What is a Database

* Store
* Manipulate
* Retrieve
  + Every data goes to computer server

PostgreSQL is database engine

SQL is a structure language

SELECT first\_name FROM person

How does table store a data?

* By having Columns & Rows

Relational Database

* Two or more tables have relational.

Connecting to DB server by:

* Connect using client
* GUI Client
* Terminal/ CMD
* Application
  + Pg Admin is what we will use
    - Graphical UI for database.

Go onto the terminal and type, in order to create a table: CREATE DATABASE test;

Type \l (it’s like ls on termainal, ls means list so backlash “l”ist)

Type psql –help to see the commandline

Since you created one database called test, type

psql -h localhost -p 5432 -U jwoonk test

\q to exist

For making short you can just type \c test after you joined psql

**A very dangerous command!!!**

After you create database, in order to remove, you DROP DATABASE [database name];

Removing is very dangerous move because you can lose data permantely.

How to create table with Postgres:

CREATE TABLE table\_name (

Column name + data type + constraints if any

)

Example.

CREATE TABLE person (

**id** int,

first\_name VARCHAR (50),

last\_name VARCHAR (50),

gender VARCHAR(6),

date\_of\_birth date,

)

<https://www.postgresql.org/docs/12/datatype.html>

if you want to look at what datatype you can select then go to TABLE 8.1 DATA TYPES

After you create a table, in order to check, type \d

List of relations

Schema | Name | Type | Owner

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

public | person | table | jwoonk

(1 row)

Type \d [table name]

Since you just created table called person

Type \d person

IF you want to drop a table simply insert DROP TABLE [table-name]

Notice that you there is difference between database and table?

Example of adding constraint (conditional)

Example.

CREATE TABLE person (

**id** BIGSERIAL NOT NULL PRIMARY KEY,

first\_name VARCHAR (50) NOT NULL,

last\_name VARCHAR (50) NOT NULL,

gender VARCHAR(6) NOT NULL,

date\_of\_birth date NOT NULL,

email VARCHAR(150)

);

After you create it, this will show

test=# \d

List of relations

Schema | Name | Type | Owner

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

public | person | table | jwoonk

public | person\_id\_seq | sequence | jwoonk

(2 rows)

person\_id\_seq is there and type is sequence. This is not a table is has to do with BIGSERIAL that you put when making column for id. => autoincrementing eight-byte integer

Table "public.person"

Column | Type | Collation | Nullable | Default

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

id | bigint | | not null | nextval('person\_id\_seq'::regclass)

first\_name | character varying(50) | | not null |

last\_name | character varying(50) | | not null |

gender | character varying(7) | | not null |

date\_of\_birth | date | | not null |

email | character varying(150) | | |

Indexes:

"person\_pkey" PRIMARY KEY, btree (id)

HOW TO INSERT RECORDS INTO TABLE

So far we have DB called test, and person table.

Person table has id, first\_name, last\_name, gender, date\_of\_birth and email.

Let’s try to add someone in that table.

INSERT INTO person (

first\_name,

last\_name,

gender,

date\_of\_birth)

VALUES (‘Anne’, ‘Smith’, ‘Female’, DATE ‘1988-01-09’);

Notice that id value is not inserted because it’s sequenced id

INSERT INTO person (

first\_name,

last\_name,

gender,

date\_of\_birth)

VALUES (‘Jake’, ‘Jones’, ‘Male’, DATE ‘1990-01-10’, ‘jake@gmail.com’);

\*\*MAKE SURE to use single quote\*\*

So basically

INSERT INTO [table name] (

Col name)

VALUES (‘[col1.value]’,‘[so forth]’);

If you want to check what table you have in a database, \dt

<https://www.mockaroo.com/>

Will help you to generate 1000 rows of table for you. When you’re downloading make sure to include create table.

After download the file, modify little bit of table like which one you want to make it nullify or no nullify. Then get the directory of the file location and open terminal again to type

For an example => \i /Users/jwoonk/Downloads/person.sql;

This will bring the file into the database

HOW TO LOOK INSIDE THE TABLE?

After table being added to a DB then use command to see what’s inside like SELECT \* FROM person;

How to Sort An Order?

1 2 3 4 5 ASC (THIS IS DEFAULT)

5 4 3 2 1 DESC

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

Will give the sort of ascending of country of birth.

You can also sort by two attributes like

SELECT \* FROM person ORDER BY id, email;

If I want to find value once a time, use DISTINCT:

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

IF you want select data conditionally(filter it) you can use WHERE:

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 = ‘Pietersma’;

Comparison Operator = SELECT:

For example SELECT 1 =1; will show t meaning true

**Limit, Offset & Fetch**

If you want to limit table by 10:

SELECT \* FROM person LIMIT 10;

If you want to select after 5th row of the table then limit 10, you will need to use OFFSET

SELECT \* FROM person OFFSET 5 LIMIT 10;

SQL usually never used LIMIT by its standard but it became a thing, if you want to stay traditional then use Fetch.

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

SELECT \* FROM person OFFSET 5 FETCH FIRST ONLY; will only get 6th row of the table.

**IN**

In will save time of line of command like SELECT \* FROM person WHERE country\_of\_birth = ‘China’ OR WHERE country\_of\_birth = ‘France’ OR WHERE country\_of\_birth = ‘Brazil’;

Using IN will save it like this:

SELECT \* FROM person WHERE country\_of\_birth IN (‘China’,’Brazil’,’France’)

You can use ORDER BY along with it too.

**Between**

Between will use as range like for example:

SELECT \* FROM person

WHERE date\_of\_birth

BETWEEN DATE ‘2000-01-01’ AND ‘2015-01-01’;

**LIKE**

If you want to filter data by letters LIKE will help you do that:

SELECT \* FROM person WHERE email LIKE ‘%.com’

SELECT \* FROM person WHERE email LIKE ‘%gmail.com’

SELECT \* FROM person WHERE email LIKE ‘%gmail.%’

SELECT \* FROM person WHERE email LIKE ‘\_ \_ \_ \_ \_@%’ -> this one grabs five letters of email address.

NOTICE letter is case sensitive!

**GROUP BY**

SELECT DISTINCT country\_of\_birth FROM person will grab country name in db once a time.

But in order to find out how many ppl are in this each country then use GROUP BY.

This will count people in each country.

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

This will order the country by ABC order.

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

**GROUP BY HAVING**

If you want to make more filter like after grouping by, you can use having:

SELECT country\_of\_birth, COUNT(\*) FROM person GROUP BY country\_of\_birth **HAVING COUNT(\*) > 5 ORDER BY** country\_of\_birth (this will order the country by ABC order)

This will count countries of people that have more than 5.

You can refer more of command on this link:

<https://www.postgresql.org/docs/12/functions-aggregate.html>

**ADDING NEW TABLE AND DATA USING MOCKAROO**

Ultimately you have to use select also.

Right now we will use MAX, MIN, AVG, and SUM

Just now made another mock data that contains car data. Inserted by \i [directory]

Then you can use MAX if you want to find out the maximum of value:

SELECT MAX(price) FROM car;

max

----------

99887.56

(1 row)

Or Minimum:

SELECT MIN(price) FROM car;

min

----------

10056.12

(1 row)

Or Average:

SELECT AVG(price) FROM car;

avg

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

53992.651550000000

(1 row)

Since there are long decimals, you can round it

SELECT ROUND(AVG(price)) FROM car;

round

-------

53993

(1 row)

This will grab the highest price of each make and model

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

make | model | min

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

Mitsubishi | Sigma | 35679.98

Mercedes-Benz | S-Class | 13607.76

BMW | M6 | 41502.43

Infiniti | FX | 64196.43

Saturn | Aura | 77424.73

Kia | Amanti | 99145.05

Subaru | Forester | 19005.87

Ford | Econoline E350 | 87711.31

Jeep | Grand Cherokee | 44946.49

….

This will grab the lowest price of each make and model

SELECT make, MIN(price) FROM car GROUP BY make;

If you want to grab total amount of every car then use SUM:

SELECT SUM(price) FROM car;

sum

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

53992651.55

(1 row)

If you want to group it and more specific of each car’s make:

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

make | sum

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

GMC | 2457206.39

Maybach | 203082.97

Lincoln | 1031274.38

Honda | 1232501.04

Ram | 131602.00

….

What if we want to have discount price of 10% of each car price?

SELECT id, make, model, price, ROUND(price\*.10, 2) FROM car;

(number, 2 decimal)

SELECT id, make, model, price, ROUND(price\*.10, 2), ROUND(price – (price\*.10), 2) FROM car;

ALIAS (AS)

This one is to re-name a column.

SELECT id, make, model, price AS original\_price, round(price \*.10) AS discounted\_value FROM car;

**Coalesce**

Basically allows the default values to check if it exist or not, so the value of it does exist then it will out the existed value

SELECT COALESCE(1);

coalesce

----------

1

(1 row)

SELECT COALESCE(null, 1);

coalesce

----------

1

(1 row)

SELECT COALESCE(null, null, 1, 10);

coalesce

----------

1

(1 row)

The reason why to use this method is to query data things that don’t exist inside the rows.

For example SELECT email FROM person; will query all people’s email WITH email address not given as blank. So you want to use Coalesce to fill those empty cells. SELECT COALESCE(email, ‘No email’) FROM person;

SELECT id, make, model, price, price \*.10, FROM car;

coalesce

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

vjirusek0@stumbleupon.com

etapin1@usgs.gov

No email

No email

No email

handrasch5@taobao.com

No email

aloosmore7@list-manage.com

[hgarron8@devhub.com](mailto:hgarron8@devhub.com)

…..

NULLIF

SELECT 10 / 0;

Nullif takes two arguments, returns null (blank) if first argument if matches with second argument.

SELECT NULLIF(10, 10);

nullif

--------

(1 row)

If they don’t match, it returns first argument back:

SELECT NULLIF(100, 19);

nullif

--------

100

(1 row)

So we can utilize Nullif like this:

SELECT 10 / NULLIF(2,9);

?column?

----------

5

(1 row)

Little bit complex example:

SELECT COALESCE(10 / NULLIF(0,0), 0);

Will return 0. Not null.

TIMESTAMPS AND DATES

SELECT NOW();

now

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

2020-07-01 00:35:57.413535-07

(1 row)

IF you want only date you can SELECT NOW()::DATE;

now

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

2020-07-01

(1 row)

IF you want only time you can SELECT NOW()::TIME;

now

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

00:38:13.742725

(1 row)

<https://www.postgresql.org/docs/12/datatype-datetime.html>

ADDING AND SUBTRACTING WITH DATES

If you want to going back one year back from current date

SELECT NOW() – INTERVAL ‘1 YEAR’;

?column?

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

2019-07-01 01:44:39.473616-07

(1 row)

Or 10 years

SELECT NOW() - INTERVAL '10 YEAR';

?column?

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

2010-07-01 01:45:02.153523-07

(1 row)

Or 10 months

SELECT NOW() - INTERVAL '10 MONTH';

?column?

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

2019-09-01 01:45:53.184921-07

(1 row)

SELECT NOW() - INTERVAL '10 MONTH 10 DAYS';

?column?

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

2019-08-22 01:46:21.130064-07

(1 row)

You can also add (+);

This can be useful if you want to add x amounts of months.

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

?column?

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

2021-05-11 00:00:00

(1 row)

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

date

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

2021-05-11

(1 row)

You can extract specific date:

SELECT EXTRACT(YEAR FROM NOW());

date\_part

-----------

2020

(1 row)

SELECT EXTRACT(MONTH FROM NOW());

date\_part

-----------

7

(1 row)

SELECT EXTRACT(DOW FROM NOW());

date\_part

-----------

3 (3 means Wednesday)

(1 row)

AGE FUNCTION

SELECT first\_name, last\_name, gender, country\_of\_birth, date\_of\_birth,AGE(NOW(), date\_of\_birth)AS AGE FROM person;

first\_name | last\_name | gender | country\_of\_birth | date\_of\_birth | age

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

Vito | Jirusek | Male | Nigeria | 1991-09-26 | 28 years 9 mons 5 days 01:57:39.398776

Esme | Tapin | Male | France | 1996-07-05 | 23 years 11 mons 27 days 01:57:39.398776

Marena | Hughson | Female | China | 2014-06-29 | 6 years 2 days 01:57:39.398776

Clywd | Smidmore | Male | Indonesia | 1995-02-22 | 25 years 4 mons 7 days 01:57:39.398776

Norma | Nazer | Female | China | 2019-10-14 | 8 mons 18 days 01:57:39.398776

Hana | Andrasch | Female | China | 2007-06-28 | 13 years 3 days 01:57:39.398776

Nanni | Macrow | Female | Japan | 2011-08-05 | 8 years 10 mons 27 days 01:57:39.39877

PRIMARY KEYS (FIRST HOW DO WE REMOVE PRIMARY KEY?)

Let’s say if one table has two persons’ names that are same how we are going to distinguish? By primary keys (or ID like passport\_number).

ID is usually something you cannot be duplicated unless you make alter table.

First find out the name of PRIMARY KEY. In person table, primary key is person\_pkey.

Then:

ALTER TABLE person DROP CONSTRAINT person\_pkey;

You can add with the same id.

SELECT \* FROM person WHERE id = 1;

id | first\_name | last\_name | email | gender | date\_of\_birth | country\_of\_birth

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

1 | Vito | Jirusek | vjirusek0@stumbleupon.com | Male | 1991-09-26 | Nigeria

1 | Vito | Jirusek | vjirusek0@stumbleupon.com | Male | 1991-09-26 | Nigeria

1 | John | Jirusek | vjirusek0@stumbleupon.com | Male | 1991-09-26 | Nigeria

(3 rows)

ADDING PRIMARY KEY BACK

In order to add PK. Then make alter table.

ALTER TABLE person ADD PRIMARY KEY(id);

UNIQUE CONSTRAINTS

In order to find a value that has duplicated like email addresses. Sometimes two users could use one same email:

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

If there is two users using same email can be problematic. If we were to send a email to each user, how do we know which username should we sent under? So we have to make email as constraint (unique value) to prevent this scenario beforehand.

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

-If you don’t want to name a constraint then you can just type, ALTER TABLE person ADD UNIQUE (email);

After this will add another constraint of one of columns.

Table "public.person"

Column | Type | Collation | Nullable | Default

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

id | bigint | | not null | nextval('person\_id\_seq'::regclass)

first\_name | character varying(50) | | not null |

last\_name | character varying(50) | | not null |

email | character varying(150) | | |

gender | character varying(7) | | not null |

date\_of\_birth | date | | not null |

country\_of\_birth | character varying(50) | | |

Indexes:

"person\_pkey" PRIMARY KEY, btree (id)

"unique\_email\_address" UNIQUE CONSTRAINT, btree (email).

DELETE FROM person WHERE id = ####; Will delete a certain user.

If you want to delete a constraint simply type, ALTER TABLE person DROP CONSTRAINT [constraint-name];

CHECK CONSTRAINTS

If you want to regulate a user to choose more than what you expected such as gender, you might want two genders (Female and Male) in your website.

ALTER TABLE person ADD CONSTRAINT gender\_contraint CHECK (gender = 'Female' OR gender = 'Male');

Table "public.person"

Column | Type | Collation | Nullable | Default

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

id | bigint | | not null | nextval('person\_id\_seq'::regclass)

first\_name | character varying(50) | | not null |

last\_name | character varying(50) | | not null |

email | character varying(150) | | |

gender | character varying(7) | | not null |

date\_of\_birth | date | | not null |

country\_of\_birth | character varying(50) | | |

Indexes:

"person\_pkey" PRIMARY KEY, btree (id)

"unique\_email\_address" UNIQUE CONSTRAINT, btree (email)

Check constraints:

**"gender\_contraint" CHECK (gender::text = 'Female'::text OR gender::text = 'Male'::text)**

DELETE RECORDS

Trying to use WHERE keyword to filter out before what you want to delete otherwise there’s risk of you removing everydata.

If you want to be specific you can add AND after WHERE like,

DELETE FROM person WHERE gender = ‘Female’ AND country\_of\_birth = ‘England’;

Update Records

UPDATE person SET email = ‘ommar@gmail.com’ where id = 2011;

UPDATE [table name] SET [email column] = ‘email adress’, [name column] = ‘changed name’ where [column] = ‘’ or #;

TIP for using UPDATE, always try to use WHERE otherwise you might update whole DB.

ON CONFLICT DO NOTHING

If one of the constraint value existed already before inserting, it’s going to give an error:

INSERT INTO person (id, first\_name, last\_name, email, gender, date\_of\_birth, country\_of\_birth) VALUES (2, Esme, Rapin, john@mail.com, Male, DATE '1996-07-05', France);

ERROR: duplicate key value violates unique constraint "person\_pkey"

DETAIL: Key (id)=(2) already exists.

But if you want to bypass this error you can use ON CONLFICT ([constrant]) DO [something]

Read UPSERT

UPSERT

INSERT INTO person (id, first\_name, last\_name, email, gender, date\_of\_birth, country\_of\_birth) VALUES (2, Esme, Rapin, john@mail.**edu**, Male, DATE '1996-07-05', France)

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

Notice that the email address domain has changed.

You can add more values after SET email… by simply putting comma, then change values.

FOREIGN KEYS, JOINS & RELATIONSHIPS

ADDING RELATIONSHIPS BETWEEN TABLES

If you want one person owns one car and one car owns by one person.

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,

email VARCHAR(100),

date\_of\_birth DATE NOT NULL,

country\_of\_birth VARCHAR(50) NOT NULL,

**car\_id BIGINT REFERENCES car (id)** #notice that this one is NOT NULL because a person might not own a car.

UNIQUE(car\_id)

);

CREATE TABLE car (

id BIGSERIAL NOT NULL PRIMARY KEY,

make VANCHAR(100) NOT NULL,

model VARCHAR(100) NOT NULL,

price NUMERIC(19,2) NOT NULL

);

Make sure to create car table first before person because car is not defined yet if you create person table first.

UPDATING FOREIGN KEYS COLUMNS

You can simply use update as you like to update,

UPDATE person SET car\_id = 1 WHERE id = 2;

If you want a person’s id 9000 to have car\_id where it’s blank or null then use UPDATE command.

UPDATE person SET car\_id = 13 WHERE id = 9000;

INNER JOINS

How do we combine two tables? By using inner joins.

TABLE A + TABLE B = C, how do we put this login in sql syntax:

SELECT \* FROM person JOIN car ON person.car\_id = car.id;

(\x will let you show the display expanded.)

This will show the table that connects with car\_id on person’s table which by foreign keys

LEFT JOINS

The problem with INNER JOINS, sometimes you want to show all person’s data on person table that joined with car table but some people might not have cars, INNER JOINS wont show those people on JOIN TABLE. So using LEFT JOINS will help you to lay out all people even if they are not connect to car table.

SELECT \* FROM person

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

If you want to see join table that person does not own a car in other word where a foreign key is null (or blank) you simply use WHERE like,

SELECT \* FROM person LEFT JOIN car ON car.id = person.car\_id WHERE car.\* IS NULL;

DELETING RECORDS WITH FOREIGN KEYS

If you are trying to delete a row that has connected to other table (like person to car table).

Delete the car info that a person might be connect to car\_id, then delete the person’s info first so it get disconnect with the car table ,then you will be able to delete that row of car.

\*\*CASCADE will letting you to delete a row that connects with other table, by passing foreign key’s default rules. BUT it’s not recommended to use this because you want to have full control of your data, not merely abling to delete whatever the data is telling you to delete.

Exporting Query Results To CSV

Use \? To find out how to export files under input/output section. Genenrally people would use this command:

**\copy (SELECT \* FROM person LEFT JOIN car ON car.id = person.car\_id) TO ‘[your directory where you want to save]/[file’s name]’ DELIMITER ‘,’ CSV HEADER;**

SERIAL & SEQUENCES

BIGSERIAL in sql will be placed as bigint but looking at default will say nextval(‘[something\_id\_sql’::regclass])

What it means that whenever we are creating a row in that talbe will auto increment. Using nextval will let its number to increment.

Reason why you want to utilize this is to make sure to have place a value in a certain value you want. For example, you might want add a person in the table where has only 3 people in that table. The new person wants to be 10th id when created. You can eaither use nextval to increament to that point or use ALTER SEQUENCE person\_id\_seq RESTART WITH 9; (Notice it’s not 10 because next row you are adding will start after 9 which is 10th)

EXTENSIONS

There are extensions you can use for postgres, simply type,

SELECT \* FROM pg\_available\_extensions;

One of the useful extension you can use is uuid-ossp, it will generate universally unique IDs.

HOW TO INSTALL AN EXTENSION

Type on sql command line,

CREATE EXTENSION IF NOT EXISTS “[extension name from the extension list]”;

Notice it’s double quote!

(IF NOT EXSITS, saying if it wasn’t installed, install this.)

Example:

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE EXTENSION

UNDERSTAND UUID DATA TYPE

It generates a number that always be unique value, so it’s good for primary key setting.

HOW TO USE AN INSTALLED EXTENSION?

Type on sql command line,

\df

Example:

test=# \df

List of functions

Schema | Name | Result data type | Argument data types | Type

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

public | uuid\_generate\_v1 | uuid | | func

public | uuid\_generate\_v1mc | uuid | | func

public | uuid\_generate\_v3 | uuid | namespace uuid, name text | func

public | uuid\_generate\_v4 | uuid | | func

public | uuid\_generate\_v5 | uuid | namespace uuid, name text | func

public | uuid\_nil | uuid | | func

public | uuid\_ns\_dns | uuid | | func

public | uuid\_ns\_oid | uuid | | func

public | uuid\_ns\_url | uuid | | func

public | uuid\_ns\_x500 | uuid | | func

(10 rows)

Will give you the list which extensions are installed and the variations of usage you can use in each extension.

Notice the name column that’s the name you can use as a function to invoke.

You can test how the function works by SELECT [function name]();

Example:

SELECT uuid\_generate\_v4();

uuid\_generate\_v4

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

fd142799-bdf3-4eb3-b9fa-0078ca1e48aa

(1 row)

UUID As Primary Keys

CREATE TABLE car (

**car\_uid UUID** NOT NULL PRIMARY KEY,

make VANCHAR(100) NOT NULL,

model VARCHAR(100) NOT NULL,

price NUMERIC(19,2) NOT NULL

);

CREATE TABLE person (

**person\_uid UUID** NOT NULL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

gender VARCHAR(7) NOT NULL,

email VARCHAR(100),

date\_of\_birth DATE NOT NULL,

country\_of\_birth VARCHAR(50) NOT NULL,

**car\_id UUID REFERENCES car(car\_uid)**

UNIQUE(car\_uid),

UNIQUE(email)

);

---INSERT INTO PERSON

Insert into person(**person\_uid**, first\_name, last\_name, gender, email, date\_of\_birth, country\_of\_birth) values (**uuid\_generate\_v4(),** ‘John’, ‘Kang’, ‘Male’, ‘john@gmail.com’ ‘1990-01-01’, ‘Korea’);

….

If you want to update a person’s car status, then use UPDATE:

UPDATE person SET car\_uid = ‘[long uuid that’s been given, just copy and paste here’] WHERE person\_uid =’[peron’s uuid]’;

After update them

You can SELECT \* FROM person

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

Since they are using same uid, you can shorten the command by,

SELECT \* FROM person

JOIN car USING (car\_uid);

You can also use LEFT JOIN to see if other people’s info who don’t own a car also simply type LEFT left to JOIN command.