30th Aug '22

① Create table, CREATE Database is used to set up the dB.

② The whole dB setup information is stored in file called Schema.

IMP ③ SQL likes to use semicolon and single quotes

EVERY COMMAND CONSISTS OF FOLLOWING:
the action ("statement") the table it should run on, and the conditions ("clauses").

④ Delete Queries require "WHERE" clause. Can also use <, >, <=, >= and also logical operators like AND, OR, NOT,

⑤ Create queries use INSERT INTO which requires columns to insert stuff into & which values to put in those columns

eg- INSERT INTO Users (name, email) VALUES ('foobar', 'foobar@bar.com');

(6) Update requires 'SET' (using key="value" pairs)
eg-
UPDATE Users
SET name = 'barfoo', email = 'bar@foo.com'
WHERE email = 'foo@bar.com';

(7) Read Queries use SELECT eg- SELECT * FROM users WHERE created_at < '2013-12-11 15:35:59'.
* ⇒ all columns
SELECT DISTINCT if we want unique values of column. Spose, list of diff. names of users without any duplicates say:-
SELECT DISTINCT users.name FROM Users

(8) 1. JOIN / INNER JOIN: Keep only the rows from both tables they match.
SELECT * FROM Users JOIN posts ON users.id = posts.user_id

2. LEFT OUTER JOIN - keep rows from left table & add rows from right table which match up to left tables
3. RIGHT OUTER JOIN: the opposite, keep all rows in right table
4. FULL OUTER JOIN: keep all rows from both tables, even if there are mismatches. Set mismatched cell to NULL.

* To return single values, we use aggregate functions:
eg:- SELECT MAX(users.age) FROM users

* Aliases are used - 'AS'. eg-
SELECT MAX(users.age) AS highest_age
FROM users

** With aggregate functions we use 'HAVING' instead of 'WHERE'

(1) To return columns we use ⇒
*, specific column name.
(2) To return rows (particular rows) we use WHERE keyword.

(3) WHERE... EQUALS
WHERE... Greater than
WHERE... Greater than or equal
AND
OR

* IN ⇒ using WHERE clause, we can find rows where a value is in a list of several possible values.

SELECT * FROM friends_of_pickles WHERE species IN ('cat', 'human');
Above line would return friends of pickles that are either a cat or a

human. => Returns rows with 'cat' & 'human'

→ To find rows that are not in a list, you use NOT IN istead of IN.

④ Using DISTINCT with SELECT, we do not return duplicates
eg -
SELECT DISTINCT gender, species FROM friends_of_pickles WHERE height_cm <100;

⇒ Above statement will get gender/species combinations of the animales less than 100 cm in height

** NOTE: Even though there are multiple male dogs under that height, we only see one row that returns 'male' and 'dog'.

⑤ ORDER BY (returns all columns)
This is used to sort the rows by some kind of attribute, we use ORDER BY. Example: we want to sort friends_of_pickles by name, we run:

⇒ SELECT * FROM friends_of_pickles ORDER BY name;
Above statement returns name in alephabetical ascending order.

* To sort values in descending order we use DESC keyword at end of query.

(6) Limit # of returned rows.

→ If we just want to see a few examples of data in table, we can select just first few rows with LIMIT keyword. By using ORDER BY, you would get the first rows of that order.

eg: SELECT * FROM friends_of_pickles ORDER BY weight_cm LIMIT 2;
// returns two shortest friends of pickles.

NOTE: The LIMIT keyword comes after DESC keyword.

Ques: Can you return the single row of the tallest friends_of_pickles?

↓

order by, could be used to sort from large to small in descending order. Single row could be obtained by using LIMIT

(7) COUNT(*) → returns single column returns total number of rows

SELECT COUNT(*) FROM friends_of_pickles;

(8) COUNT(*) w/ WHERE returns number of rows that matches the WHERE clause

eg: SELECT COUNT(*) FROM friends_of_pickles WHERE species = 'human';

should return 2 (rows) for following table friends_of_pickles

	ID	Gender	species	weight_in
1	Dave	male	human	180
2	Mary	female	human	160
3	Fry	male	cat	30
4	Leela	female	cat	95

(9) SUM(num_legs) returns total number of legs in the family
→ single column
→ returns single column

~~AA~~ similarly for AVG(condition)
→ returns single column

(10) similarly for MAX(conditions), MIN(condition)
SELECT MIN(num_legs) FROM family_members;

(11) GROUP BY:

(1) We can use aggregate functions such as COUNT, SUM, AVG, MAX and MIN with the GROUP BY clause.

When you GROUPBY something, you split the table into different piles based on value of each row.

eg: SELECT COUNT(*), species FROM friends_of_pickle GROUP BY species; would return number of rows for each species.

(12) NESTED QUERIES:

SELECT ..... = ( )

Parenthesis Query is executed first.

(13) NULL

Sometimes a given column might not have any value that is null for eg. a dog will not have fav-book value. so to find rows where value of column is or NOT NULL. We use

⟹ IS NULL or IS NOT NULL;

(14) Date

→ a column can also have a date value. like 1985-07-20 means 20 July 1985

→ we can compare dates by > and <.

eg:-

⟹ SELECT * FROM celebs_born WHERE birthdate < '1985-07-20'; returns celebs born before 20 July 1985

(15) INNER JOIN:

① Select columns that you want then write first table name followed by "INNER JOIN" second table name "ON" column/id = column/id
(Common attribute b/w two tables)

Eg:

character table:

id	name
1	Doogie
2	Barney
3	Lily

character_tv_show:

id	char_id	tv_show_name
1	3	How I met ur mother
2	2	"
3	1	Doogie Howser

character_actor

id	char_id	actor_name
1	3	Alyson Hannigan
2	2	Neil Patrick
3	1	Neil Patrick

Ques) Can u use an Inner Join to pair each character name w/ the actor who plays them?

Ans.1
```
SELECT character.name, character_actor.actor_name
FROM character INNER JOIN character_actor
ON character.id = character_actor.char_id;
```

Date: YOUVA

(16) MULTIPLE JOINS:

character

id	name
1	Doogie Browser
2	Barney Stinson
3	Lily Aldrin
4	Willow Rosenberg

tv_show

id	name
1	Buffy the Vampire
2	How I Met Ur Mother
3	Doogie Howser, M.D

character_tv_show

id	character_id	tv_show_id
1	1	3
2	2	2
3	3	2
4	4	1

actor

id	name
1	Alyson Hannig
2	Neil Patrick

character_actor

id	character_id	actor_id
1	1	2
2	2	2
3	3	1
4	4	1

Ex: To get each character name with his/her TV show name, we can write

```
SELECT character.name, tv_show.name FROM
character INNER JOIN character_tv_show
ON character.id = character_tv_show.character
_id

INNER JOIN tv_show ON character_tv_show.tv_
_show_id = tv_show.id;
```

=>

id	character_id	name
1	1	Doogie Howser, M.D
2	2	How I Met Ur Mother
3	3	How I Met Ur Mother
4	4	Buffy the Vampire

Now character.id would be equated with character_tv_show.character_id from above table & names from character table and above table would be retrieved (on corresponding ids)

Ques.) Can you use two joins to pair each character name with the actor who plays them?

Ans.

```
SELECT character.name, actor.name FROM
character INNER JOIN character_actor ON
character.id = character_actor.character_id
INNER JOIN actor ON character_actor.actor_id=
actor.id;
```

(17) Joins with WHERE

4 can use WHERE in multiple join to be more specific

eg:- To get list of characters & TV shows that are not in "Buffy the Vampire Slayer" and are not "Barney Stinson". You would run:

```
SELECT character.name, tv_show.name
FROM character INNER JOIN
character_tv_show ON character.id =
character_tv_show.character_id
INNER JOIN tv_show ON character_tv_
show.tv_show_id = tv_show.id WHERE
character.name != 'Barney Stinson'
AND tv_show_name != 'Buffy the
Vampire Slayer';
```

(18) LEFT JOIN

This returns all data from left table and if there isn't corresponding data for the second table. It returns NULL for those columns.

RIGHT JOIN and OUTER JOIN are not present in SQLite

(19) Table Alias

→ Queries can be shorter on using aliases

⇒ If we want to use an alias for a table, you add AS *alias_name* after table name.

eg- To use left join b/w characters and tv shows with aliases, you would run:

```
SELECT coname, t.name FROM character
as c LEFT JOIN character_tv_show AS
ct ON c.id = ct.character_id
LEFT JOIN tv_show AS t ON
ct.tv_show_id = t.id;
```

(11) Column Alias can also be used:

⇒ Add AS *alias_name* after column name

eg:-

```
SELECT character.name AS character,
tv_show.name AS name FROM character
LEFT JOIN character_tv_show ON
character.id = character_tv_show.character_id
LEFT JOIN tv_show ON character_tv_show.tv_
show_id = tv_show.id;
```

(20) TO peform join on single table we perform self join which basically creates copy & peuforms Join.

SELECT r1.name AS object, r2.name AS beats FROM rps AS r1 INNER JOIN rps AS r2 ON r1.defeats_id = r2.id;

rps

id	name	defeats_id
1	Rock	3
2	Paper	1
3	Scissors	2

(21) SELECT * FROM robots WHERE name LIKE "Robot 20__";

robots

id	name
1	Robot 2000
2	" 2001
3	Dragon
4	Robot 2111

Ans Return "Robot" followed by a year between 2000 and 2099.

(22) SELECT *,
CASE WHEN species = 'human' THEN "talk"
WHEN species = 'dog' THEN "bark"
ELSE "meow"
END AS sound FROM friends_of_pickles;

col name

Qu Return results with column named sound that returns "talk" for human,

"bark" for dogs, & "meow" for cats?

** Left Outer Join:

(1) consists of some extra rows not present in Right table.

→ Right table consists of similar values more than once on matching column.

→ Left table consists of extra values in primary key column NOT found in right table.

** GROUP BY: ["each" used in Question]

⇒ Used to group rows of similar values in specified columns

⇒ Generally used w/ Aggregate Functions

* It's difficult to use "WHERE" w/ GROUP BY. Therefore "HAVING" clause was introduced to use with GROUP BY.

* HAVING clause constraints are applied to grouped Rows.