   ii. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.

d. Durability
   i. ensures that once a transaction is committed, its changes are permanent and will survive any subsequent system failures.

How is PostgreSQL ACID?

a. PostgreSQL ensures atomicity by either fully completing or failing transactions.
b. PostgreSQL ensures consistency because your database rolls back all changes when the transaction fails in order to keep data consistent.
c. PostgreSQL ensures isolation because transactions are kept separated from each other and ensures that transactions previously are completed before the next transaction goes through.
d. PostgreSQL ensures durability because the database logs changes and you can recover the database from a previous state in case your server fails.

Day17

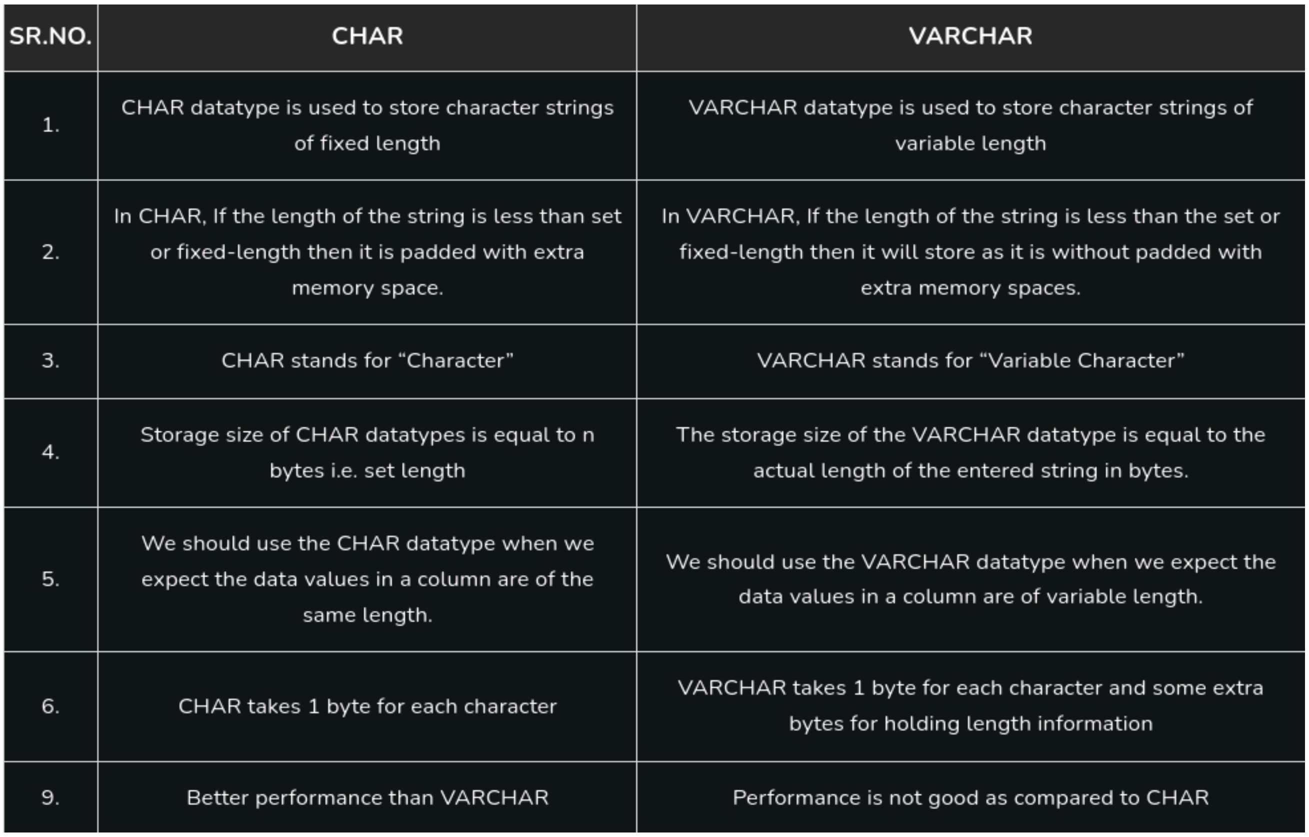
Related Researches

Int vs bigint

a. Int is 32 bit long (4 bytes), but bigint is 64 bit long (8 bytes)
b. Int can be used for small counters, small numeric values, etc where value falls in the range. But, bigint can be used for financial data, scientific calculations, etc
c. bigint uses more memory than int, which can impact performance if large numbers of these values are stored or processed.
d. Operations and processing speed for bigint can be slower than int.

Char vs varchar

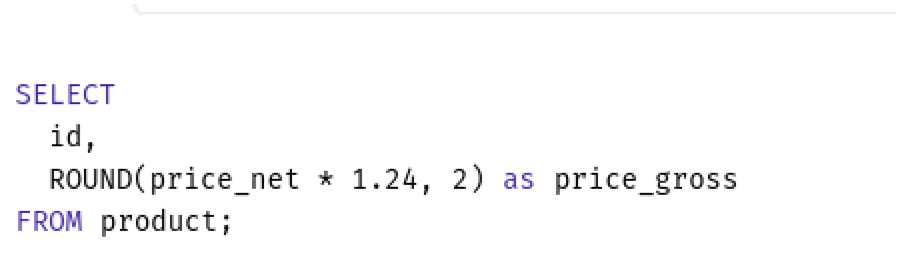
a. Char has better performance than varchar because the database doesn't need to calculate the length.
b. **char**: Fixed-length, space-padded, more storage for shorter data.
c. **varchar**: Variable-length, no padding, efficient for varying lengths.



SR.NO.	CHAR	VARCHAR
1.	CHAR datatype is used to store character strings of fixed length	VARCHAR datatype is used to store character strings of variable length
2.	In CHAR, If the length of the string is less than set or fixed-length then it is padded with extra memory space.	In VARCHAR, If the length of the string is less than the set or fixed-length then it will store as it is without padded with extra memory spaces.
3.	CHAR stands for "Character"	VARCHAR stands for "Variable Character"
4.	Storage size of CHAR datatypes is equal to n bytes i.e. set length	The storage size of the VARCHAR datatype is equal to the actual length of the entered string in bytes.
5.	We should use the CHAR datatype when we expect the data values in a column are of the same length.	We should use the VARCHAR datatype when we expect the data values in a column are of variable length.
6.	CHAR takes 1 byte for each character	VARCHAR takes 1 byte for each character and some extra bytes for holding length information
9.	Better performance than VARCHAR	Performance is not good as compared to CHAR

Rounding off in SQL

a. ROUND()



```
SELECT
  id,
  ROUND(price_net * 1.24, 2) as price_gross
FROM product;
```

   i. Here, the result will be rounded to decimals upto 2 places, like 1.236 will be rounded to 1.24
   ii. But if ROUND function is only passed with a single parameter, it returns a Number, like 1.236 will simply be 1.

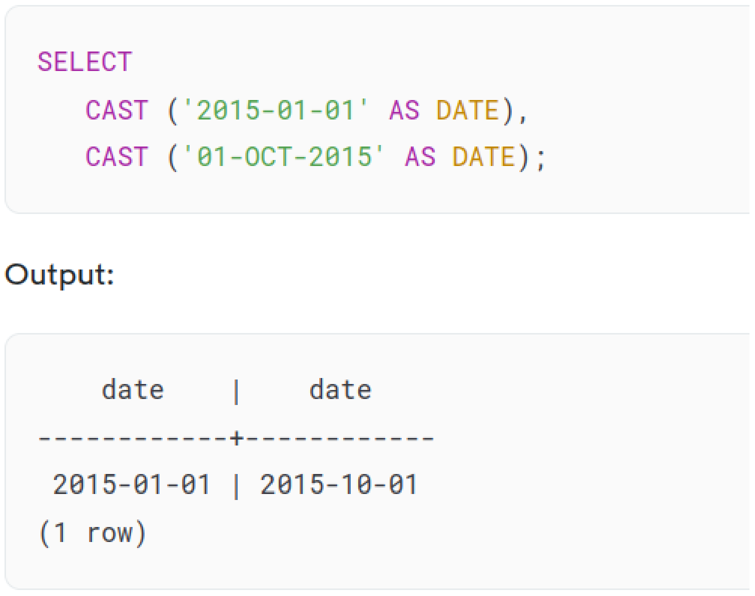
b. SELECT FLOOR(123.4567);
   i. Will return 123

c. SELECT CEIL(123.4567);
   i. Will return 124

d. SELECT TRUNCATE(123.4567, 2);
   i. Using TRUNC or TRUNCATE will truncate to a specified number of decimal places without rounding. The given example just returns 123.45 without round.

Converting Datatypes

a. **Cast() function and cast :: operator**
    i. CAST (value AS target_data_type)
        1. CAST (‘123’ as Integer)
        2. If we do something like **CAST(‘10C’ as integer),** then it will throw error.
        3. We can also use CAST for converting a string to date, both of these return the same output.
        4. When converting string to double, we can't just use DOUBLE, but need to use DOUBLE PRECISION.
            a. SELECT CAST ('10.2' AS DOUBLE PRECISION);
        5. SELECT CAST('{"name": "John"}' AS JSONB);



```
SELECT
   CAST ('2015-01-01' AS DATE),
   CAST ('01-OCT-2015' AS DATE);
```

Output:

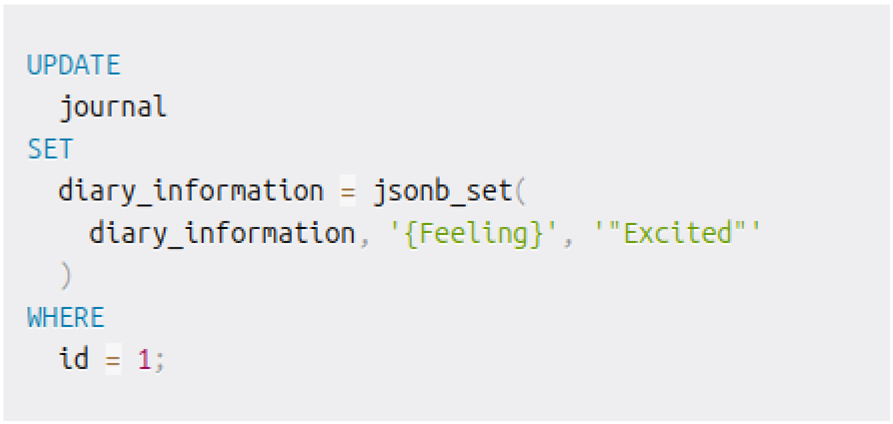
```
    date    |    date
------------+------------
 2015-01-01 | 2015-10-01
(1 row)
```

    ii. CAST OPERATOR (::)
        1. value::target_data_type
        2. SELECT '123'::INTEGER;
        3. SELECT '2019-06-15 14:30:20'::timestamp;

b. **CONVERT function**
    i. SELECT CONVERT(varchar, 25.65);
    ii. SELECT CONVERT(datetime, '2017-08-25');
    iii. Convert function can also take a third parameter like this
        1. SELECT CONVERT(varchar, '2017-08-25', 101);
            a. Here, 101 means style code (date format). In SQL Server, style 101 corresponds to the MM/DD/YYYY format.
        2. SELECT CONVERT(VARCHAR, '2024-06-26', 102) AS formatted_date;
            a. -- Result: '2024.06.26'

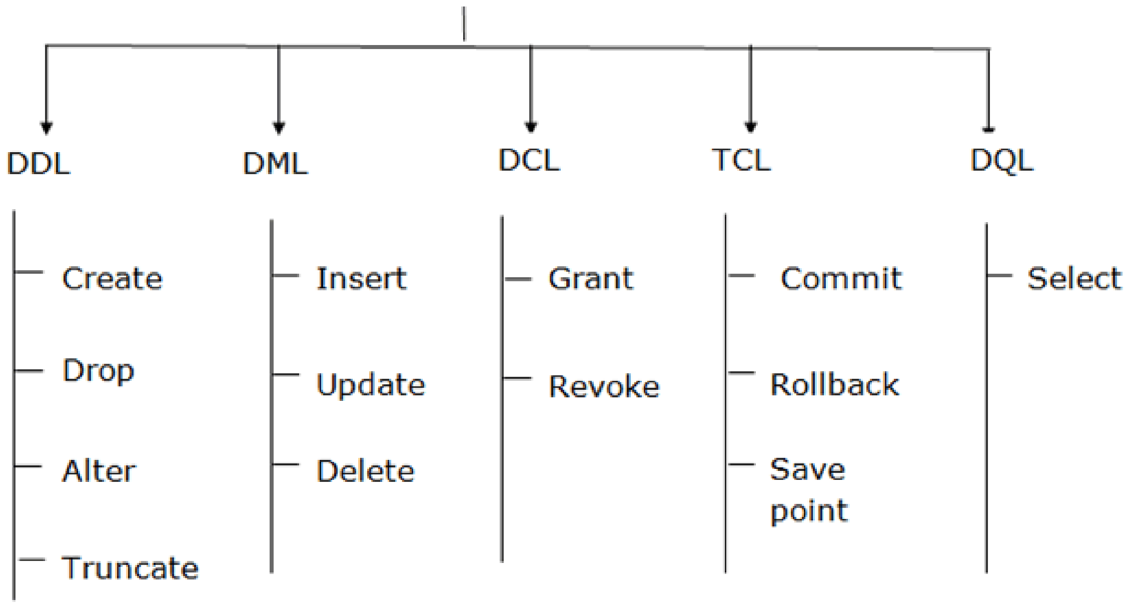
Storing JSON in PostgreSQL

a. Stored in JSONB format (JSON Binary)
b. Json data is in binary format, so indexing is faster, and query performance is also better.
c. Better storage and retrieval of data.
d. **->**: This operator allows you to **extract a specific value from a JSON object**, you specify the key as a “child” to the “parent”.
    i. SELECT
       Id, day,
       diary_information -> 'Feeling' AS Feeling
       FROM
       journal;
    ii. **this operator extracts the field name, with the quote around it.**
e. - - > : to extract a JSON object field as text without the quotes around it from a JSON object.
f. json_agg: This function aggregates JSON values into a JSON array.
g. Jsonb_set : updates an existing JSON object field with a new value.

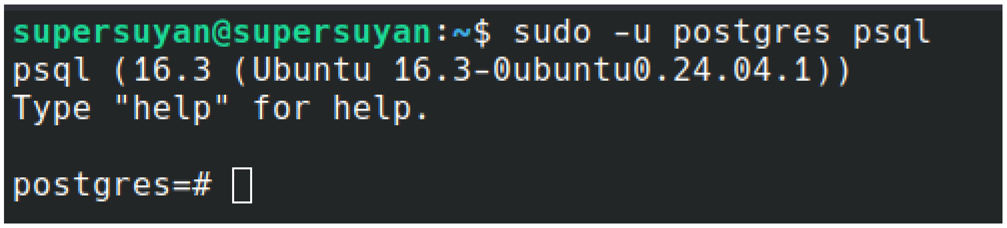


```
UPDATE
  journal
SET
  diary_information = jsonb_set(
    diary_information, '{Feeling}', '"Excited"'
  )
WHERE
  id = 1;
```

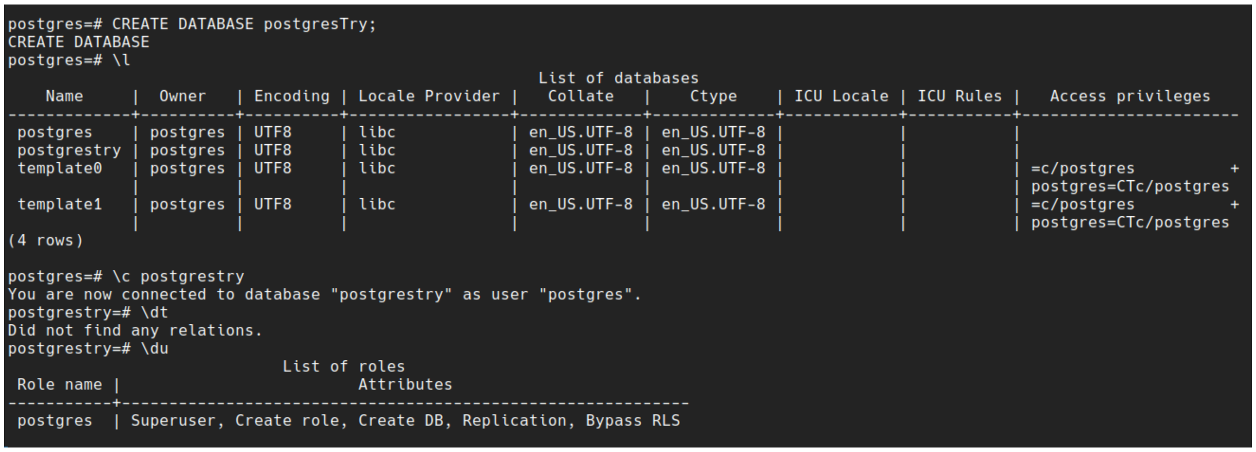
    i. **This will update the "Feeling" key in the "diary_information" column of the "journal" table with the new value "Excited".**
h. We shouldnot store everything as json just so we can
    i. Using constraints will be hard with json.
    ii. We cannot use foreign key in json, and will need to use cross join to join json data in another table.
    iii. The JSONB data type in PostgreSQL gives you much more flexibility, but with great flexibility comes more planning



Well, this is how i entered psql.



```
supersuyan@supersuyan:~$ sudo -u postgres psql
psql (16.3 (Ubuntu 16.3-0ubuntu0.24.04.1))
Type "help" for help.

postgres=# 
```

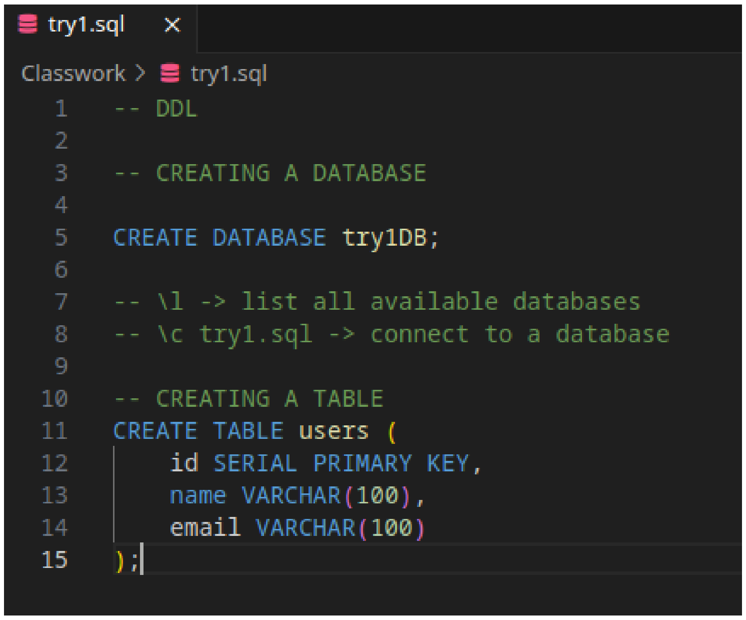


```
postgres=# CREATE DATABASE postgresTry;
CREATE DATABASE
postgres=# \l
                                                   List of databases
    Name     |  Owner   | Encoding | Locale Provider |   Collate   |    Ctype    | ICU Locale | ICU Rules |   Access privileges
-------------+----------+----------+-----------------+-------------+-------------+------------+-----------+-----------------------
 postgres    | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           |
 postgrestry | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           |
 template0   | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +
             |          |          |                 |             |             |            |           | postgres=CTc/postgres
 template1   | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +
             |          |          |                 |             |             |            |           | postgres=CTc/postgres
(4 rows)

postgres=# \c postgrestry
You are now connected to database "postgrestry" as user "postgres".
postgrestry=# \dt
Did not find any relations.
postgrestry=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS
```

DDL

a. DATA DEFINITION LANGUAGE
b. changes the structure of the table like creating a table, deleting a table, altering a table, etc.
c. auto-committed that means it can permanently save all the changes in the database.

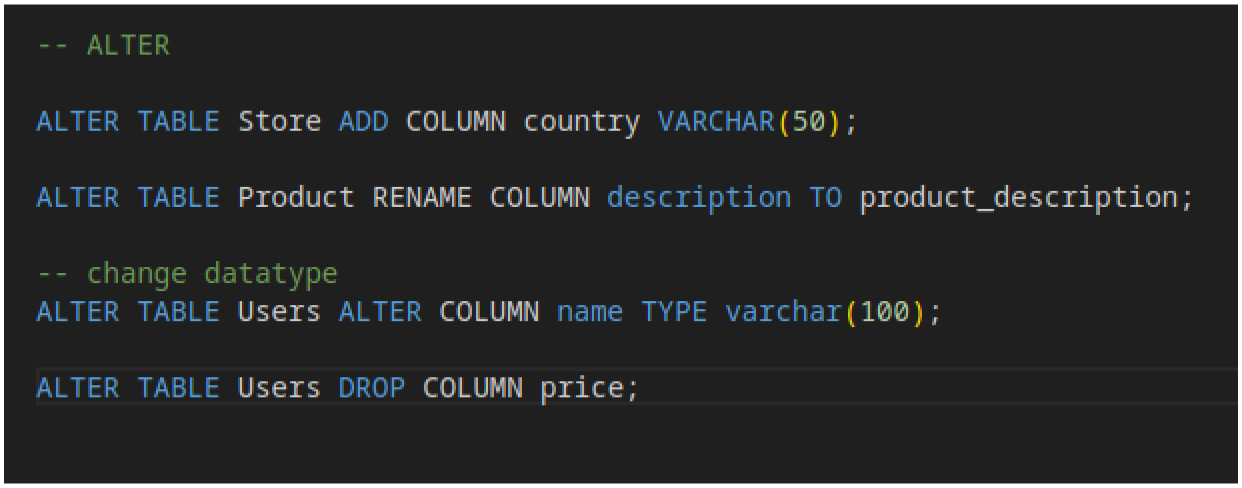
d. CREATE
    i. create a new table in the database.



```sql
-- DDL

-- CREATING A DATABASE

CREATE DATABASE try1DB;

-- \l -> list all available databases
-- \c try1.sql -> connect to a database

-- CREATING A TABLE
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);
```

e. DROP
    i. It is used to delete both the structure and record stored in the table.

f. ALTER
    i. alter the structure of the database.
    ii. modify the characteristics of an existing attribute or probably to add a new attribute.



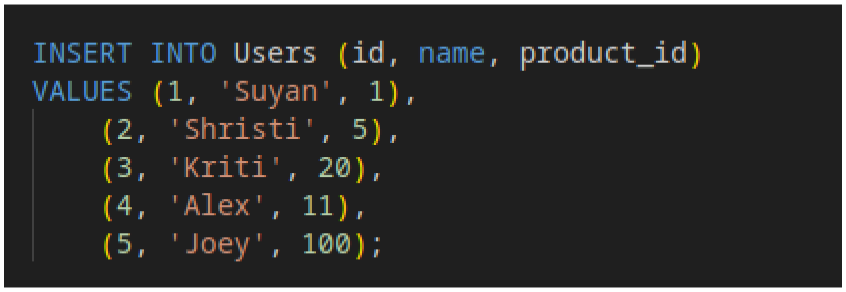
```sql
-- ALTER

ALTER TABLE Store ADD COLUMN country VARCHAR(50);

ALTER TABLE Product RENAME COLUMN description TO product_description;

-- change datatype
ALTER TABLE Users ALTER COLUMN name TYPE varchar(100);

ALTER TABLE Users DROP COLUMN price;
```

g. TRUNCATE
    i. It is used to delete all the rows from the table and free the space containing the table.

DML

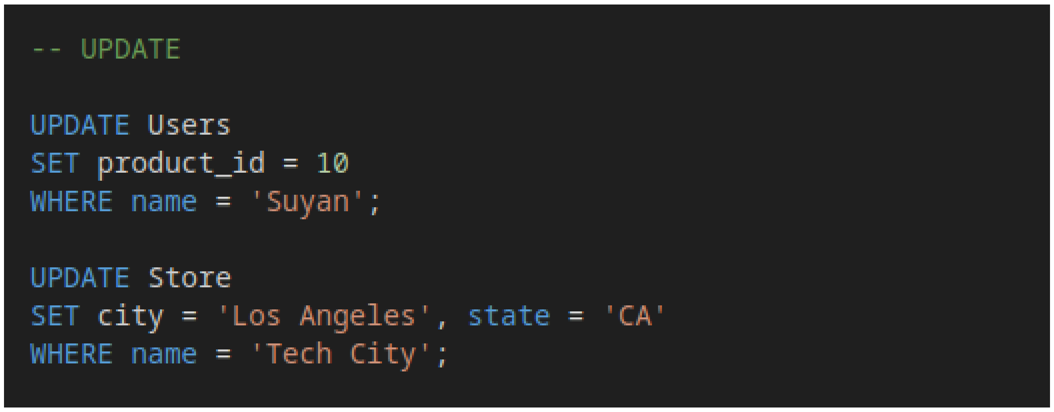
a. DATA MANIPULATION LANGUAGE
b. used to modify the database.
c. not auto-committed that means it can't permanently save all the changes in the database.

a. INSERT
   i. used to insert data into the row of a table.



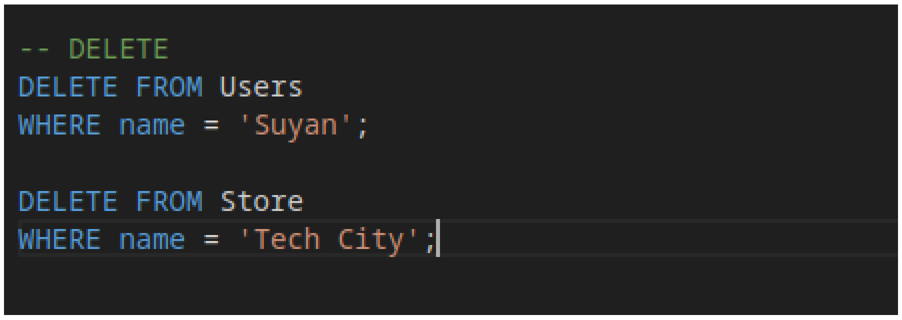
```sql
INSERT INTO Users (id, name, product_id)
VALUES (1, 'Suyan', 1),
    (2, 'Shristi', 5),
    (3, 'Kriti', 20),
    (4, 'Alex', 11),
    (5, 'Joey', 100);
```

b. UPDATE
   i. update or modify the value of a column in the table.



```sql
-- UPDATE

UPDATE Users
SET product_id = 10
WHERE name = 'Suyan';

UPDATE Store
SET city = 'Los Angeles', state = 'CA'
WHERE name = 'Tech City';
```

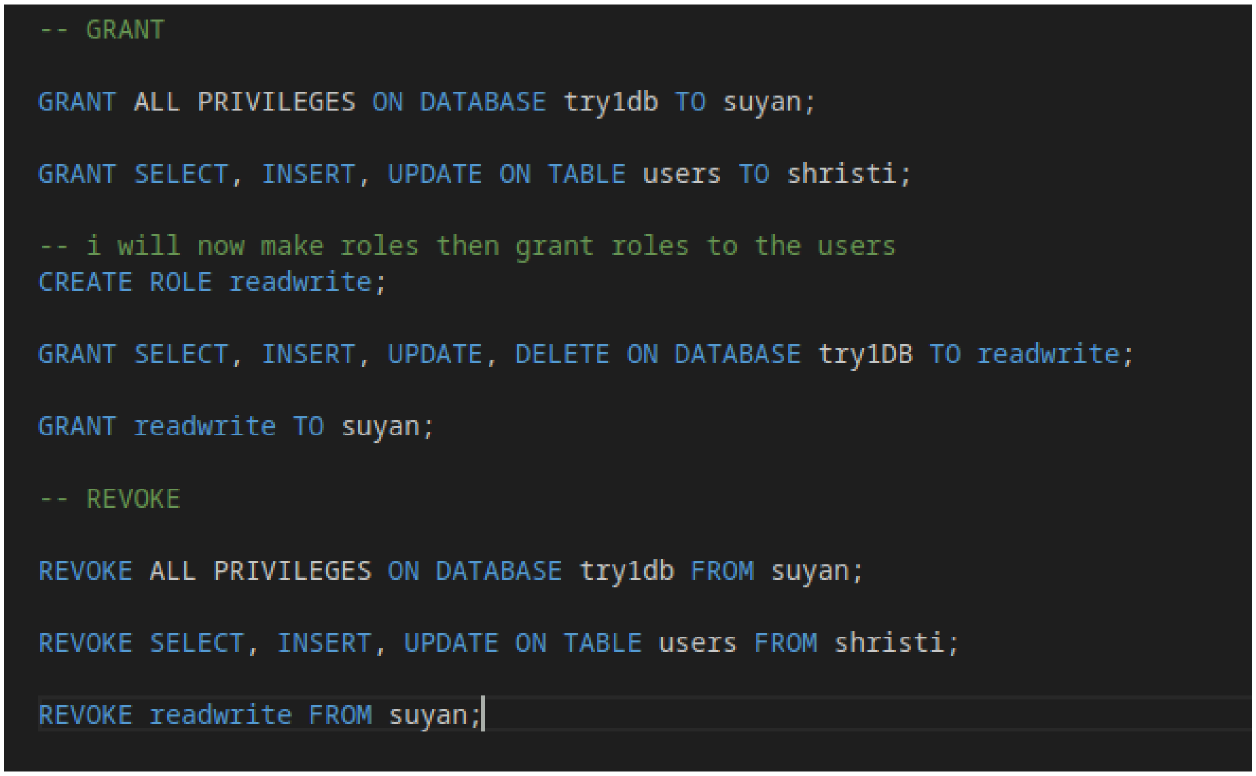
c. DELETE
   i. remove one or more row from a table.



```sql
-- DELETE
DELETE FROM Users
WHERE name = 'Suyan';

DELETE FROM Store
WHERE name = 'Tech City';
```

DCL

a. DATA CONTROL LANGUAGE
b. used to grant and take back authority from any database user.
c. GRANT
    i. used to give user access privileges to a database.
d. REVOKE
    i. to take back permissions from the user.



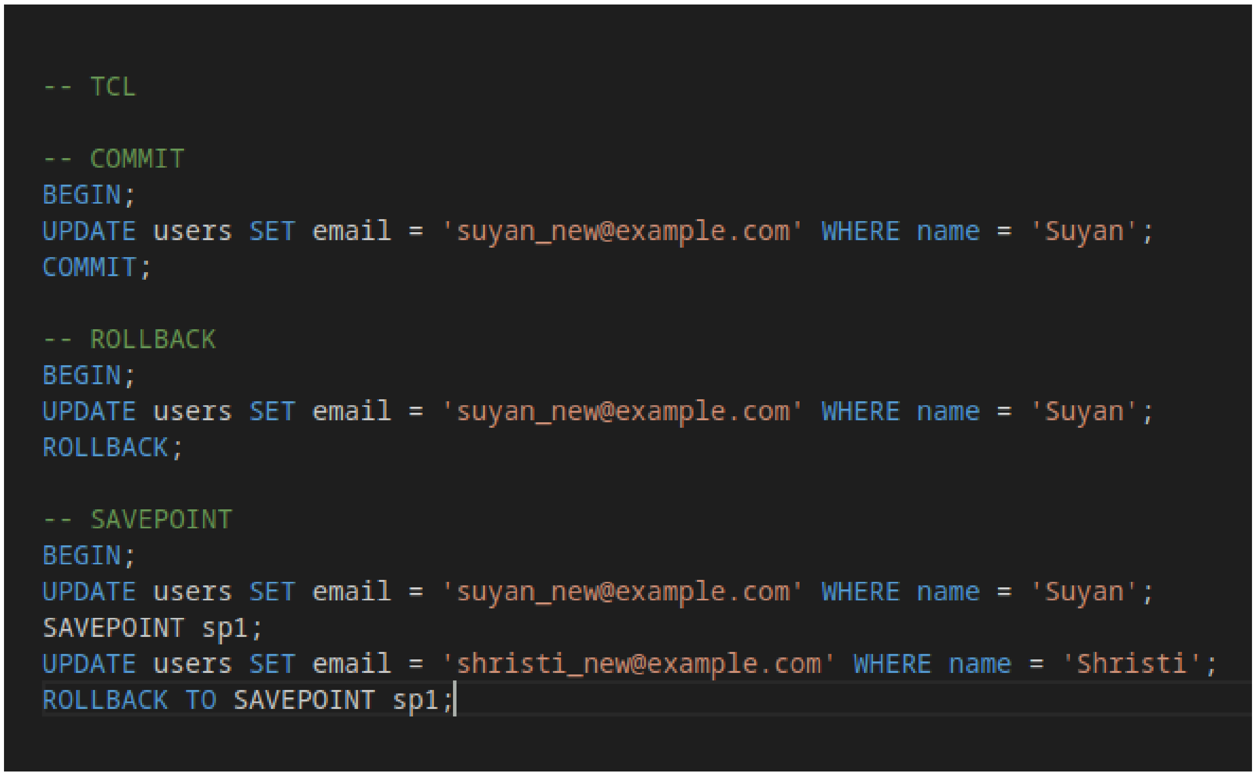
```sql
-- GRANT

GRANT ALL PRIVILEGES ON DATABASE try1db TO suyan;

GRANT SELECT, INSERT, UPDATE ON TABLE users TO shristi;

-- i will now make roles then grant roles to the users
CREATE ROLE readwrite;

GRANT SELECT, INSERT, UPDATE, DELETE ON DATABASE try1DB TO readwrite;

GRANT readwrite TO suyan;

-- REVOKE

REVOKE ALL PRIVILEGES ON DATABASE try1db FROM suyan;

REVOKE SELECT, INSERT, UPDATE ON TABLE users FROM shristi;

REVOKE readwrite FROM suyan;
```

DQL

a. SELECT

TCL

a. Transaction Control Language
b. used to manage transactions in a database
c. Commit
    i. to save all the transactions to the database.
d. Rollback
    i. to undo transactions that have not already been saved to the database.
e. SAVEPOINT
    i. to temporarily save a transaction so you can rollback to that point whenever necessary.



```sql
-- TCL

-- COMMIT
BEGIN;
UPDATE users SET email = 'suyan_new@example.com' WHERE name = 'Suyan';
COMMIT;

-- ROLLBACK
BEGIN;
UPDATE users SET email = 'suyan_new@example.com' WHERE name = 'Suyan';
ROLLBACK;

-- SAVEPOINT
BEGIN;
UPDATE users SET email = 'suyan_new@example.com' WHERE name = 'Suyan';
SAVEPOINT sp1;
UPDATE users SET email = 'shristi_new@example.com' WHERE name = 'Shristi';
ROLLBACK TO SAVEPOINT sp1;
```

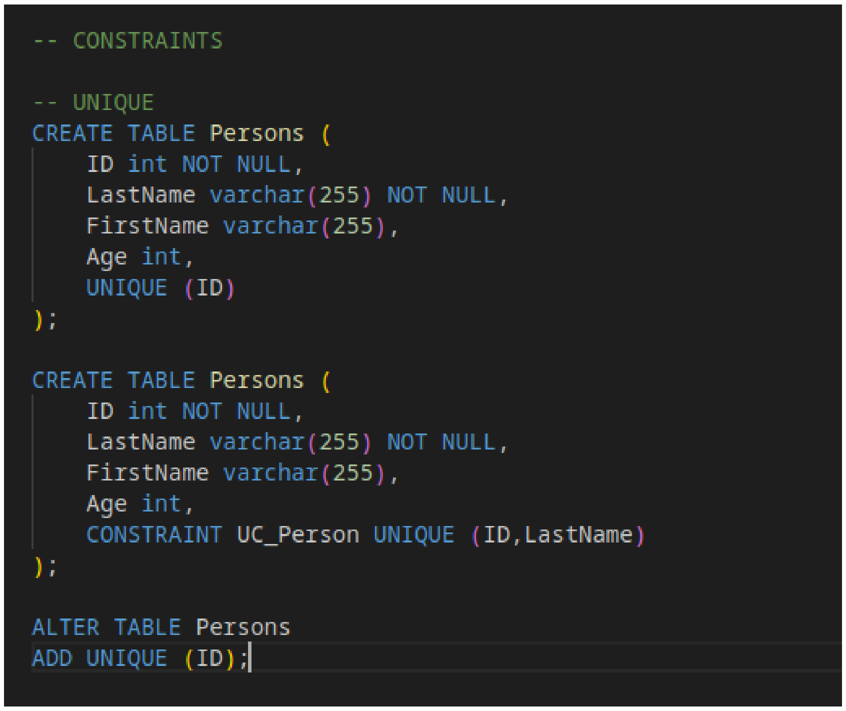
Constraints

a. Constraints are used to limit the type of data that can go into a table.
b. This ensures the accuracy and reliability of the data in the table
c. If there is any violation between the constraint and the data action, the action is aborted.

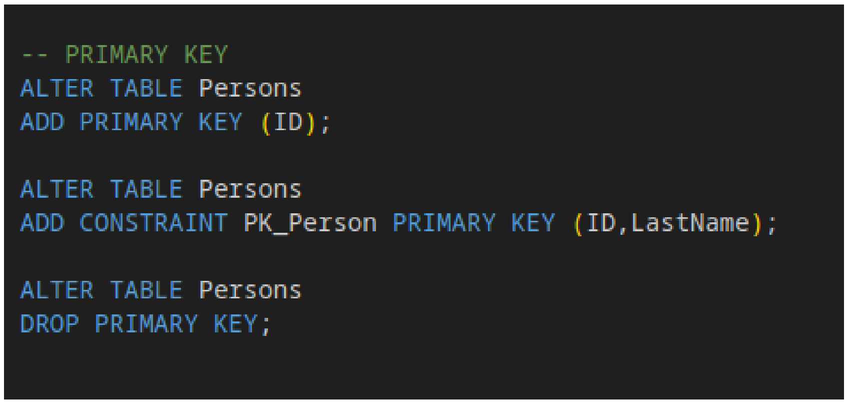
d. NOT NULL
   i. to NOT accept NULL values.

e. UNIQUE
   i. ensures that all values in a column are different.

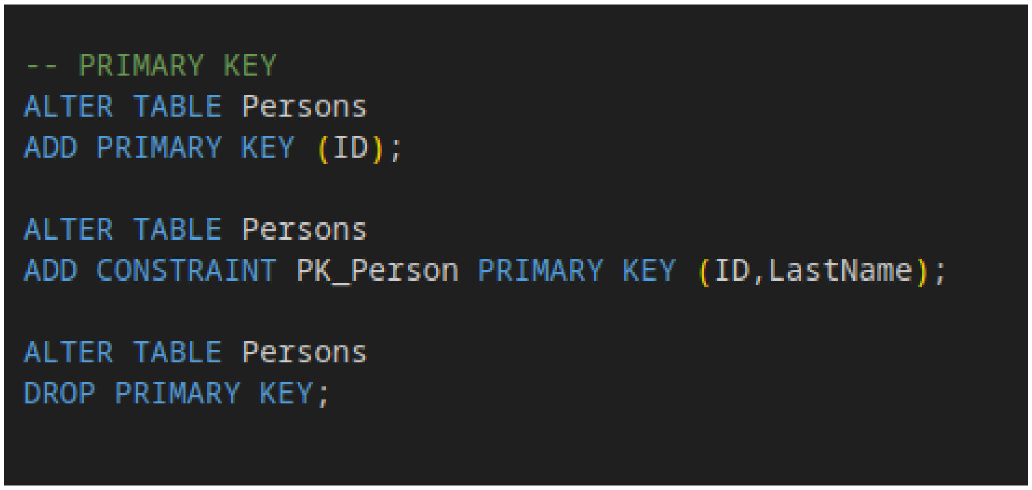
f. PRIMARY KEY
   i. Primary keys must contain UNIQUE values, and cannot contain NULL values.
   ii. A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).



```sql
-- CONSTRAINTS

-- UNIQUE
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);

CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);

ALTER TABLE Persons
ADD UNIQUE (ID);
```

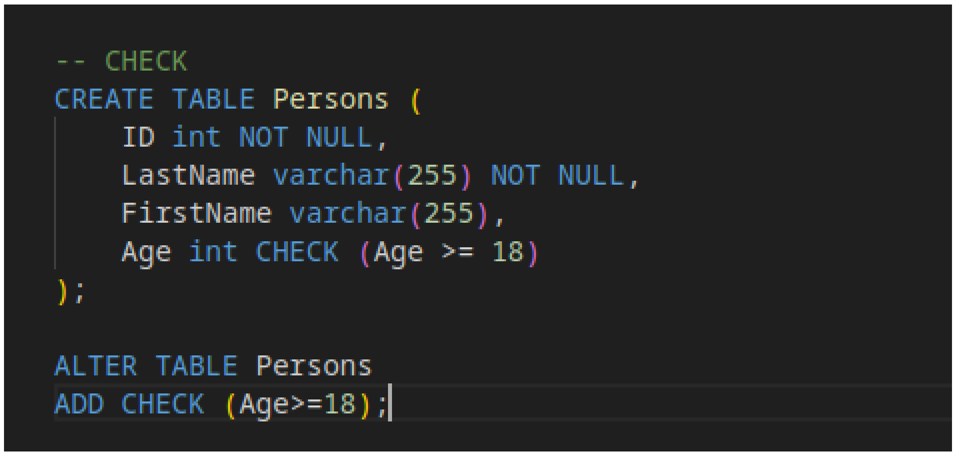


```sql
-- PRIMARY KEY
ALTER TABLE Persons
ADD PRIMARY KEY (ID);

ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);

ALTER TABLE Persons
DROP PRIMARY KEY;
```



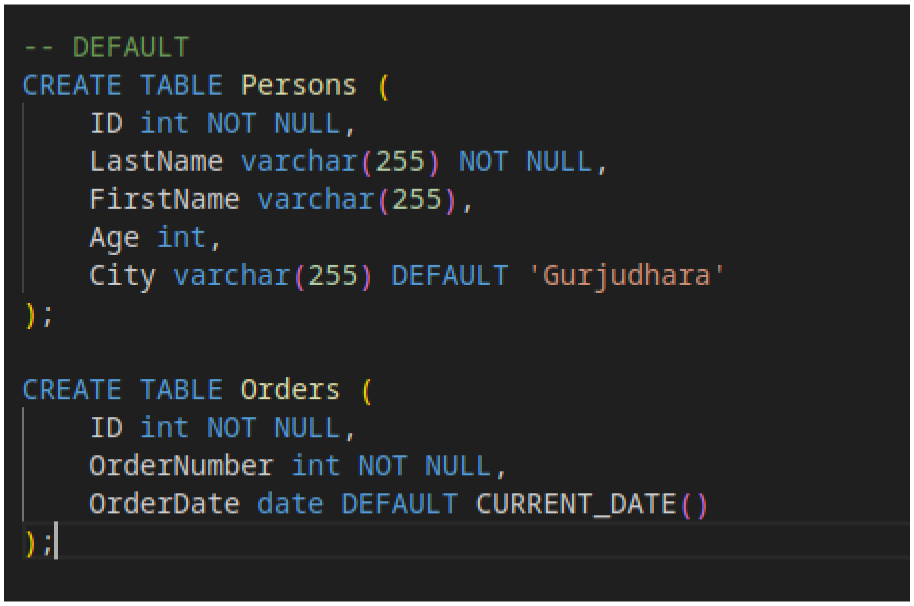
```sql
-- PRIMARY KEY
ALTER TABLE Persons
ADD PRIMARY KEY (ID);

ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);

ALTER TABLE Persons
DROP PRIMARY KEY;
```

g. CHECK
   i. to limit the value range that can be placed in a column.



```sql
-- CHECK
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age >= 18)
);

ALTER TABLE Persons
ADD CHECK (Age>=18);
```

h. DEFAULT
   i. is used to set a default value for a column.



```sql
-- DEFAULT
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Gurjudhara'
);

CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT CURRENT_DATE()
);
```