WHAT IS A DATABASE?

Database is a place where you can store, manipulate and retrieve data. So usually, this data is stored inside a computer server and then you can retrieve it, you can query the data, manipulate the data, CRUD Operations.

What is DBMS and RDBMS?

****

What is PostgreSQL?

Postgres is the actual database engine (Tool or software)

SQL – Structured Query Language

SQL – Is a programming Language

SQL – Manage data held in a relational database, Easy to learn, Very Powerful

How is data stored?

Stores data in Tables

Columns

Rows

SQL – It Allows to manage data in a Relational Database

What is Relational Database (RDBMS)?

It is simply a relation between one or more tables

PostgreSQL?

Object-relational database management system

Allows us to work with the relational databases

Modern

Open Source

Other Database Engine (Tool or Software)

MySQL, ORACLE, Microsoft SQL Server

Connecting to Database (DB) Server

- Connect using client
- GUI Client
- Terminal / Commands
- Application

Commands

cmd - psql

psql –version: Version POSTGRESQL

Help

\?

\l - List all databases

CREATE DATABASE database_name;

\c - Connect Command

\c dbname

Important Command – DROP DATABASE DBNAME; (Very Dangerous)

How to create a table with PostgreSQL?

Command - CREATE TABLE table_name (

Column_name + data_type + constraints if any);

Example: TABLE WITHOUT CONSTRAINTS

CREATE TABLE person (

id int,

first_name VARCHAR(50),

last_name VARCHAR(50),

gender VARCHAR(15),

date_of_birth TIMESTAMP,

);

VARCHAR(50) means just Characters, 50 Length.

Function TIMESTAMP includes Full Date, Hour, Minutes and Seconds.

Function DATE

\d - (Describe) List all the Tables present in the Database.

\d table_name – It Describes the table column names, datatypes etc.

Example: TABLE WITH CONSTRAINTS

CREATE TABLE person (

id BIGSERIAL NOT NULL PRIMARY KEY,

first_name VARCHAR(50) NOT NULL,

last_name VARCHAR(50) NOT NULL,

gender VARCHAR(10) NOT NULL,

date_of_birth DATE NOT NULL,

);

BIGSERIAL (bigint - "type")– SEQUENCE NEXTVAL (Auto Increment) (Useful)

In SQL DATE Function - 'Year - Month - Date'

How to INSERT Data / Records into a Table?

INSERT INTO person (

first_name,

last_name,

gender,

date_of_birth) VALUES ("MANOJ", "KUMAR", "MALE", "1999-05-23");

https://www.mockaroo.com/ - Random Data Generator – Can be downloaded in .csv, .json, .xml etc., formats.

For Inserting hundreds of records at a time in PostgreSQL the command is -

\i .sql_file_location;

Example:

\i /Users/Sraddha/Downloads/person.sql;

Command –

SELECT * FROM person(table_name);

*** - means Select every column in that table**

If you want to select only particular column then the command is

SELECT column_name1, column_name2, column_name3 FROM table_name;

ORDER BY KEYWORD:

We can sort our data by using "ORDER BY" Keyword in the SQL Query (Ascending Order) (Descending Order).

Command -

SELECT * FROM person(table_name) ORDER BY country_of_birth(column_name) LIMIT (for limiting the resultant records data) 25 – By default Ascending order;

If you want to implicitly sort the data, we can use ASC or DESC key words.

Works for both Dates, Numbers and Strings.

In case of NULL values If we ORDER BY DESC the resultant data will be in NULL values.

DISTINCT keyword:

The SELECT DISTINCT statement is used to return only distinct(different) values.

Command –

SELECT DISTINCT column_name FROM person ORDER BY country_of_birth LIMIT 20 (By default Ascending Order);

SELECT DISTINCT column_name FROM person ORDER BY country_of_birth DESC LIMIT 30 (Implicitly included "DESC" key word);

WHERE Clause Key Word:

WHERE clause allows us to filter the data based on conditions.

Example1:

SELECT * FROM person WHERE column_name="?";

Example2:

SELECT * FROM person WHERE gender = 'Female' ORDER BY first_name DESC LIMIT 15;

Example3: (Also add multiple columns in the WHERE clause condition using "AND OR" keywords)

SELECT * FROM person WHERE gender = 'Male' AND country_of_birth = 'Vietnam';

Example4:

SELECT * FROM person WHERE gender = 'Male' AND (country_of_birth = 'Russia' OR country_of_birth = 'China');

Example5:

SELECT * FROM person WHERE gender = 'Male' AND (country_of_birth = 'Russia' OR country_of_birth = 'China') AND last_name = 'Mishaw';

Example6:

SELECT * FROM person WHERE gender = 'Female' AND (country_of_birth = 'Russia' OR country_of_birth = 'China') AND last_name = 'Brewers';

COMPARISON Operators:

Comparison Operators allows us to perform Arithmetic Operations, Bitwise and other logical operations on the data.

Example:

SELECT 1=1;

Result: t (true)

SELECT 1=2;

Result: f (false)

=, <, >, <=, >= etc.,

Check weather a number not equal <>

Works with numbers as well as with Strings

Example:

SELECT ‘AMOGOS’ <> ‘amigos’;

We can filter down our data by using WHERE clause

IN keyword:

Example1:

SELECT * FROM person WHERE country_of_birth = 'China' OR country_of_birth = 'France' OR country_of_birth= 'Brazil';

Example2:

SELECT * FROM person WHERE country_of_birth IN ('China', 'France', 'Brazil', 'Vietnam');

Example3:

select * from person WHERE country_of_birth IN ('China', 'France', 'Nigeria', 'Indonesia', 'Sweden', 'Grenada') ORDER BY country_of_birth;

BETWEEN Keyword:

Use the BETWEEN keyword to select data from a certain range.

Example1:

That we want to select date of birth everyone in the range that was born between 2000 and 2015.

SELECT * FROM person WHERE date_of_birth BETWEEN DATE ‘2000-01-01’ AND ‘2017-01-01’;

SELECT * FROM person WHERE date_of_birth BETWEEN DATE '2000-01-01' AND '2023-01-01';

LIKE & ILIKE operator’s:

Like operator is used to match text values against a pattern using wildcards

Example1:

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

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

SELECT * FROM person WHERE email LIKE '%@yahoo.com';

Example2:

SELECT * FROM person WHERE email LIKE '%@yahoo.%';

```
id  | first_name | last_name  | gender | date_of_birth |          email          |
country_of_birth
-----+------------+------------+--------+---------------+-------------------------+------------------
178 | Abey      | Brailsford | Male   | 2023-04-20    | abrailsford4x@yahoo.com | Portugal
188 | Therine    | Bonhill    | Female | 2023-04-03    | tbonhill57@yahoo.co.jp  | Bulgaria
(2 rows)
```

Example3:

```
SELECT * FROM person WHERE email LIKE '________@%';

id  | first_name | last_name |  gender  | date_of_birth |          email          |
country_of_birth
-----+------------+-----------+----------+---------------+-------------------------+------------------
  3 | Irina      | Kayser    | Female   | 2023-09-09    | ikayser2@mac.com        | China
  8 | Eadie      | Genery    | Female   | 2022-11-27    | egenery7@epa.gov        | China
 10 | Derk       | Troker    | Male     | 2023-10-18    | dtroker9@nsw.gov.au     | Bulgaria
 13 | Chery      | Reisen    | Female   | 2023-01-12    | creisenc@netlog.com     | Colombia
 25 | Pinchas    | Lorans    | Male     | 2023-06-02    | ploranso@senate.gov     | China
 29 | Florida    | Loakes    | Female   | 2023-02-04    | floakess@g.co           | Indonesia
 44 | Beryl      | Evers     | Bigender | 2023-08-24    | bevers17@alexa.com      | South
Africa
 67 | Miles      | Faldo     | Bigender | 2023-05-24    | mfaldo1u@tumblr.com     | China
 75 | Eddy       | Tuvey     | Female   | 2023-08-22    | etuvey22@tuttocitta.it  | Indonesia
 81 | Hobie      | Crich     | Male     | 2023-01-12    | hcrich28@wikia.com      | Brazil
101 | Martita    | Wimms     | Female   | 2023-03-31    | mwimms2s@weibo.com      |
Poland
116 | Kizzee     | Landa     | Female   | 2023-03-28    | klanda37@live.com       | Indonesia
126 | Morrie     | Massy     | Male     | 2023-07-11    | mmassy3h@clickbank.net  | Russia
129 | Deny       | Swede     | Female   | 2023-01-15    | dswede3k@cam.ac.uk      | Norway
132 | Gonzalo    | Allon     | Male     | 2022-12-07    | gallon3n@feedburner.com | China
151 | Bellanca   | Sybbe     | Female   | 2023-01-09    | bsybbe46@gravatar.com   | Serbia
159 | Kati       | Beeby     | Female   | 2023-04-10    | kbeeby4e@odnoklassniki.ru |
Indonesia
184 | Emlen      | Tasch     | Male     | 2022-11-17    | etasch53@mysql.com      | Japan
186 | Berrie     | Corde     | Female   | 2022-12-15    | bcorde55@dmoz.org       | Uganda
(19 rows)
```

Example4:

```
SELECT * FROM person WHERE country_of_birth LIKE 'P%';

id  | first_name |  last_name  |   gender   | date_of_birth |          email          |
country_of_birth
-----+------------+-------------+------------+---------------+-------------------------+------------------
  2 | Joella     | Le Hucquet  | Female     | 2022-12-16    | jlehucquet1@spiegel.de  |
Philippines
  5 | Ranee      | Tuckett     | Female     | 2023-03-14    | rtuckett4@networkadvertising.org
| Philippines
  9 | Kathy      | Brewster    | Female     | 2023-03-07    |                         | Poland
 11 | Xavier     | Baigent     | Male       | 2023-01-25    | xbaigenta@timesonline.co.uk |
Philippines
 19 | Noni       | Cello       | Female     | 2022-12-24    |                         | Peru
 20 | Vevay      | Eisikowitch | Female     | 2023-10-19    | veisikowitchj@tinypic.com
| Portugal
 22 | Gawen      | Tapley      | Male       | 2023-06-17    |                         | Peru
 30 | Cosimo     | McCarlie    | Male       | 2023-09-08    | cmccarliet@odnoklassniki.ru |
Portugal
 33 | Kahlil     | Guichard    | Male       | 2023-02-01    |                         | Philippines
```

66 | Filberte | Burness | Male | 2023-02-08 | fburness1t@springer.com | Philippines

82 | Dollie | Vest | Female | 2023-05-17 | dvest29@nasa.gov | Poland

84 | Sasha | McGinnell | Non-binary | 2023-10-13 | | Poland

90 | Doreen | Kohn | Female | 2022-11-18 | dkohn2h@wikimedia.org | Philippines

91 | Linda | Revie | Female | 2023-07-18 | | Peru

93 | Kare | Larking | Female | 2022-12-05 | klarking2k@wordpress.org | Portugal

96 | Jamison | Farlam | Male | 2023-06-22 | jfarlam2n@army.mil | Poland

97 | Nathan | Secker | Male | 2023-02-02 | nsecker2o@cmu.edu | Philippines

101 | Martita | Wimms | Female | 2023-03-31 | mwimms2s@weibo.com | Poland

107 | Parker | Draper | Male | 2023-09-29 | pdraper2y@youku.com | Panama

122 | Del | Grey | Female | 2023-01-01 | dgrey3d@mozilla.com | Philippines

138 | Dwight | Dimbylow | Polygender | 2023-03-20 | | Philippines

141 | Ferguson | Clyburn | Male | 2023-09-03 | fclyburn3w@blogtalkradio.com | Portugal

146 | Durante | Abramchik | Male | 2023-01-24 | dabramchik41@toplist.cz | Philippines

160 | Kaiser | Osichev | Male | 2023-01-24 | kosichev4f@alexa.com | Philippines

169 | Jasmine | Battershall | Female | 2023-04-14 | jbattershall4o@dailymotion.com | Philippines

177 | Gaston | Shutte | Male | 2023-04-27 | gshutte4w@flavors.me | Poland

178 | Abey | Brailsford | Male | 2023-04-20 | abrailsford4x@yahoo.com | Portugal

-- More --

Example5:

SELECT * FROM person WHERE country_of_birth ILIKE 'p%';

id | first_name | last_name | gender | date_of_birth | email | country_of_birth

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

2 | Joella | Le Hucquet | Female | 2022-12-16 | jlehucquet1@spiegel.de | Philippines

5 | Ranee | Tuckett | Female | 2023-03-14 | rtuckett4@networkadvertising.org | Philippines

9 | Kathy | Brewster | Female | 2023-03-07 | | Poland

11 | Xavier | Baigent | Male | 2023-01-25 | xbaigenta@timesonline.co.uk | Philippines

19 | Noni | Cello | Female | 2022-12-24 | | Peru

20 | Vevay | Eisikowitch | Female | 2023-10-19 | veisikowitchj@tinypic.com | Portugal

22 | Gawen | Tapley | Male | 2023-06-17 | | Peru

30 | Cosimo | McCarlie | Male | 2023-09-08 | cmccarliet@odnoklassniki.ru | Portugal

33 | Kahlil | Guichard | Male | 2023-02-01 | | Philippines

66 | Filberte | Burness | Male | 2023-02-08 | fburness1t@springer.com | Philippines

82 | Dollie | Vest | Female | 2023-05-17 | dvest29@nasa.gov | Poland

84 | Sasha | McGinnell | Non-binary | 2023-10-13 | | Poland

90 | Doreen | Kohn | Female | 2022-11-18 | dkohn2h@wikimedia.org | Philippines

91 | Linda | Revie | Female | 2023-07-18 | | Peru

93 | Kare | Larking | Female | 2022-12-05 | klarking2k@wordpress.org | Portugal

96 | Jamison | Farlam | Male | 2023-06-22 | jfarlam2n@army.mil | Poland

97 | Nathan | Secker | Male | 2023-02-02 | nsecker2o@cmu.edu | Philippines

101 | Martita | Wimms | Female | 2023-03-31 | mwimms2s@weibo.com | Poland

107 | Parker | Draper | Male | 2023-09-29 | pdraper2y@youku.com | Panama

GROUP BY Keyword:

This is very powerful and basically allows us to group our data based on a column.

Example1:

SELECT country_of_birth, COUNT(*) FROM person GROUP BY country_of_birth;

country_of_birth | count

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

Bangladesh | 2

Indonesia | 25

Venezuela | 2

Uruguay | 1

Luxembourg | 1

Czech Republic | 3

Sweden | 4

Uganda | 1

Macedonia | 1

Example2:

SELECT country_of_birth, COUNT(*) FROM person GROUP BY country_of_birth ORDER BY country_of_birth;

country_of_birth | count

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

Afghanistan | 1

Aland Islands | 1

Argentina | 1

Armenia | 1

Azerbaijan | 1

Bahamas | 1

Bangladesh | 2

Belarus | 1

Brazil | 12

Bulgaria | 2

Canada | 2

China | 42

Colombia | 3

Comoros | 1

Costa Rica | 2

Croatia | 1

Czech Republic | 3

GROUP BY HAVING clause:

Having keyword works with GROUP BY. It allows us to perform an extra layer of filtering to the resulting data.

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

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

Example1: HAVING Keyword after GROUP BY before ORDER BY Keywords

```
SELECT country_of_birth, COUNT(*) FROM person GROUP BY country_of_birth
HAVING COUNT(*) > 20 ORDER BY country_of_birth;

country_of_birth | count

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

China            |   42

Indonesia        |   25

(2 rows)
```

MINIMUM, MAXIMUM, AVERAGE Functions:

To find out the most expensive car and its details like (MAX(), MIN(), AVG()) from the cardata table.

```
SELECT MIN(price) FROM cardata;

min

----------

10747.14

(1 row)
```

```
SELECT MAX(price) FROM cardata;

max

-----------

299825.29

(1 row)
```

```
SELECT AVG(price) FROM cardata;

avg

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

151764.193650000000

(1 row)
```

```
SELECT ROUND(AVG(price)) FROM cardata;

round

--------

151764

(1 row)
```

```
SELECT make, model, MAX(price) FROM cardata GROUP BY make, model;

   make     |        model         |    max

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

Pontiac      | GTO                  | 221674.79

Dodge        | Dynasty              | 287814.56

Toyota       | Land Cruiser         | 289161.30
```

```
Mercedes-Benz | S-Class          | 285021.81
Infiniti      | FX               | 224489.67
BMW           | M6               | 235888.64
Saturn        | Aura             | 115214.40
Kia           | Amanti           | 163552.27
Oldsmobile    | Silhouette       |  92220.94
Daewoo        | Lanos            | 240010.14
Subaru        | Forester         | 155727.62
Ford          | Econoline E350   | 272517.40
Ferrari       | 612 Scaglietti   | 100679.26
Jeep          | Grand Cherokee   | 190607.04
Hyundai       | Accent           | 289035.81
Isuzu         | Rodeo            | 271740.95
Mercedes-Benz | G55 AMG          |  86511.43
Infiniti      | G                | 291707.74
Buick         | Electra          | 121390.44
Nissan        | Xterra           | 238861.18
Volvo         | XC60             | 117587.76
Chevrolet     | Cavalier         | 296938.56
Isuzu         | Axiom            | 220119.85
Audi          | 4000             |  92867.32
GMC           | Rally Wagon G3500 | 140471.90
Audi          | A8               | 147458.65
Hyundai       | Sonata           | 207112.09
Chrysler      | Voyager          |  73570.51
Jaguar        | XJ               |  11528.55
Bentley       | Continental      | 231457.18
Ford          | Contour          | 132541.85
Mazda         | Mazda5           | 285553.20
Mazda         | Protege          |  20661.65
Lexus         | IS-F             | 298394.53
Chevrolet     | Corvair          |  27871.42
Nissan        | JUKE             |  78662.70
Dodge         | Magnum           |  72701.72
Dodge         | D350 Club        |  30471.94
Ford          | Ranger           |  18598.38
-- More  --
```

```
SELECT make, model, MIN(price) FROM cardata GROUP BY make, model;
    make      |        model        |    min
--------------+---------------------+----------
Pontiac       | GTO                 | 130390.53
Dodge         | Dynasty             | 287814.56
Toyota        | Land Cruiser        |  69339.70
Mercedes-Benz | S-Class             |  14841.21
Infiniti      | FX                  | 224489.67
BMW           | M6                  | 134724.89
Saturn        | Aura                | 115214.40
Kia           | Amanti              | 163552.27
Oldsmobile    | Silhouette          |  87636.95
```

```
Daewoo       | Lanos                | 240010.14
Subaru       | Forester             |  62153.31
Ford         | Econoline E350       | 272517.40
Ferrari      | 612 Scaglietti       | 100679.26
Jeep         | Grand Cherokee       |  85269.26
Hyundai      | Accent               | 192908.06
Isuzu        | Rodeo                | 259570.42
Mercedes-Benz | G55 AMG             |  86511.43
Infiniti     | G                    |  19344.62
Buick        | Electra              | 121390.44
Nissan       | Xterra               | 222797.64
Volvo        | XC60                 | 117587.76
Chevrolet    | Cavalier             |  69665.97
Isuzu        | Axiom                | 220119.85
Audi         | 4000                 |  92867.32
GMC          | Rally Wagon G3500    | 140471.90
Audi         | A8                   |  88365.64
Hyundai      | Sonata               |  51104.61
-- More  --
```

```
SELECT make,MIN(price) FROM cardata GROUP BY make;
    make      |    min
---------------+-----------
GMC          |  17901.66
Maybach      |  58505.85
Lincoln      |  33946.95
Honda        |  17554.79
Daewoo       | 240010.14
Spyker       | 200922.92
Morgan       |  28806.35
Ford         |  14020.67
Scion        |  16898.41
Maserati     |  63583.20
Dodge        |  14180.01
Chevrolet    |  14933.55
Saturn       |  31450.72
Infiniti     |  19344.62
MINI         |  28200.00
Bentley      |  19200.31
Austin       | 183409.93
Peugeot      |  94540.42
Pontiac      |  10997.28
Porsche      |  28148.12
Plymouth     |  20656.43
Audi         |  11233.39
Rolls-Royce  |  39515.14
Jaguar       |  11528.55
Lexus        |  40949.49
```

```
Lotus        |  87916.21
Kia          |  10967.50
-- More  --

SELECT make, MAX(price) FROM cardata GROUP BY make;
    make      |    max
---------------+-----------
GMC          | 299825.29
Maybach      |  96522.53
Lincoln      | 297728.50
Honda        | 297343.31
Daewoo       | 240010.14
Spyker       | 200922.92
Morgan       |  28806.35
Ford         | 298521.67
Scion        |  82544.96
Maserati     | 281628.05
Dodge        | 287814.56
Chevrolet    | 299464.54
Saturn       | 224406.17
Infiniti     | 291707.74
MINI         | 162203.65
Bentley      | 231457.18
Austin       | 183409.93
Peugeot      | 282192.18
Pontiac      | 296828.26
Porsche      | 295854.26
Plymouth     | 235433.12
Audi         | 280302.48
Rolls-Royce  |  39515.14
Jaguar       | 243367.65
Lexus        | 298394.53
Lotus        | 251496.91
Kia          | 291349.49
-- More  --

SELECT make, ROUND(AVG(price)) FROM cardata GROUP BY make LIMIT 20;
   make    | round
-----------+--------
GMC        | 138215
Maybach    |  80808
Lincoln    | 169016
Honda      | 166504
Daewoo     | 240010
Spyker     | 200923
Morgan     |  28806
Ford       | 140398
Scion      |  49722
Maserati   | 169668
Dodge      | 152909
```

```
Chevrolet | 148359
Saturn    | 142672
Infiniti  | 135598
MINI      | 107211
Bentley   | 101814
Austin    | 183410
Peugeot   | 188366
Pontiac   | 139565
Porsche   | 172529
(20 rows)
```

SUM Function:

```
SELECT SUM(price) FROM cardata;
    sum
--------------
151764193.65
(1 row)
```

```
SELECT ROUND(SUM(price)) FROM cardata;
  round
-----------
151764194
(1 row)
```

```
SELECT make, ROUND(SUM(price)) FROM cardata GROUP BY make LIMIT 25;
   make     |  round
-------------+---------
GMC         | 7048963
Maybach     |  323232
Lincoln     | 2197202
Honda       | 4495597
Daewoo      |  240010
Spyker      |  200923
Morgan      |   28806
Ford        | 9687475
Scion       |   99443
Maserati    |  848342
Dodge       | 8257110
Chevrolet   | 9346646
Saturn      |  998703
Infiniti    | 2576367
MINI        |  321633
```

ARITHMETIC OPERATORS

Addition, Subtraction, Multiplication, Division (/ - Quotient), Modulus (% - Remainder), Power (^), Factorial (!)

Query1: Company wants to provide an additional discount of 15% of the actual price. Display the Actual Price and Discounted Prices in a Tabular column format.

Solution:

```
SELECT id, make, model, price, price * .15 FROM cardata LIMIT 25;

id |     make      |     model      |   price   |  ?column?

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

 1 | Dodge         | Ram 2500       |  38275.58 |  5741.3370

 2 | Bentley       | Brooklands     |  98237.39 | 14735.6085

 3 | BMW           | M6             | 235888.64 | 35383.2960

 4 | Porsche       | 944            | 204289.67 | 30643.4505

 5 | Chrysler      | Town & Country |  91877.97 | 13781.6955

 6 | Cadillac      | Fleetwood      | 190007.05 | 28501.0575

 7 | Lexus         | SC             | 103267.24 | 15490.0860

 8 | GMC           | Suburban 2500  |  84615.78 | 12692.3670

 9 | Mercury       | Mountaineer    | 227273.11 | 34090.9665

10 | Porsche       | 928            | 167586.94 | 25138.0410

11 | Infiniti      | QX56           | 249539.44 | 37430.9160

12 | Spyker        | C8             | 200922.92 | 30138.4380

13 | Mercedes-Benz | R-Class        | 299313.30 | 44896.9950

14 | GMC           | Yukon          | 237408.01 | 35611.2015

15 | BMW           | 3 Series       |  98311.89 | 14746.7835
```

```
SELECT id, make, model, price, ROUND(price * .15) FROM cardata LIMIT 25;

id |     make      |     model      |   price   | round

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

 1 | Dodge         | Ram 2500       |  38275.58 |  5741

 2 | Bentley       | Brooklands     |  98237.39 | 14736

 3 | BMW           | M6             | 235888.64 | 35383

 4 | Porsche       | 944            | 204289.67 | 30643

 5 | Chrysler      | Town & Country |  91877.97 | 13782

 6 | Cadillac      | Fleetwood      | 190007.05 | 28501

 7 | Lexus         | SC             | 103267.24 | 15490

 8 | GMC           | Suburban 2500  |  84615.78 | 12692

 9 | Mercury       | Mountaineer    | 227273.11 | 34091

10 | Porsche       | 928            | 167586.94 | 25138

11 | Infiniti      | QX56           | 249539.44 | 37431

12 | Spyker        | C8             | 200922.92 | 30138

13 | Mercedes-Benz | R-Class        | 299313.30 | 44897

14 | GMC           | Yukon          | 237408.01 | 35611

15 | BMW           | 3 Series       |  98311.89 | 14747
```

```
SELECT id, make, model, price, ROUND(price * .15, 2) FROM cardata LIMIT 25;

id |     make      |     model      |   price   |  round

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

 1 | Dodge         | Ram 2500       |  38275.58 |  5741.34

 2 | Bentley       | Brooklands     |  98237.39 | 14735.61
```

```
 3 | BMW          | M6            | 235888.64 | 35383.30

 4 | Porsche      | 944           | 204289.67 | 30643.45

 5 | Chrysler     | Town & Country |  91877.97 | 13781.70

 6 | Cadillac     | Fleetwood     | 190007.05 | 28501.06

 7 | Lexus        | SC            | 103267.24 | 15490.09

 8 | GMC          | Suburban 2500  |  84615.78 | 12692.37

 9 | Mercury      | Mountaineer   | 227273.11 | 34090.97

10 | Porsche      | 928           | 167586.94 | 25138.04

11 | Infiniti     | QX56          | 249539.44 | 37430.92

12 | Spyker       | C8            | 200922.92 | 30138.44

13 | Mercedes-Benz | R-Class       | 299313.30 | 44897.00

14 | GMC          | Yukon         | 237408.01 | 35611.20

15 | BMW          | 3 Series      |  98311.89 | 14746.78
```

SELECT id, make, model, price, ROUND(price * .15, 2), ROUND(price - (price * .15))
FROM cardata LIMIT 25;

```
id |     make      |     model      |   price   |  round   | round

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

 1 | Dodge        | Ram 2500       |  38275.58 |  5741.34 |  32534

 2 | Bentley      | Brooklands     |  98237.39 | 14735.61 |  83502

 3 | BMW          | M6            | 235888.64 | 35383.30 | 200505

 4 | Porsche      | 944           | 204289.67 | 30643.45 | 173646

 5 | Chrysler     | Town & Country |  91877.97 | 13781.70 |  78096

 6 | Cadillac     | Fleetwood     | 190007.05 | 28501.06 | 161506

 7 | Lexus        | SC            | 103267.24 | 15490.09 |  87777

 8 | GMC          | Suburban 2500  |  84615.78 | 12692.37 |  71923

 9 | Mercury      | Mountaineer   | 227273.11 | 34090.97 | 193182

10 | Porsche      | 928           | 167586.94 | 25138.04 | 142449

11 | Infiniti     | QX56          | 249539.44 | 37430.92 | 212109

12 | Spyker       | C8            | 200922.92 | 30138.44 | 170784

13 | Mercedes-Benz | R-Class       | 299313.30 | 44897.00 | 254416

14 | GMC          | Yukon         | 237408.01 | 35611.20 | 201797

15 | BMW          | 3 Series      |  98311.89 | 14746.78 |  83565
```

SELECT id, make, model, price, ROUND(price * .15, 2), ROUND(price - (price * .15), 2)
FROM cardata LIMIT 25;

```
id |     make      |     model      |   price   |  round   |  round

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

 1 | Dodge        | Ram 2500       |  38275.58 |  5741.34 |  32534.24

 2 | Bentley      | Brooklands     |  98237.39 | 14735.61 |  83501.78

 3 | BMW          | M6            | 235888.64 | 35383.30 | 200505.34

 4 | Porsche      | 944           | 204289.67 | 30643.45 | 173646.22

 5 | Chrysler     | Town & Country |  91877.97 | 13781.70 |  78096.27

 6 | Cadillac     | Fleetwood     | 190007.05 | 28501.06 | 161505.99

 7 | Lexus        | SC            | 103267.24 | 15490.09 |  87777.15

 8 | GMC          | Suburban 2500  |  84615.78 | 12692.37 |  71923.41

 9 | Mercury      | Mountaineer   | 227273.11 | 34090.97 | 193182.14

10 | Porsche      | 928           | 167586.94 | 25138.04 | 142448.90
```

ALIAS Keyword:

We can use "ALIAS (AS)" keyword for overriding any column.

SELECT id, make, model, price AS actual_price, ROUND(price * .15, 2) AS fifteen_percentage_discount, ROUND(price - (price * .15), 2) AS discounted_price FROM cardata LIMIT 25;

```
id |     make      |     model      | actual_price | fifteen_percentage_discount |
discounted_price
----+---------------+----------------+--------------+-----------------------------+-----------------
 1 | Dodge         | Ram 2500       |    38275.58 |                 5741.34 |        32534.24
 2 | Bentley       | Brooklands     |    98237.39 |                14735.61 |        83501.78
 3 | BMW           | M6             |   235888.64 |                35383.30 |       200505.34
 4 | Porsche       | 944            |   204289.67 |                30643.45 |       173646.22
 5 | Chrysler      | Town & Country |    91877.97 |                13781.70 |        78096.27
 6 | Cadillac      | Fleetwood      |   190007.05 |                28501.06 |       161505.99
 7 | Lexus         | SC             |   103267.24 |                15490.09 |        87777.15
 8 | GMC           | Suburban 2500  |    84615.78 |                12692.37 |        71923.41
 9 | Mercury       | Mountaineer    |   227273.11 |                34090.97 |       193182.14
10 | Porsche       | 928            |   167586.94 |                25138.04 |       142448.90
11 | Infiniti      | QX56           |   249539.44 |                37430.92 |       212108.52
12 | Spyker        | C8             |   200922.92 |                30138.44 |       170784.48
13 | Mercedes-Benz | R-Class        |   299313.30 |                44897.00 |       254416.31
14 | GMC           | Yukon          |   237408.01 |                35611.20 |       201796.81
15 | BMW           | 3 Series       |    98311.89 |                14746.78 |        83565.11
```

COALESCE Keyword:

`COALESCE` is a function in PostgreSQL that returns the first non-null argument. It is often used with the `SELECT` statement to handle null values effectively. The syntax of the `COALESCE` function is as follows: `COALESCE (argument_1, argument_2, …)`. It accepts an unlimited number of arguments and returns to the first argument that is not null. If all arguments are null, the `COALESCE` function will return null. The `COALESCE` function evaluates arguments from left to right until it finds the first non-null argument. All the remaining arguments from the first non-null argument are not evaluated. The `COALESCE` function provides the same functionality as `NVL` or `IFNULL` function provided by SQL-standard.

Here is an example of how to use the `COALESCE` function in PostgreSQL. Suppose we have a table named `items` with four fields: `id`, `product`, `price`, and `discount`. We want to query the net prices of the products using the following formula: `net_price = price - discount`. If the discount is null, it is assumed to be zero. We can use the `COALESCE` function to substitute a default value for null values when querying the data as follows:

```
SELECT product, (price - COALESCE(discount, 0)) AS net_price FROM items;
```

```
Query1: From the dataset "person" we want to select every single email and
for those people that don't have an email we simply want to have an email
with the value of "email not provided".
```

```
Solution:
```

```
SELECT first_name, COALESCE(email, 'Email not Provided') FROM person LIMIT
35;
first_name |              coalesce
------------+-----------------------------------
Genovera   | Email not Provided
Joella     | jlehucquet1@spiegel.de
Irina      | ikayser2@mac.com
Westbrooke | Email not Provided
Ranee      | rtuckett4@networkadvertising.org
Leigha     | Email not Provided
Lars       | Email not Provided
Eadie      | egenery7@epa.gov
Kathy      | Email not Provided
Derk       | dtroker9@nsw.gov.au
Xavier     | xbaigenta@timesonline.co.uk
Rubin      | rrathboneb@whitehouse.gov
```

```
Chery      | creisenc@netlog.com
Karlan     | kplankd@google.com.au
Elora      | ekitchine@google.ca
Cori       | cturrellf@earthlink.net
Hermy      | hmatelyunasg@opensource.org
Bernadine  | bselesnickh@bloomberg.com
Noni       | Email not Provided
Vevay      | veisikowitchj@tinypic.com
Poppy      | pjoubertk@wordpress.com
```

NULLIF Keyword:

NULLIF is a built-in function in **PostgreSQL** that returns a null value if the first argument equals the second argument, otherwise it returns the first argument. Here is the syntax of the NULLIF function:

```
NULLIF (argument_1, argument_2);
```

The NULLIF function is commonly used to handle null values in PostgreSQL. For example, to substitute a null value in a column with a default value, you can use the COALESCE function along with the NULLIF function.

Here is an example:

```
SELECT COALESCE(NULLIF(column_name, ''), 'default_value') FROM table_name;
```

TIMESTAMPS AND DATES COURSE

```
SELECT NOW();
now
-----------------------------------
2023-12-15 20:36:35.336216+05:30

SELECT NOW() AS present_time_date_stamp;
present_time_date_stamp
-----------------------------------
2023-12-15 20:37:40.259381+05:30
```

Important: - YYYY-MM-DD HH:MM:Sec.MilliSec+timezone

```
SELECT NOW()::DATE AS present_date;
present_date
--------------
2023-12-15

SELECT NOW()::TIME AS present_time;
present_time
-----------------
20:42:43.189978

SHOW TIMEZONE;
TimeZone
---------------
Asia/Calcutta
```

ADDING AND SUBSTRACTING WITH DATES

Subtract 1Year from the present time

```
SELECT NOW() - INTERVAL '1 YEAR' AS present_year;
           present_year
-----------------------------------
2023-03-25 20:21:42.942371+05:30
(1 row)
```

```
SELECT NOW() - INTERVAL '10 MONTHS' AS present;
             present
----------------------------------
2023-05-25 20:22:40.299507+05:30
(1 row)
```

ADDING DATES

```
SELECT NOW() + INTERVAL '1 MONTH' AS future_time;
           future_time
----------------------------------
2024-04-25 20:24:11.335232+05:30
(1 row)
```

FOR CASTING ONLY DATE WITH TIMESTAMP

```
SELECT NOW()::DATE + INTERVAL '1 MONTH' AS future_time;
     future_time
---------------------
2024-04-25 00:00:00
(1 row)
```

FOR CASTING DATE WITHOUT TIMESTAMP

```
SELECT (NOW() + INTERVAL '1 MONTH')::DATE;
    date
------------
2024-04-25
(1 row)

SELECT (NOW() + INTERVAL '2 MONTHS')::DATE AS future_date;
future_date
-------------
2024-05-25
(1 row)
```

EXTRACTING FIELDS FROM TIMESTAMP

Extract only the Year from the Timestamp

```
SELECT EXTRACT(YEAR FROM NOW()) AS present_year;
present_year
--------------
         2024
(1 row)
```

Extract only the month from the Timestamp

```
SELECT EXTRACT(MONTH FROM NOW()) AS present_month;
present_month
---------------
             3
(1 row)
```

Extract only the Day from the Timestamp

```
SELECT EXTRACT(DAY FROM NOW()) AS present_day;
present_day
-------------
          25
(1 row)
```

Extract only the Day of the Week from the Timestamp

```
SELECT EXTRACT(DOW FROM NOW()) AS present_day_of_the_week;
present_day_of_the_week
-------------------------
                       1
(1 row)
```

Extract only the Century from the Timestamp

```
SELECT EXTRACT(CENTURY FROM NOW()) AS present_century;

present_century
-----------------
              21
(1 row)
```

AGE FUNCTION

Command for Using AGE() Function

```
SELECT first_name, last_name, gender, date_of_birth, country_of_birth,
AGE(NOW(), date_of_birth) AS age FROM person LIMIT 40;

first_name |  last_name  |   gender    | date_of_birth | country_of_birth
|                  age

Genovera   | Gladwin     | Genderqueer | 2023-05-13    | France
| 10 mons 12 days 21:10:56.419167

Joella     | Le Hucquet  | Female      | 2022-12-16    | Philippines
| 1 year 3 mons 9 days 21:10:56.419167

Irina      | Kayser      | Female      | 2023-09-09    | China
| 6 mons 16 days 21:10:56.419167

Westbrooke | Ivanichev   | Male        | 2023-02-26    | Colombia
| 1 year 27 days 21:10:56.419167

Ranee      | Tuckett     | Female      | 2023-03-14    | Philippines
| 1 year 11 days 21:10:56.419167

Leigha     | MacDavitt   | Female      | 2023-10-14    | Indonesia
| 5 mons 11 days 21:10:56.419167

Lars       | Napoleon    | Male        | 2023-03-11    | Russia
| 1 year 14 days 21:10:56.419167

Eadie      | Genery      | Female      | 2022-11-27    | China
| 1 year 3 mons 28 days 21:10:56.419167

Kathy      | Brewster    | Female      | 2023-03-07    | Poland
| 1 year 18 days 21:10:56.419167

Derk       | Troker      | Male        | 2023-10-18    | Bulgaria
| 5 mons 7 days 21:10:56.419167

Xavier     | Baigent     | Male        | 2023-01-25    | Philippines
| 1 year 2 mons 21:10:56.419167

Rubin      | Rathbone    | Male        | 2023-01-31    | China
| 1 year 1 mon 25 days 21:10:56.419167

Chery      | Reisen      | Female      | 2023-01-12    | Colombia
| 1 year 2 mons 13 days 21:10:56.419167

Karlan     | Plank       | Male        | 2023-01-25    | China
| 1 year 2 mons 21:10:56.419167

Elora      | Kitchin     | Female      | 2023-04-09    | Luxembourg
| 11 mons 16 days 21:10:56.419167

Cori       | Turrell     | Male        | 2023-07-07    | Nicaragua
| 8 mons 18 days 21:10:56.419167

Hermy      | Matelyunas  | Male        | 2023-09-01    | China
| 6 mons 24 days 21:10:56.419167

Bernadine  | Selesnick   | Non-binary  | 2023-09-23    | South Korea
| 6 mons 2 days 21:10:56.419167

Noni       | Cello       | Female      | 2022-12-24    | Peru
| 1 year 3 mons 1 day 21:10:56.419167 ...... etc., (40 rows)
```

PRIMARY KEYS

A PRIMARY KEY IS A VALUE IN THE COLUMN WHICH UNIQUELY IDENTIFIES A RECORD IN ANY TABLE. Example: - BIGSERIAL Keyword

If you try to insert the record with the same id that already exists in the table, I'll provide us this error: -

```
insert into person (id, first_name, last_name, gender, date_of_birth,
email, country_of_birth) values (1,'Genovera', 'Gladwin', 'Genderqueer',
'2023-05-13', null, 'France');

ERROR:  duplicate key value violates unique constraint "person_pkey"

DETAIL:  Key (id)=(1) already exists.
```

To drop the table Unique Primary Key Constraint

```
ALTER TABLE person DROP CONSTRAINT person_pkey;
```

If the Unique Primary Key Constraint is not available for a table: -

This is the result for the Query.

```
SELECT * FROM person WHERE id=1;

id | first_name | last_name |   gender    | date_of_birth | email |
country_of_birth
```

```
----+------------+-----------+-------------+---------------+-------+------
------------

  1 | Genovera   | Gladwin   | Genderqueer | 2023-05-13    |       |
France

  1 | Genovera   | Gladwin   | Genderqueer | 2023-05-13    |       |
France

(2 rows)
```

We can add any number of records with the same “id” provided.

UNIQUE CONTRAINTS

Unique Constraints allow us to have unique values for a given column.

```
SELECT email, count(*) FROM person GROUP BY email LIMIT 20;

              email               | count

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

hmatelyunasg@opensource.org      |     1

etuvey22@tuttocitta.it           |     1

rbenjamin4y@nyu.edu              |     1

rtuckett4@networkadvertising.org |     1

rswanger3z@wikispaces.com        |     1

                                 |    58

ikayser2@mac.com                 |     1

..... more


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

email | count

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

       |    58

(1 row)
```

```
insert into person (first_name, last_name, gender, date_of_birth, email,
country_of_birth) values ('Andrei', 'Le Hucquet', 'Female', '2022-12-16',
'jlehucquet1@spiegel.de', 'Philippines');

INSERT 0 1

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

         email          | count

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

                        |    58

jlehucquet1@spiegel.de |     2

(2 rows)


ALTER TABLE person ADD CONSTRAINT unique_email_address UNIQUE(email);

ERROR:  could not create unique index "unique_email_address"

DETAIL:  Key (email)=(jlehucquet1@spiegel.de) is duplicated.


ALTER TABLE person ADD CONSTRAINT unique_email_address UNIQUE(email);

test=# \d person;

                      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(50)  |           | not null |

date_of_birth     | date                   |           | not null |

email             | character varying(150) |           |          |

country_of_birth | character varying(50)  |           | not null |

Indexes:

    "person_pkey" PRIMARY KEY, btree (id)

    "unique_email_address" UNIQUE CONSTRAINT, btree (email)
```

```
insert into person (first_name, last_name, gender, date_of_birth, email,
country_of_birth) values ('Irina', 'Kayser', 'Female', '2023-09-09',
'ikayser2@mac.com', 'China');

ERROR:  duplicate key value violates unique constraint
"unique_email_address"

DETAIL: Key (email)=(ikayser2@mac.com) already exists.


ALTER TABLE person ADD UNIQUE(email);

test=# \d person;

                      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(50)  |           | not null |

date_of_birth     | date                   |           | not null |

email             | character varying(150) |           |          |

country_of_birth | character varying(50)  |           | not null |

Indexes:

    "person_pkey" PRIMARY KEY, btree (id)

    "person_email_key" UNIQUE CONSTRAINT, btree (email)
```

CHECK CONSTRAINTS

The Check constraints allows us to add a constraint based on a Boolean condition. So, the actual check constraint allows us to do is to make sure that we can only add a string which matches either male or female.

Check Constraint Command: -

```
ALTER TABLE person ADD CONSTRAINT gender_constraint CHECK (gender =
'Female' OR gender = 'Male');

Check constraints:

    "gender_constraint" CHECK (gender::text = 'Female'::text OR
gender::text = 'Male'::text)

test=#

insert into person (first_name, last_name, gender, date_of_birth, email,
country_of_birth) values ('Bernadine', 'Selesnick', 'Non-binary', '2023-
09-23', 'bselesnickh@bloomberg.com', 'South Korea');



ERROR:

new row for relation "person" violates check constraint
"gender_constraint"

DETAIL:

Failing row contains (208, Bernadine, Selesnick, Non-binary, 2023-09-23,
bselesnickh@bloomberg.com, South Korea).

CHECK CONSTRAINTS ARE POWERFUL.
```

DELETE OPERATION

If you want to delete any record, use the “PRIMARY KEY” and a “WHERE” Clause. If the DELETE statement is executed without a WHERE clause the whole database will be deleted.

UPDATE OPERATION

The update command allows us to Update a column, or multiple columns based on our WHERE clause.

Use the UPDATE Command with the SET and WHERE clauses or you might update or modify your entire table.

```
UPDATE person SET email = 'boxhall.holmes@yahoo.com' WHERE id = 49;
```

To Update multiple columns at a single time: -

```
UPDATE person SET email = 'kumar.manojbh@yahoo.com', first_name='Manoj',
last_name='Kumar BH' WHERE id = 49;

SELECT * FROM person WHERE first_name = 'Manoj';

id | first_name | last_name | gender | date_of_birth |          email
| country_of_birth

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

49 | Manoj      | Kumar BH  | Male   | 2022-12-23    |
kumar.manojbh@yahoo.com | Macedonia
```

ON CONFLIT DO NOTHING

How to deal with the duplicate key errors or Exceptions?

SELECT * FROM person WHERE id BETWEEN 120 AND 150;

ON CONFLICT (actual_column) DO NOTHING. that might be in conflict.

Use ON CONFLICT () only on the Column having UNIQUE CONSTRAINT or PRIMARY KEY CONSTRAINT.

Can have ON CONFLICT () on multiple columns as well.

UPSERT (DO CONFLICT ON UPDATE OPERATION)

UPDATE + INSERT (Override Existing data)

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

INSERT INTO person (id, first_name, last_name, gender, date_of_birth, email, country_of_birth) VALUES(150, 'Mitchel', 'Lowten', 'Male', DATE '2023-03-05', 'mlowten45@nhs.gov.uk', 'Egypt') ON CONFLICT (id) DO UPDATE SET email = EXCLUDED.email;

INSERT 0 1

test=# SELECT * FROM person WHERE id = 150;

```
id  | first_name | last_name | gender | date_of_birth |        email
| country_of_birth
-----+------------+-----------+--------+---------------+------------------
----+------------------
150 | Mitchel    | Lowten    | Male   | 2023-03-05    |
mlowten45@nhs.gov.uk | Egypt
(1 row)
```

We can pretty much update every single value.

INSERT INTO person (id, first_name, last_name, gender, date_of_birth, email, country_of_birth) VALUES(150, 'Manoj Jagan Reddy', 'Kumar Bhimavarapu', 'Male', DATE '2023-03-05', 'mlowten45@dell.com', 'Egypt') ON CONFLICT (id) DO UPDATE SET email = EXCLUDED.email, first_name = EXCLUDED.first_name, last_name = EXCLUDED.last_name;

INSERT 0 1

test=# SELECT * FROM person WHERE id = 150;

```
id  |    first_name     |     last_name     | gender | date_of_birth |
email        | country_of_birth
-----+-------------------+-------------------+--------+---------------+---
-----------------+------------------
150 | Manoj Jagan Reddy | Kumar Bhimavarapu | Male   | 2023-03-05    |
mlowten45@dell.com | Egypt
(1 row)
```

FOREIGN KEYS AND JOINS

Foreign Key – A Foreign Key is a column that references a primary key in another table.

INNER JOINS

The Inner Join takes whatever common in both tables (A + B = C) - Common.

BIGSERIAL & SEQUENCE

SELECT * FROM person_id_seq;

SELECT * FROM person_id_seq;

```
last_value | log_cnt | is_called
------------+---------+-----------
        211 |      30 | t
(1 row)
```

SELECT nextval('person_id_seq'::regclass);

```
nextval
---------
     212
(1 row)
```

If continued to execute, the query resultant nextval is 213

SEQUENCES is basically a BIGINT or BIGZERO.

We can RESTART the Sequence Value.

ALTER SEQUENCE person_id_seq RESTART WITH 10;

```
test=# ALTER SEQUENCE person_id_seq RESTART WITH 212;

ALTER SEQUENCE

test=# SELECT * FROM person_id_seq;

last_value | log_cnt | is_called

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

        212 |       0 | f

(1 row)
```

EXPORTING QUERY RESULTS TO CSV

```
SELECT * FROM person

LEFT JOIN cardata ON cardata.id = person.id;

id  |    first_name     |     last_name     | gender | date_of_birth |
email                | country_of_birth | id  |     make      |
model       |   price

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

   2 | Joella            | Le Hucquet        | Female | 2022-12-16    |
jlehucquet1@spiegel.de             | Philippines      |   2 | Bentley
| Brooklands        |  98237.39

To copy the Query results to a client host, use \copy command.

test=# \copy (SELECT * FROM person LEFT JOIN cardata ON cardata.id =
person.id) TO '/Users/Sraddha/Downloads/results_query.csv' DELIMITER ','
CSV HEADER;

COPY 185
```

UNDERSTANDING UUID DATA TYPES

```
Universally Unique Identifier Extension

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE EXTENSION

uuid-ossp            | 1.1             | 1.1               | generate
universally unique identifiers (UUIDs)

To use the UUID-OSSP Extension Built-in Functions.



\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)


SELECT uuid_generate_v4();

           uuid_generate_v4

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

f79a93b6-e057-45db-b3d6-5e0ec89b5ef2

(1 row)
```

This would be Unique, every time you invoke the function.

Make use of UUID as Primary Key’s in our Tables.

uuid_generate_v1(): This function generates a UUID using a combination of the current time and the MAC address of the computer executing the function.

uuid_generate_v4(): This function generates a random UUID.

INSERT INTO my_table (id, name) VALUES (uuid_generate_v4(), 'My Name');

Inserting a record into the students' table.

CREATE TABLE students (student_uid UUID NOT NULL PRIMARY KEY, name VARCHAR(100) NOT NULL);

CREATE TABLE

INSERT INTO students (student_uid, name) VALUES(uuid_generate_v4(), 'Manoj Kumar BH');

INSERT 0 1

SELECT * FROM students;

student_uid | name

---+---

511c25ec-c98d-4dfc-bd1c-74cad6ac745f | Manoj Kumar BH

(1 row)

Describe students' table.

\d students;

Table "public.students"

Column | Type | Collation | Nullable | Default

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

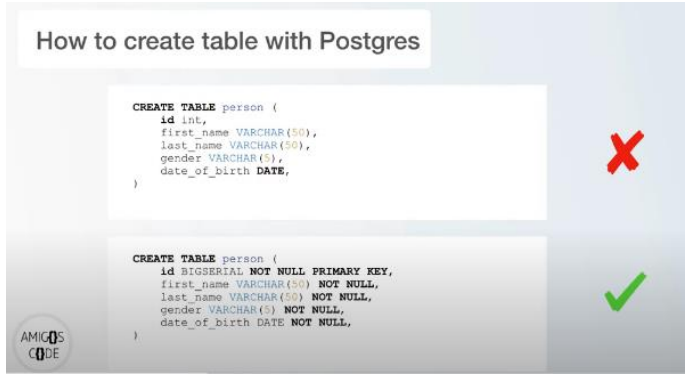
student_uid | uuid | | not null |

name | character varying(100) | | not null |

Indexes:

"students_pkey" PRIMARY KEY, btree (student_uid)

```
test=# \i /Users/Sraddha/Downloads/person.sql;
CREATE TABLE
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
```




```
Connection options:
  -h, --host=HOSTNAME      database server host or socket directory (default: "local socket")
  -p, --port=PORT          database server port (default: "5432")
  -U, --username=USERNAME  database user name (default: "amigoscode")
  -w, --no-password        never prompt for password
  -W, --password           force password prompt (should happen automatically)
```

```
➜ ~ psql -h localhost -p 5432 -U amigoscode test
psql (11.0)
Type "help" for help.

test=# \q
➜ ~ psql -h localhost -p 5432 -U amigoscode test1
psql: FATAL:  database "test1" does not exist
```

```
psql is the PostgreSQL interactive terminal.

Usage:
  psql [OPTION]... [DBNAME [USERNAME]]

General options:
  -c, --command=COMMAND    run only single command (SQL or internal) and exit
  -d, --dbname=DBNAME      database name to connect to (default: "amigoscode")
  -f, --file=FILENAME      execute commands from file, then exit
  -l, --list               list available databases, then exit
  -v, --set=, --variable=NAME=VALUE
                           set psql variable NAME to VALUE
                           (e.g., -v ON_ERROR_STOP=1)
  -V, --version            output version information, then exit
  -X, --no-psqlrc          do not read startup file (~/.psqlrc)
  -1 ("one"), --single-transaction
                           execute as a single transaction (if non-interactive)
  -?, --help[=options]     show this help, then exit
      --help=commands      list backslash commands, then exit
      --help=variables     list special variables, then exit
```