person;

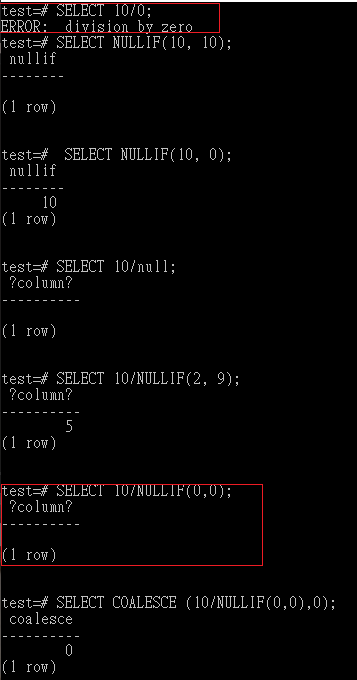
- SELECT COALESCE (email) FROM person;

- SELECT COALESCE (email,'Email not provided') FROM person;



**NULLIF**

- Use NULLIF to prevent division-by-zero error



- SELECT id, make, model, price AS original\_price,

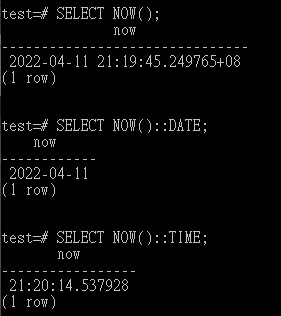
**Timestamps and Dates**

- SELECT NOW();

- SELECT NOW()::DATE;

- SELECT NOW()::TIME;

[8.5. Date/Time Types](https://www.postgresql.org/docs/14/datatype-datetime.html)



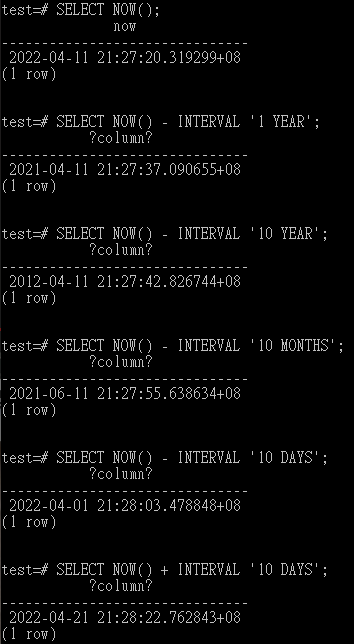
**Adding and Subtracting With Dates**

- SELECT NOW() - INTERVAL '1 YEAR';

- SELECT NOW() - INTERVAL '10 MONTHS';

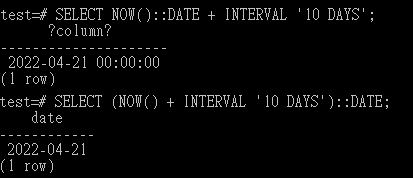
- SELECT NOW() - INTERVAL '10 DAYS;

- SELECT NOW() + INTERVAL '10 DAYS;



- SELECT NOW()::DATE + INTERVAL '10 DAYS;

- SELECT (NOW() + INTERVAL '10 DAYS)::DATE;



**Extracting Fields From Timestamp**

- SELECT NOW();

- SELECT EXTRACT(YEAR FROMNOW());

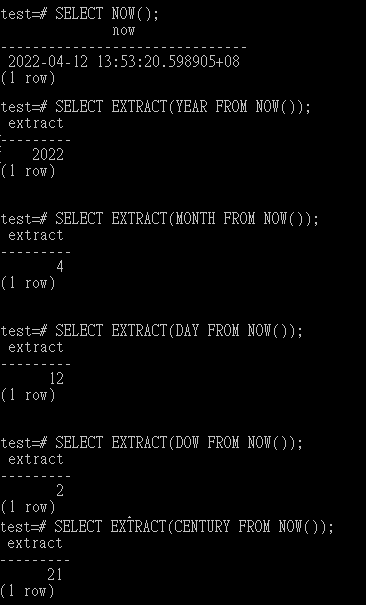
- SELECT EXTRACT(MONTH FROMNOW());

- SELECT EXTRACT(DAY FROMNOW());

- SELECT EXTRACT(DOW FROMNOW());

Note :DOW means Day of week

- SELECT EXTRACT(CENTURY FROMNOW());



**Age Function**

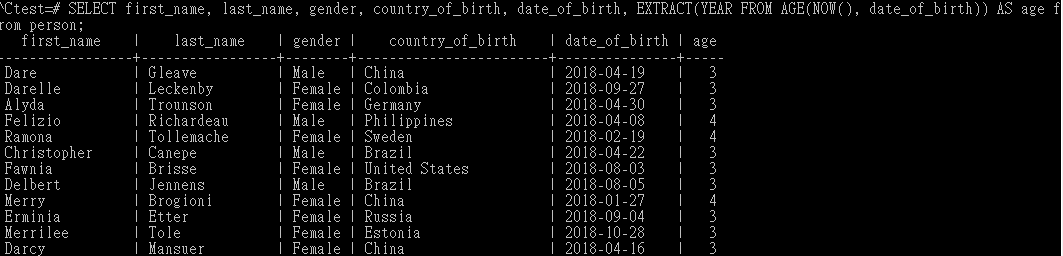
- SELECT first\_name, last\_name, gender, country\_of\_birth, date\_of\_birth,

AGE(NOW(), date\_of\_birth) AS age from person;

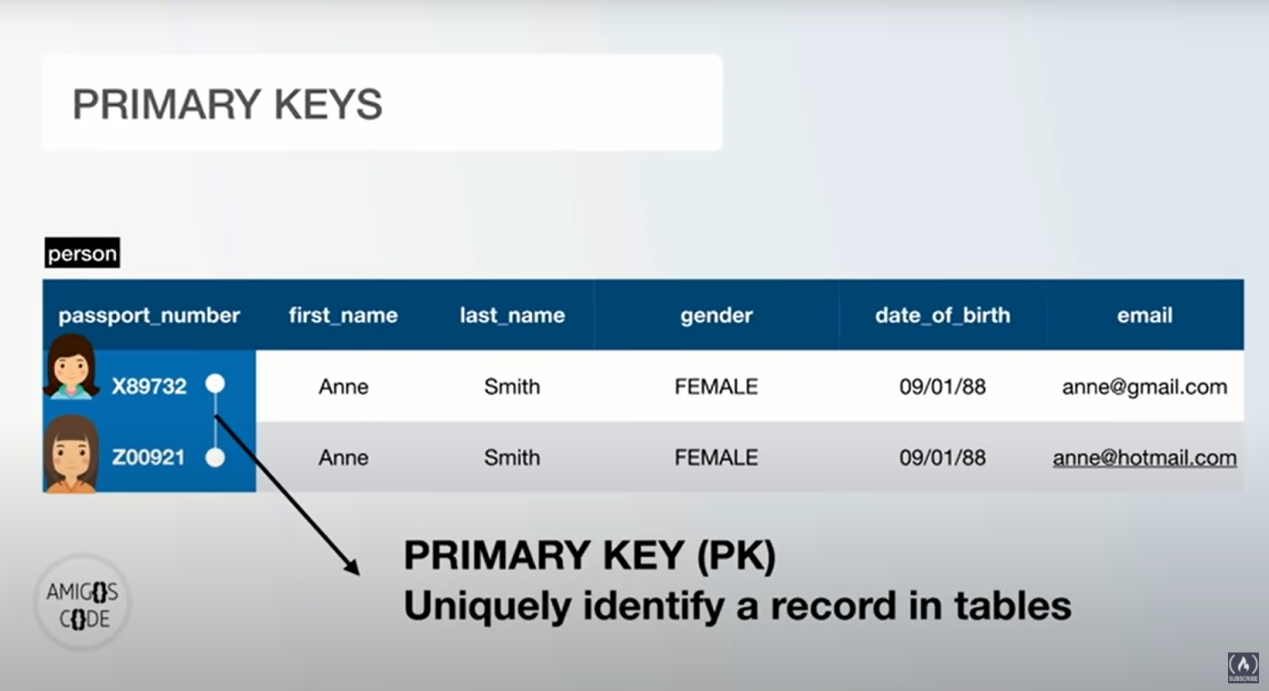


- SELECT first\_name, last\_name, gender, country\_of\_birth, date\_of\_birth,

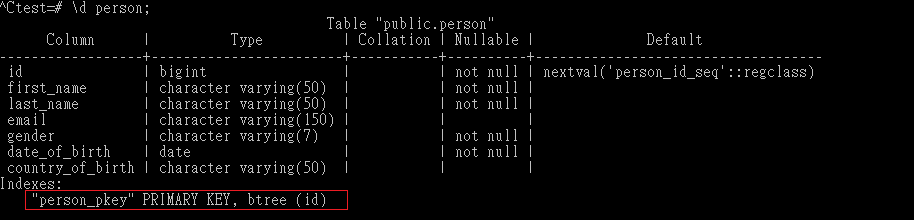
EXTRACT(YEAR FROM AGE(NOW(), date\_of\_birth)) AS age from person;



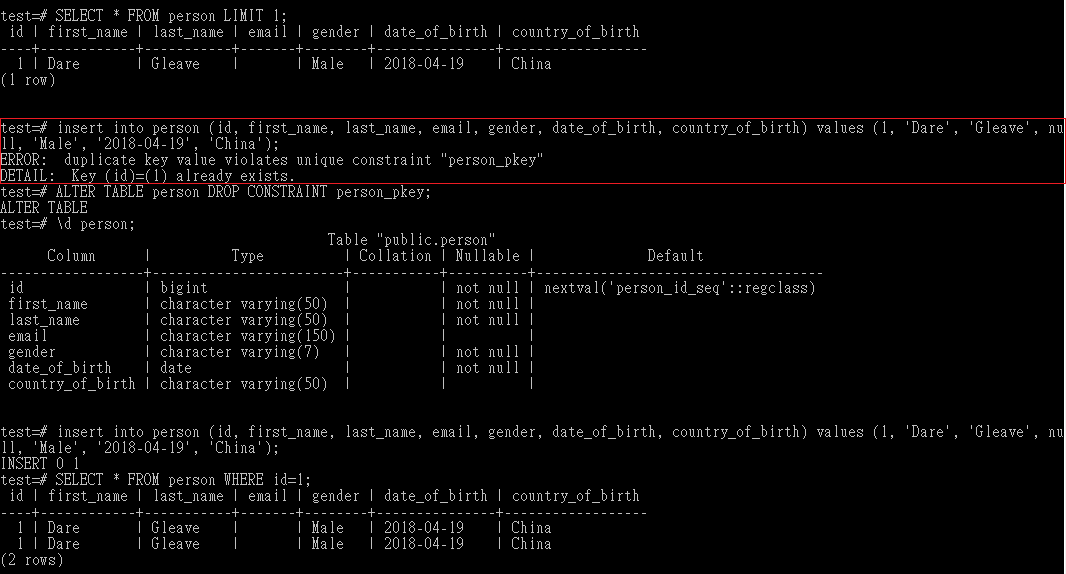
**What are Primary Keys**



**Understanding Primary Keys**

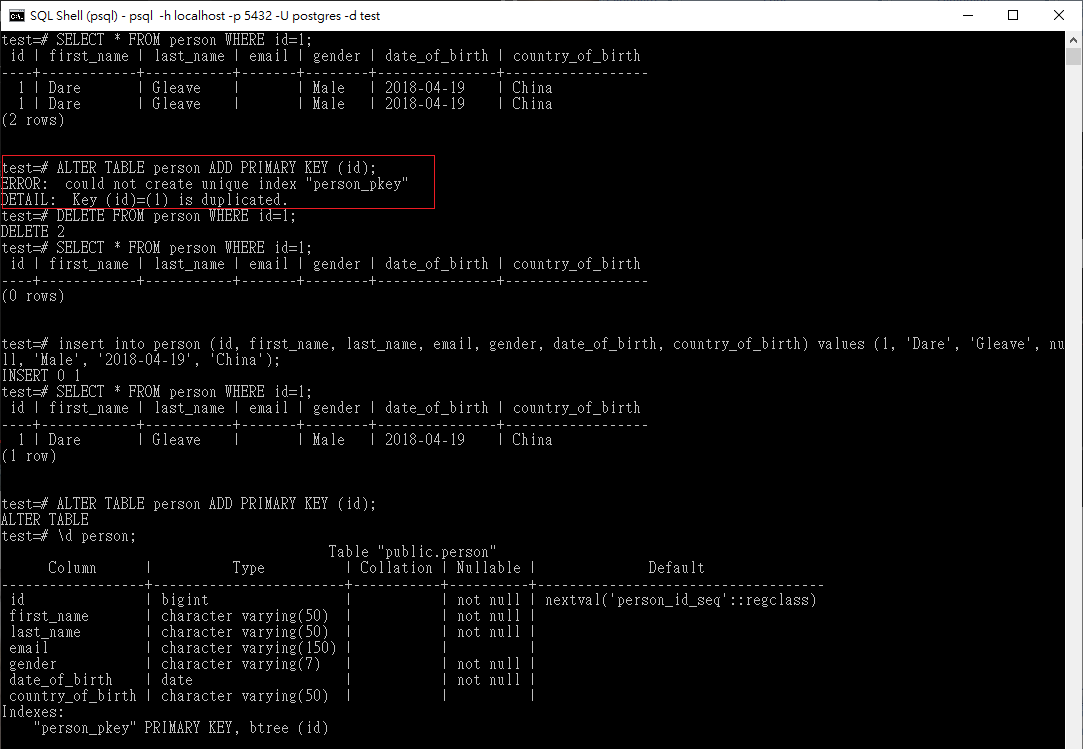


- ALTER TABLE person DROP CONSTRAINT person\_pkey;



**Adding Primary Key**

- ALTER TABLE person ADD PRIMARY KEY (id);

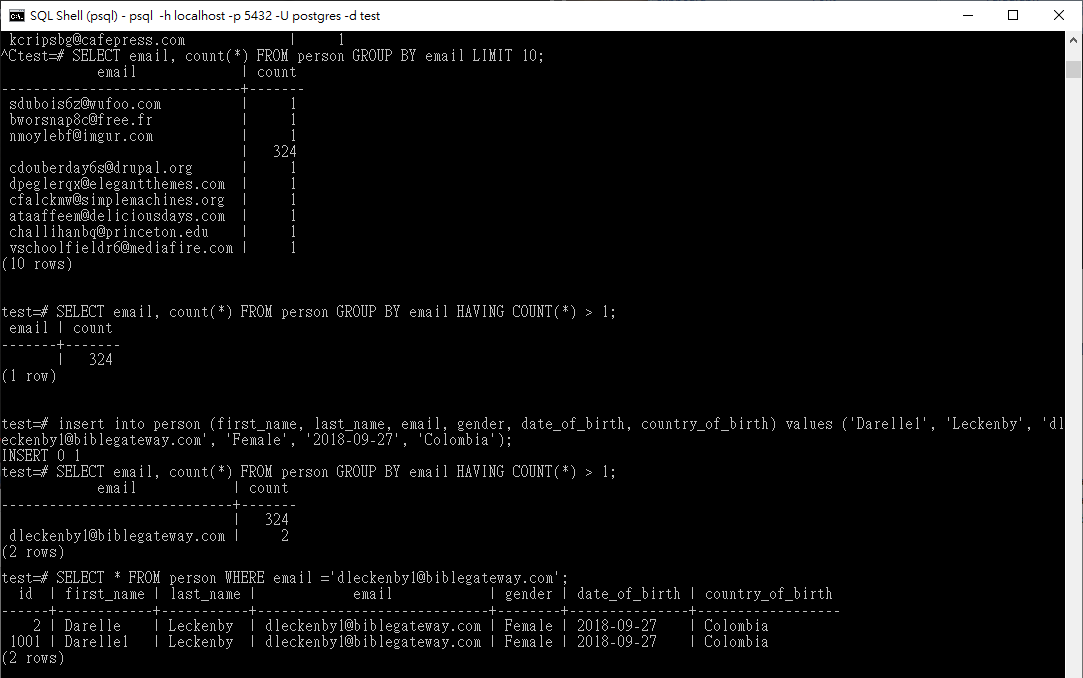


**Unique Constraints**

- SELECT email, COUNT(\*) FROM person GROUP BY email LIMIT 10;

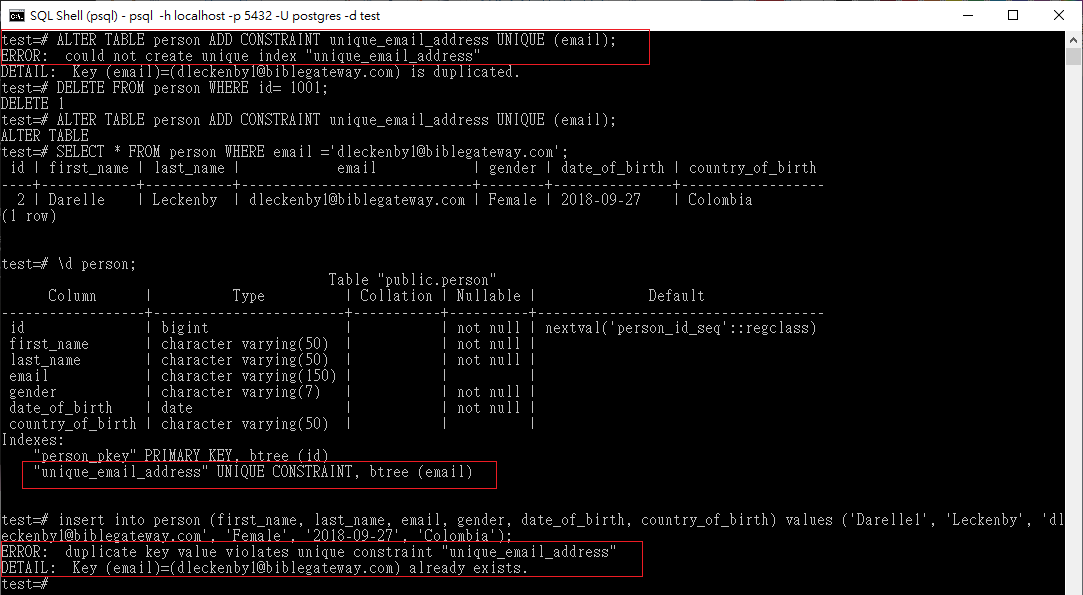
- SELECT email, COUNT(\*) FROM person GROUP BY email HAVING COUNT(\*) >1;

- SELECT \* FROM person WHERE email ='dleckenby1@biblegateway.com';



- To avoid duplicate emails and we use the following statements.

- ALTER TABLE person ADD CONSTRAINT unique\_email\_address UNIQUE (email);



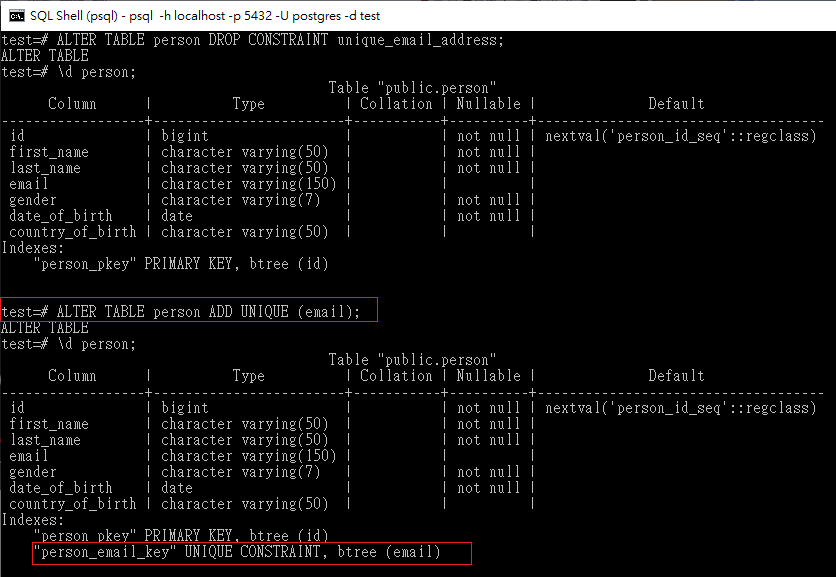
- ALTER TABLE person DROP CONSTRAINT unique\_email\_address;

- Below is other way to add key

- ALTER TABLE person ADD UNIQUE (email);

- Note : The deference between this way and the previous way, is that now we let the ***constraint name be defined by Postgres.***

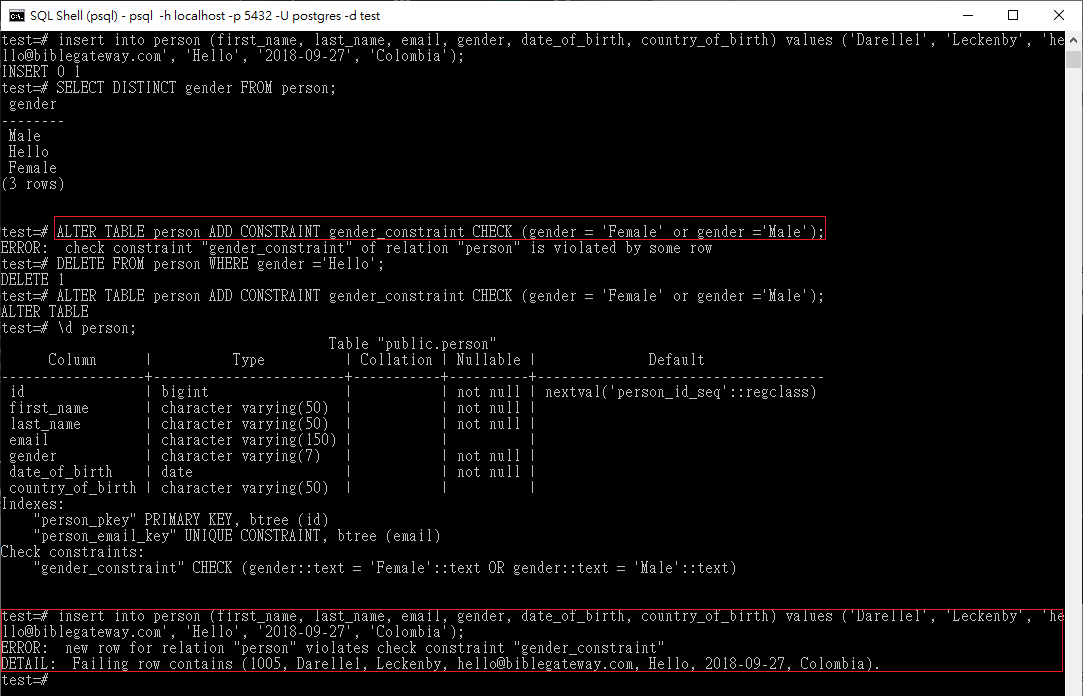
Unique constraints allows us to have a unique value per column is not the same as the primary key because primary keys job is to identify a record in a table.



**Check Constraints**

- ALTER TABLE person ADD CONSTRAINT gender\_constraint

CHECK (gender = 'Female' or gender ='Male');



**How to Delete Records**

- DELETE FROM person; <-- **DELETE ALL RECORDS FROM person**

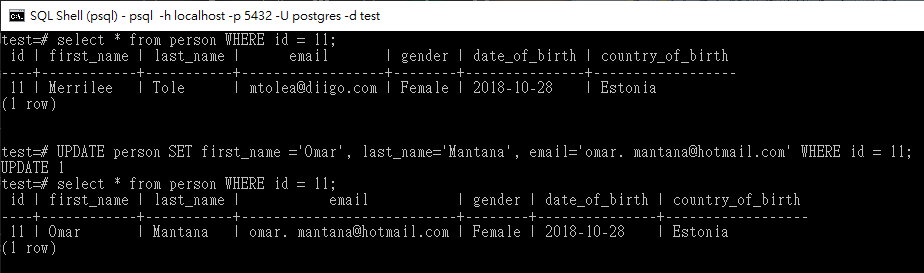
- DELETE FROM person WHERE id=1011;

- DELETE FROM person WHERE gender='Female' and country\_of\_birth='England';

**How to Update Records**

- UPDATE person SET email='felizio@gmail.com' WHERE id = 4;

- UPDATE person SET first\_name ='Omar', last\_name='Mantana', email='omar. mantana@hotmail.com' WHERE id = 11;



**On Conflict Do Nothing**

Note :only apply to the primary key or constraint

- INSERT INTO person (id, first\_name, last\_name, email, gender, date\_of\_birth,

country\_of\_birth) VALUES (17, 'test', 'test', '','Male', '2018-05-23', 'France');

- INSERT INTO person (id, first\_name, last\_name, email, gender, date\_of\_birth,

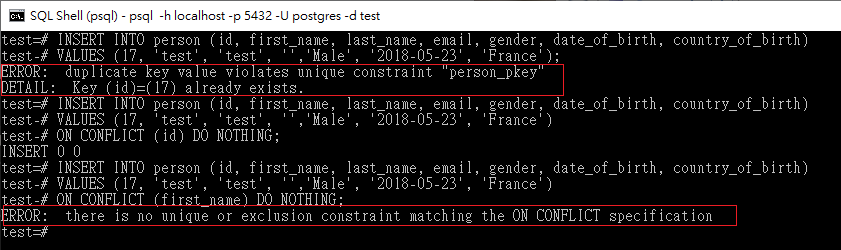
country\_of\_birth) VALUES (17, 'test', 'test', '','Male', '2018-05-23', 'France')

ON CONFLICT (id) DO NOTHING;

- INSERT INTO person (id, first\_name, last\_name, email, gender, date\_of\_birth,

country\_of\_birth) VALUES (17, 'test', 'test', '','Male', '2018-05-23', 'France')

ON CONFLICT (first\_name) DO NOTHING;



**Upsert**

- Override existing data, if present, otherwise, insert a new row

- INSERT INTO person (id, first\_name, last\_name, email, gender, date\_of\_birth,

country\_of\_birth) VALUES (17, 'test', 'test', '','Male', '2018-05-23', 'France')

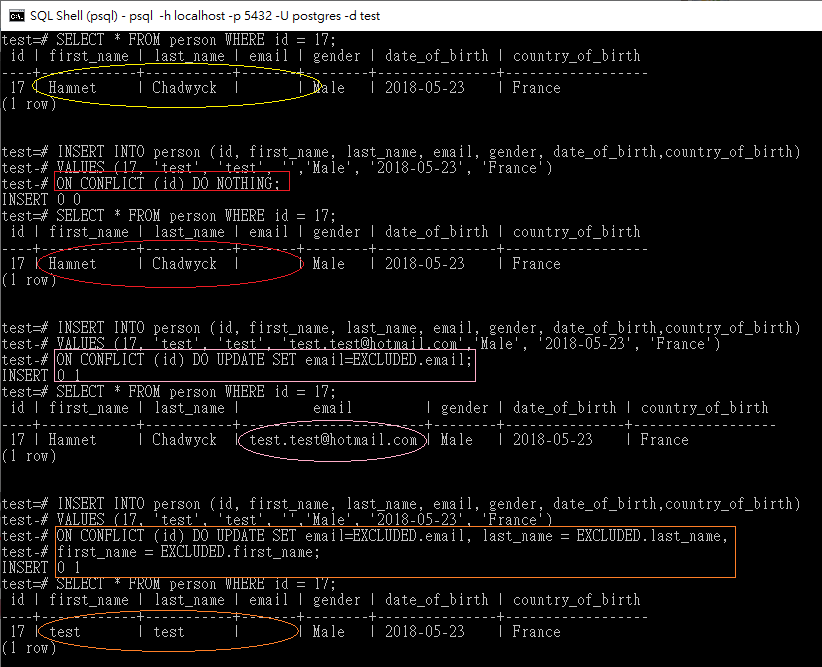
ON CONFLICT (id) DO ***UPDATE SET email = EXCLUDED.email***;

- INSERT INTO person (id, first\_name, last\_name, email, gender, date\_of\_birth,

country\_of\_birth) VALUES (17, 'test', 'test', '','Male', '2018-05-23', 'France')

ON CONFLICT (id) DO ***UPDATE SET email = EXCLUDED.email,***

***last\_name = EXCLUDED.last\_name, first\_name = EXCLUDED.first\_name*** ;



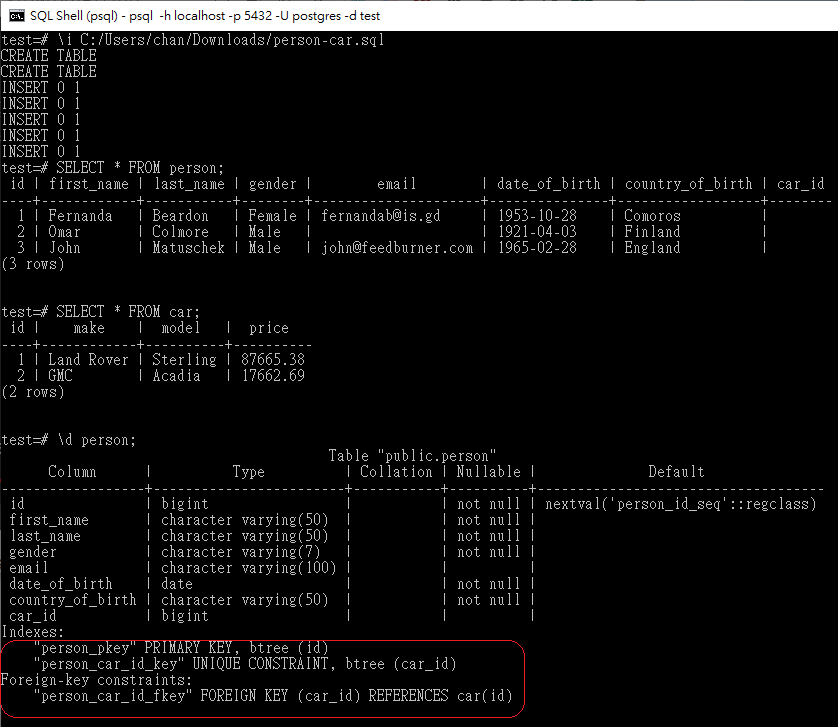
**Foreign Keys, Joins & Relationships**

- A Foreign Key is a column that references a primary key in another table. The type must be same.

**Adding Relationships Between Tables** (3:19:48)

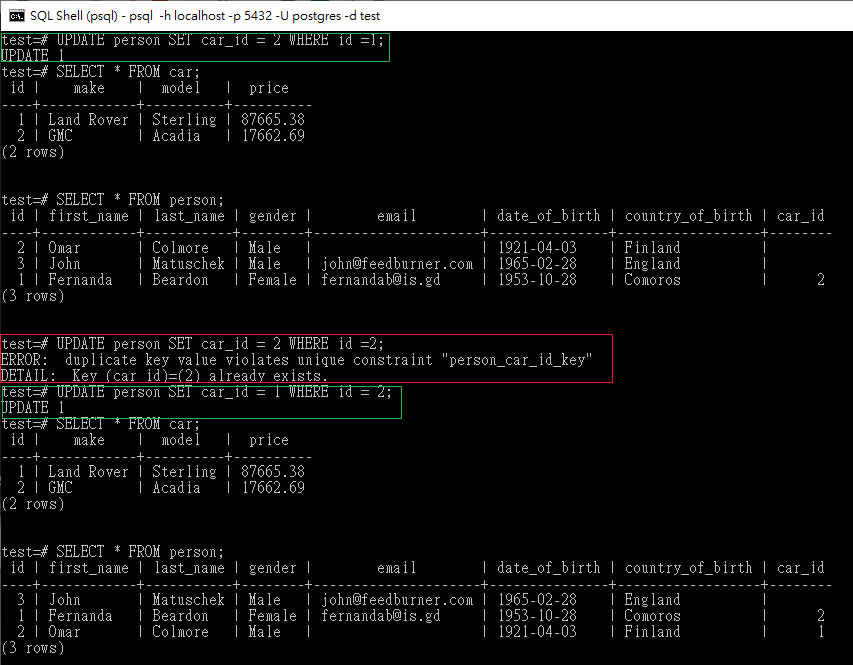
- Note : **BIGSERIAL is a special data type, which is managed by a sequence**. We need to change to BIGINT. BIGINT are pretty much the same in terms of the actual size.

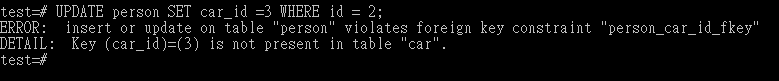
- \i C:/Users/chan/Downloads/person-car.sql



**Updating Foreign Keys Columns**

- A Foreign Key is a column that references a primary key in another table. The type must be same.



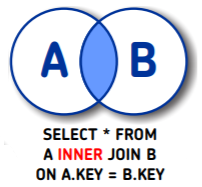


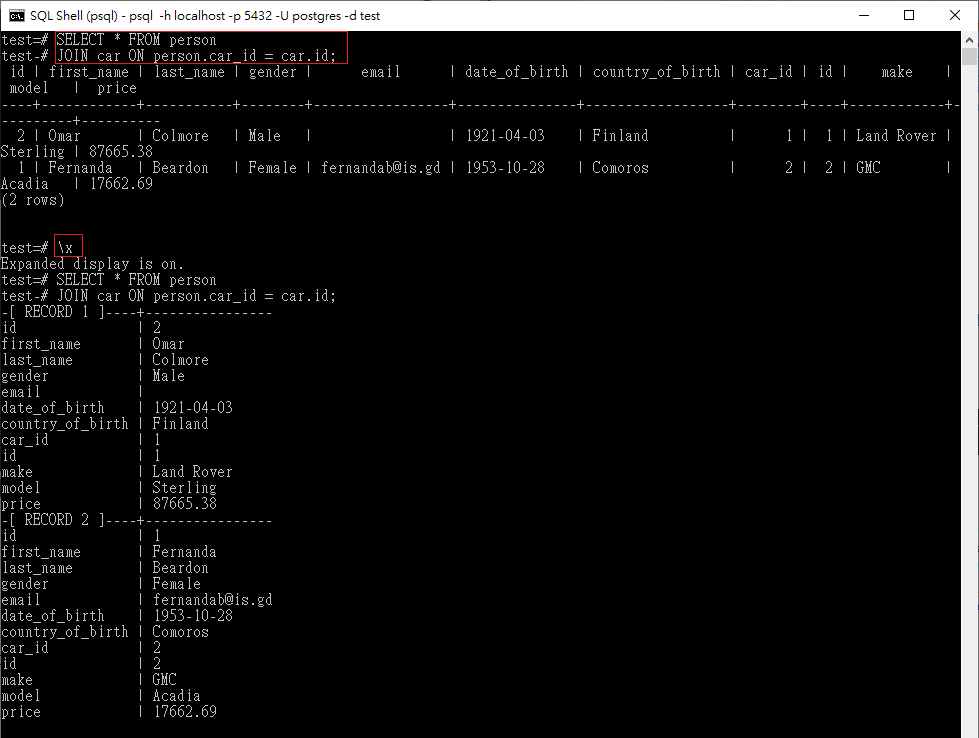
**Inner Joins**

- common in both tables (A & B). So if you have a record inside of the table A, and also a record inside of the table B. If you have a foreign key, which is present in both tables, then it takes those two records, and then gives you the result of both which we're going to call C. So A plus B equals to C. And to recap an image join takes two tables A and B. And then if we have a foreign key that is present in both tables, then we have a new record called C.

- SELECT \* FROM person

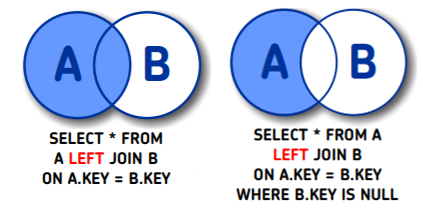
JOIN car ON person.car\_id = car.id;





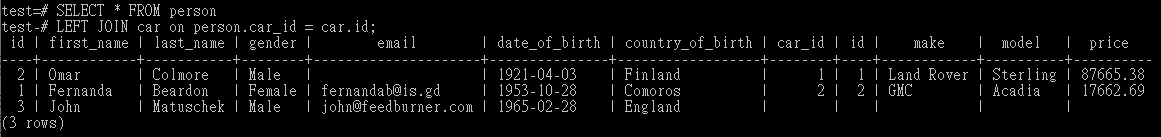
**Left Joins**

- Left joins allows us to combine two tables like inner joints. So Table A and table B. And the difference her e is that a left join include all the rows from the left table, ie Table A, as well as the records from table B that have a corresponding relationship. And also the ones that don't have a corresponding relationship. ie returns all the records, even if there isn't a match, and then you get result.



- SELECT \* FROM person

LEFT JOIN car ON person.car\_id = car.id;



- SELECT \* FROM person test-# WHERE car\_id is null;

- SELECT \* FROM person

LEFT JOIN car on person.car\_id = car.id

WHERE car.\* IS NULL;

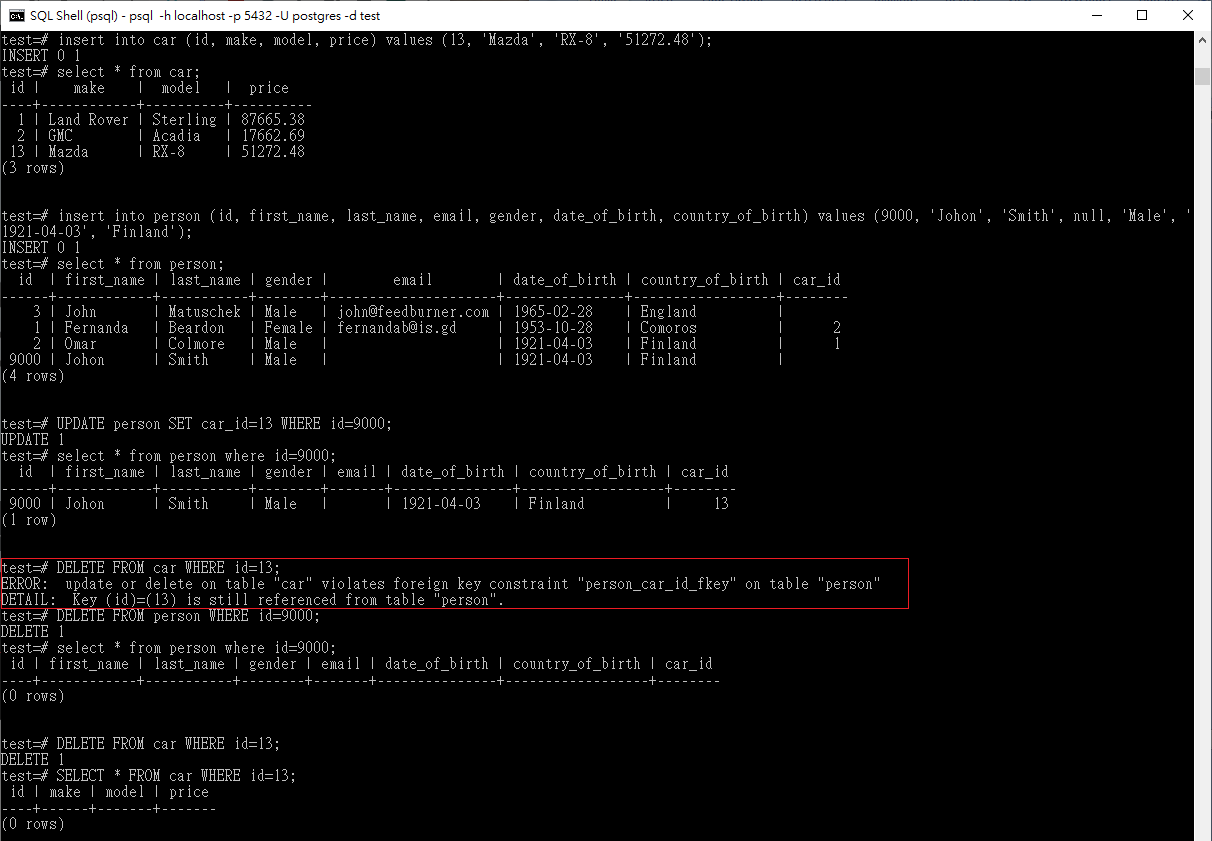


**Deleting Records with Foreign Keys**

- There are two options to do

1. Delete all records related to that foreign keys OR

2. Update the foreign key to NULL



**Exporting Query Results to CSV**

- \copy (SELECT \* FROM person LEFT JOIN car ON person.car\_id = car.id) TO 'C:/Users/chan/Downloads/output.csv' DELIMITER ',' CSV HEADER;

**Serial & Sequences**

-Will add 1 to the ID

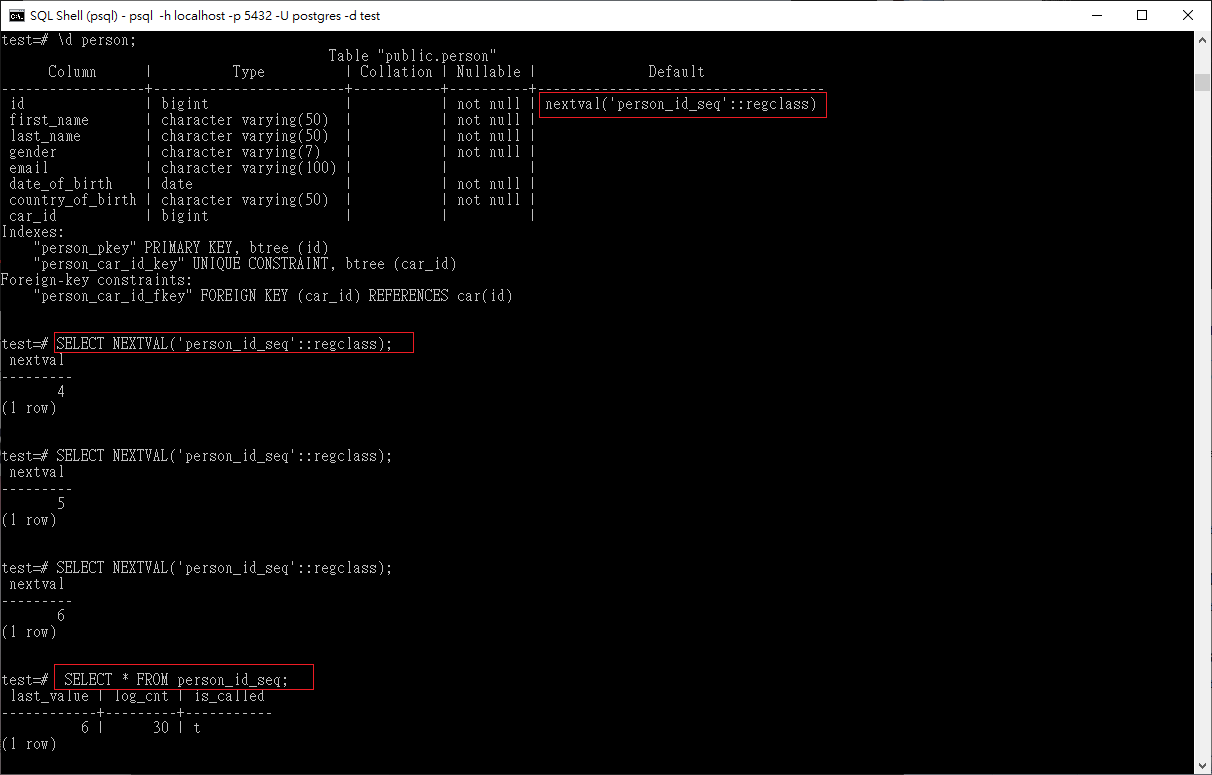
**SELECT NEXTVAL('person\_id\_seq'::regclass);**

- last value of the ID

**SELECT \* FROM person\_id\_seq;**

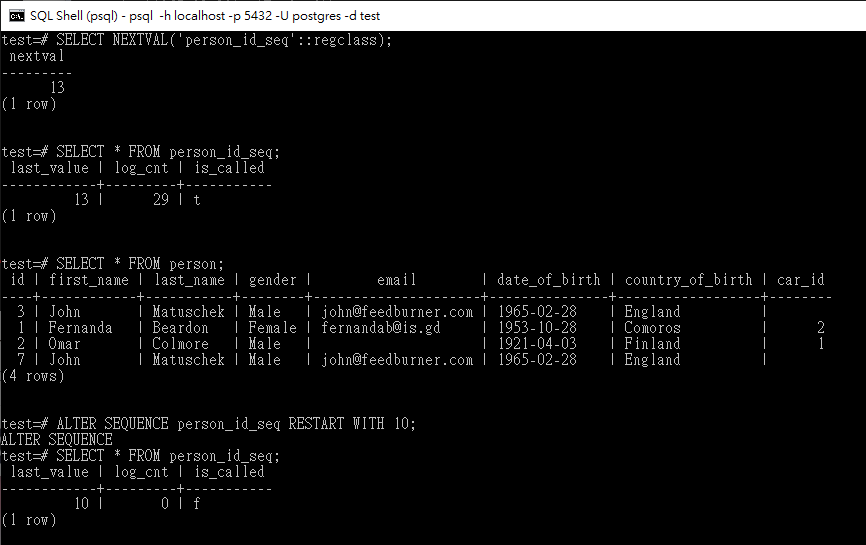
- restart the ID to a special number

**ALTER SEQUENCE person\_id\_seq RESTART WITH 10;**



- insert 1 record, then the ID will be 7.





**Extensions**

- [H.4. Extensions](https://www.postgresql.org/docs/current/external-extensions.html)

- just like features which are built in. So basically, extensions are simply functions that can add extra functionality to your database.

- list out all extensions

SELECT \* FROM pg\_available\_extensions;

**Understanding UUID Data Type**

- Universally unique identifier (UUID) allows us to have a guarantee unique identifier whenever the identifier is generated and also the cool thing about it is that is globally unique, which means that collisions is pretty much impossible. For example, IP Address and timestamp ([Universally unique identifier - Wikipedia](https://en.wikipedia.org/wiki/Universally_unique_identifier)).

-Install the extension to the database and below is to add UUID-OSSP

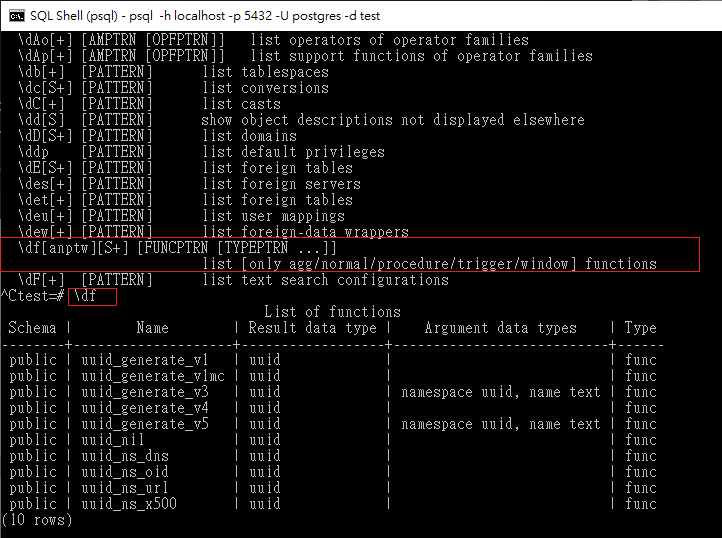
**CREATE EXTENSION IF NOT EXISTS "uuid-ossp";**

OUTPUT: CREATE EXTENSION;

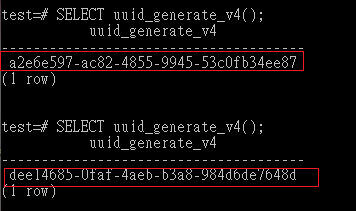
if you use " SELECT \* FROM pg\_available\_extensions;" to list all extensions, now you can find the version 1.1 in the **installed\_version** column.

- list [only agg/normal/procedure/trigger/window] functions:

**\df[anptw][S+] [FUNCPTRN [TYPEPTRN ...]]**



- We will use uuid\_generate\_v4 to generate a random number.

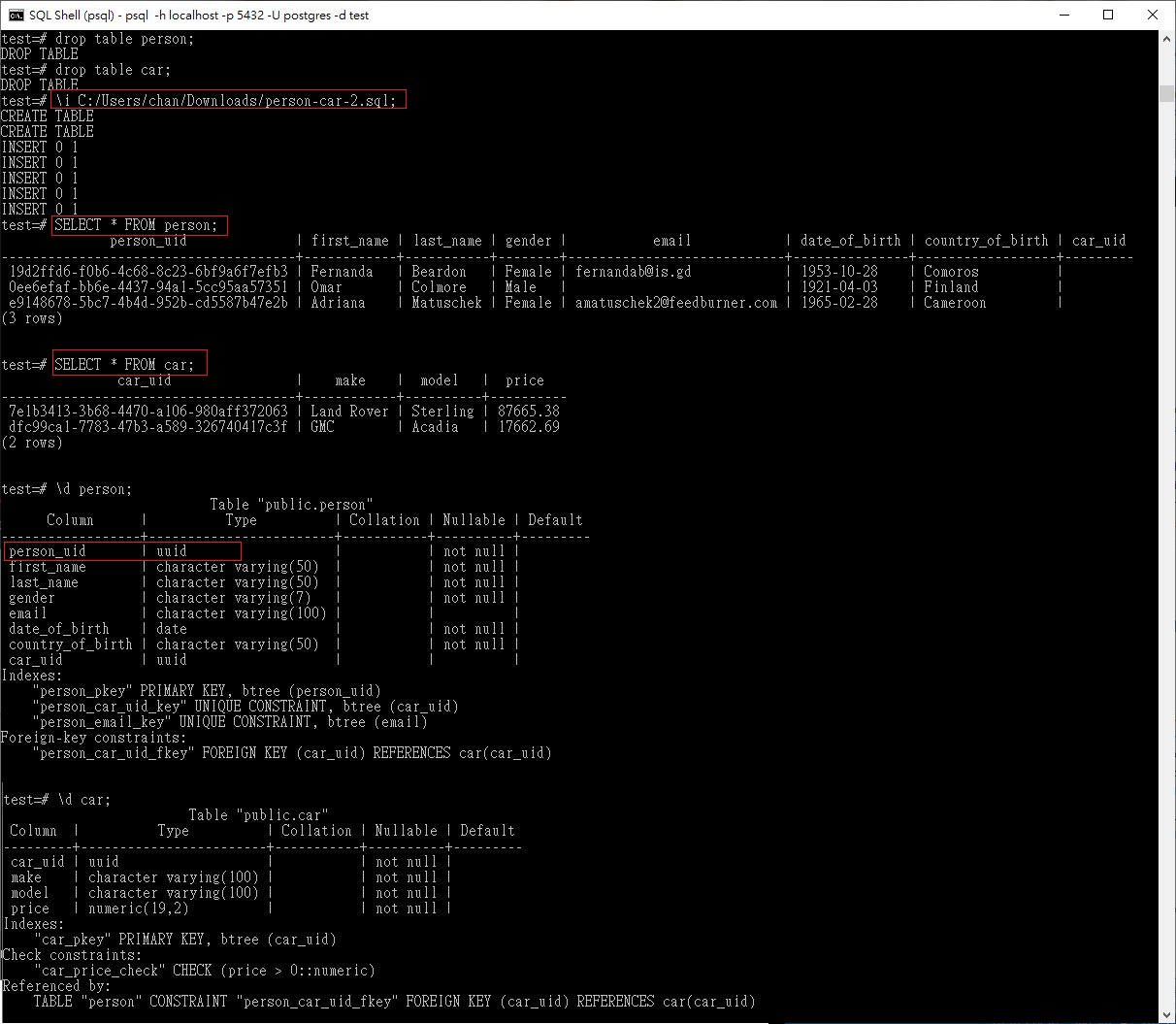


- This is a good faith for using UUID as primary keys in our tables . And one of the benefits of using us as keys is that it makes it very hard for attackers to try to mine our database. For example, if you had an API forward slash users, and then the actual user ID, so an attackers could actually exploit all the numbers. And try to delete everyone or update information, so on and so forth. But with you it's very, very difficult for them. To actually guess which person for example, is in your database. Another benefit is that because they are globally unique, that means you can migrate data across databases without any conflicts. So for example, if you had a database A and database B, and basically, if you were using big serial, so a big int or an int, then most likely you would have clashes when adding some data from database A into database B, because of the actual IDs, right? If you were using big serials, it's auto incremented. And basically, in both servers, there's no way to actually tell that IDs are different. And that's definitely a big advantage of using UUIDs.

**UUID As Primary Keys**

- Drop tables two person and car. Then run the below command:-

**C:/Users/chan/Downloads/person-car-2.sql;**



- Assign two cars to Adriana and Fernanda respectively. The command is as below:-

UPDATE person SET car\_uid='7e1b3413-3b68-4470-a106-980aff372063' WHERE person\_uid = 'e9148678-5bc7-4b4d-952b-cd5587b47e2b';

UPDATE person SET car\_uid='dfc99ca1-7783-47b3-a589-326740417c3f' WHERE person\_uid = '19d2ffd6-f0b6-4c68-8c23-6bf9a6f7efb3';

SELECT \* FROM person

JOIN car ON person.car\_uid = car.car\_uid;

SELECT \* FROM person JOIN car USING (car\_uid);

SELECT \* FROM person LEFT JOIN car USING (car\_uid);

SELECT \* FROM person LEFT JOIN car USING (car\_uid) WHERE car\_uid IS NULL;

**Conclusion**

Next steps

Spring Boot or NodeJS & Express

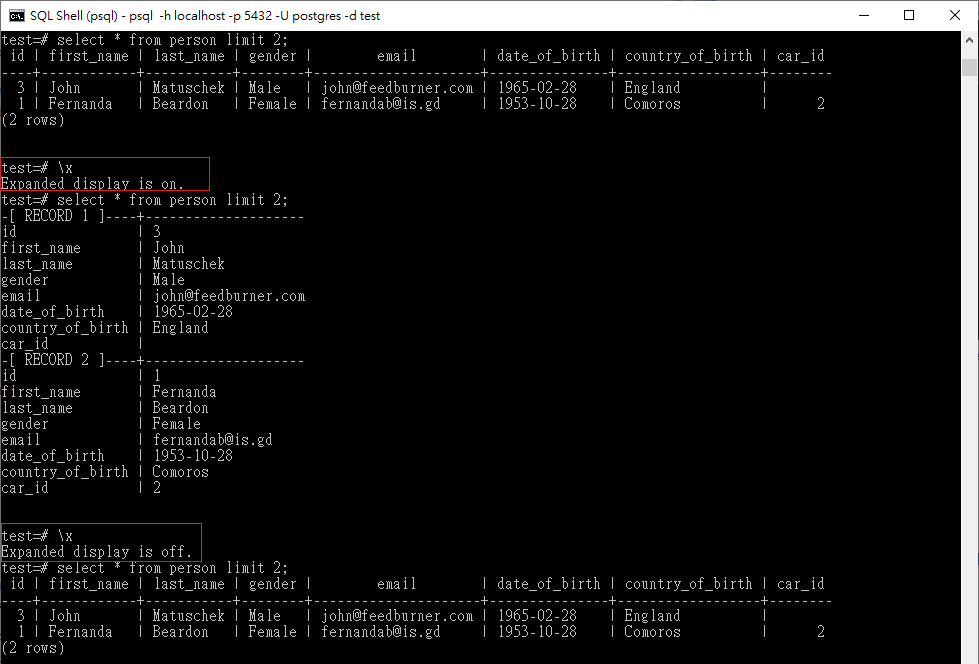
Advance PostgreSQL

**Contents**

* (0:03:16) What is a Database
* (0:05:17) What is SQL And Relational Database
* (0:09:10) What is PostreSQL AKA Postrgres
* (0:10:53) PostgreSQL Installation (Mac OS)
* (0:14:21) PostgreSQL Installation (Windows)
* (0:17:38) GUI Clients vs Terminal/CMD Clients
* (0:21:39) Setup PSQL (MAC OS)
* (0:25:22) Setup PSQL (Windows)
* (0:30:15) How to Create Database
* (0:33:35) How to Connect to Databases
* (0:38:12) **A Very Dangerous Command (*DROP DATABASE*)**
* (0:41:37) How To Create Tables
* (0:45:46) Creating Tables Without Constraints
* (0:49:12) Creating Tables with Constraints
* (0:55:55) Insert Into
* (0:59:14) Insert Into Example
* (1:02:36) Generate 1000 Rows with Mockaroo
* (1:12:28) Select From
* (1:15:18) Order By
* (1:19:53) Distinct
* (1:21:59) Where Clause and AND
* (1:25:29) Comparison Operators
* (1:29:35) Limit, Offset & Fetch
* (1:32:43) IN
* (1:35:43) Between
* (1:37:45) Like And iLike
* (1:43:10) Group By
* (1:46:41) Group By Having
* (1:52:08) Adding New Table And Data Using Mockaroo
* (1:55:40) Calculating Min, Max & Average
* (1:59:48) Sum
* (2:01:55) Basics of Arithmetic Operators
* (2:05:59) Arithmetic Operators (ROUND)
* (2:09:43) Alias
* (2:12:32) Coalesce
* (2:16:15) NULLIF
* (2:20:21) Timestamps And Dates Course
* (2:23:21) Adding And Subtracting With Dates
* (2:25:58) Extracting Fields From Timestamp
* (2:27:28) Age Function
* (2:29:24) What Are Primary Keys
* (2:31:23) Understanding Primary Keys
* (2:36:26) Adding Primary Key
* (2:40:55) Unique Constraints
* (2:49:15) Check Constraints
* (2:54:45) How to Delete Records
* (3:01:36) How to Update Records
* (3:05:55) On Conflict Do Nothing
* (3:11:09) Upsert
* (3:16:41) What Is A Relationship/Foreign Keys
* (3:19:48) Adding Relationship Between Tables
* (3:25:04) Updating Foreign Keys Columns
* (3:29:30) Inner Joins
* (3:35:17) Left Joins
* (3:40:53) Deleting Records With Foreign Keys
* (3:47:27) Exporting Query Results to CSV
* (3:50:42) Serial & Sequences
* (3:57:18) Extensions
* (3:59:39) Understanding UUID Data Type
* (4:05:54) UUID As Primary Keys
* (4:16:30) Conclusion

**Other command**

1. Clean screen :　**\! cls**
2. list tables, views and sequences: \d
3. list table only : **\dt**
4. toggle expanded output (currently off) : **\x** [on|off|auto]



**Error**

1. Unable to connect to server: could not connect to server: Connection refused (0x0000274D/10061) Is the server running on host "localhost" (::1) and accepting TCP/IP connections on port 5432? could not connect to server: Connection refused (0x0000274D/10061) Is the server running on host "localhost" (127.0.0.1) and accepting TCP/IP connections on port 5432?

**the solution was: Start -> Control panel -> Administration -> Services -> postgresql-x64-12 - start or restart**

**Reference**

[PostgreSQL: Documentation](https://www.postgresql.org/docs/)

Data Types: [https://www.postgresql.org/docs/current/datatype.html](%20https:/www.postgresql.org/docs/current/datatype.html)

Chapter 9. Functions and Operators :　[PostgreSQL: Documentation: 14: Chapter 9. Functions and Operators](https://www.postgresql.org/docs/current/functions.html)

* Comparison Functions and Operators: [PostgreSQL: Documentation: 14: 9.2. Comparison Functions and Operators](https://www.postgresql.org/docs/14/functions-comparison.html)
* Mathematical Functions and Operators : [PostgreSQL: Documentation: 14: 9.3. Mathematical Functions and Operators](https://www.postgresql.org/docs/14/functions-math.html)
* String Functions and Operators : [PostgreSQL: Documentation: 14: 9.4. String Functions and Operators](https://www.postgresql.org/docs/14/functions-string.html)
* Binary String Functions and Operators : [PostgreSQL: Documentation: 14: 9.5. Binary String Functions and Operators](https://www.postgresql.org/docs/14/functions-binarystring.html)
* Bit String Functions and Operators : [PostgreSQL: Documentation: 14: 9.6. Bit String Functions and Operators](https://www.postgresql.org/docs/14/functions-bitstring.html)
* Pattern Matching : [PostgreSQL: Documentation: 14: 9.7. Pattern Matching](https://www.postgresql.org/docs/14/functions-matching.html)