Modern SQL

SQL was originally developed in 1970 as a part of IBM's R project. And now the current standard is SQL:2016.

* SQL is mainly divided into 3 main languages:

1. **Data Manipulation Language (DML):** it's about manipulation of the database. SELECT, INSERT, UPDATE, and DELETE statements are the main commands used.
2. **Data Definition Language (DDL):** it's about defining schema for database and tables.
3. **Data Control Language (DCL):** it's about security and access control.

* SQL is based on **bags** (allows duplicates) not **sets** (no duplicates).

* Note) SELECT=Projection, WHERE=Selection

Aggregations

- Aggregations are functions its input is a bag of tuples and output is a single scalar value. It can only be used with **SELECT** statement. Also, A single SELECT statement can contain multiple aggregates.

→ AVG (col)→ Return the average col value.

→ MIN (col)→ Return minimum col value.

→ MAX (col)→ Return maximum col value.

→ SUM (col)→ Return sum of values in col.

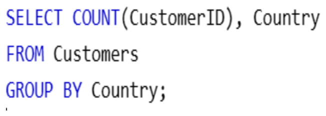
→ COUNT (col)→ Return # of values for col.

COUNT, SUM, AVG support **DISTINCT.**

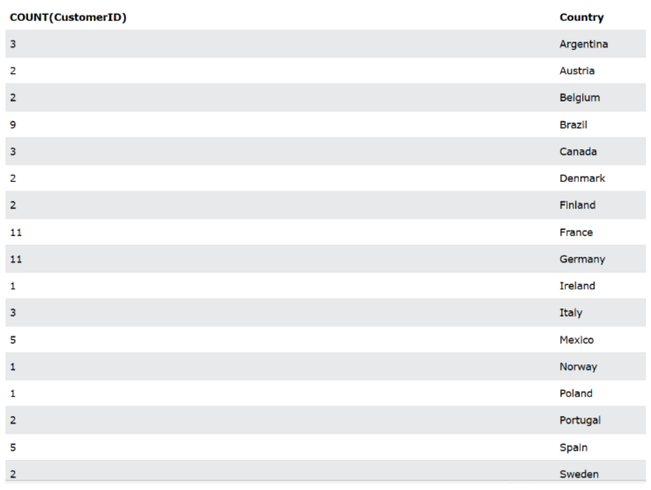
* Note) In SELECT statement, Query starts execution from (FROM) then SELECT.

* Group By clause: groups rows that have the same values into summary rows. It's often used with aggregate functions to group the result set by one or more columns.

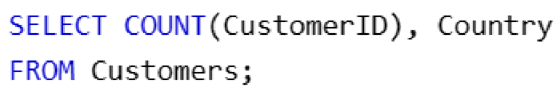
Non-aggregated values in SELECT output clause must appear in GROUP BY clause.



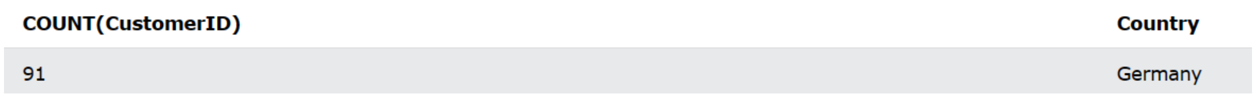
```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```



COUNT(CustomerID)	Country
3	Argentina
2	Austria
2	Belgium
9	Brazil
3	Canada
2	Denmark
2	Finland
11	France
11	Germany
1	Ireland
3	Italy
5	Mexico
1	Norway
1	Poland
2	Portugal
5	Spain
2	Sweden



```
SELECT COUNT(CustomerID), Country
FROM Customers;
```



COUNT(CustomerID)	Country
91	Germany

* Having clause: it allows for a final filter after WHERE clause is executed. It's basically a WHERE clause for Group By.

String operation

strings are case sensitive and single quotes only. There are functions to manipulate strings that can be used in any part of a query. The LIKE keyword is used for string matching in predicates. There are also concatenation between strings.

Date and Time

There are some operations on date and time that varies from a system to another.

Output Redirection

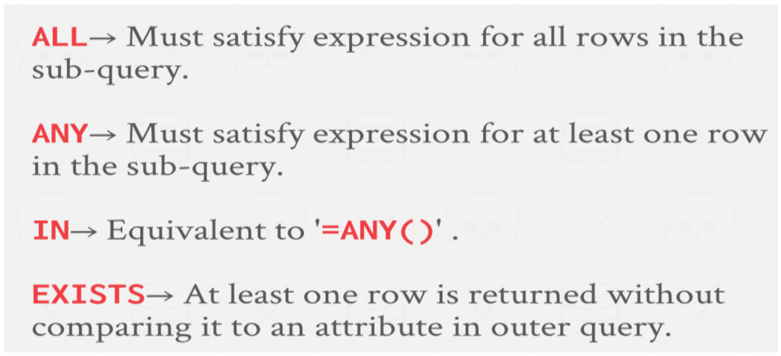
Instead of having the output in the terminal, you can store the output of the query in a another new or existing (must have the same number of columns and same data types) table.

* You can control the output using **ORDER BY** clause.

* We can use **LIMIT** clause to restrict the number of rows.

Nested Queries

Putting queries inside other queries to execute more complex logic but it's often difficult to optimize. The inner query can access attributes from the outer query.



ALL→ Must satisfy expression for all rows in the sub-query.

ANY→ Must satisfy expression for at least one row in the sub-query.

IN→ Equivalent to '=ANY()' .

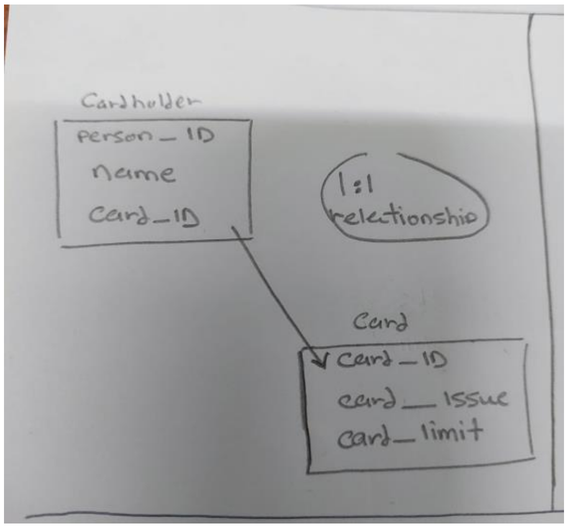
EXISTS→ At least one row is returned without comparing it to an attribute in outer query.

Database design (2H→4H)

* Primary key is the parent. Foreign key is the child.

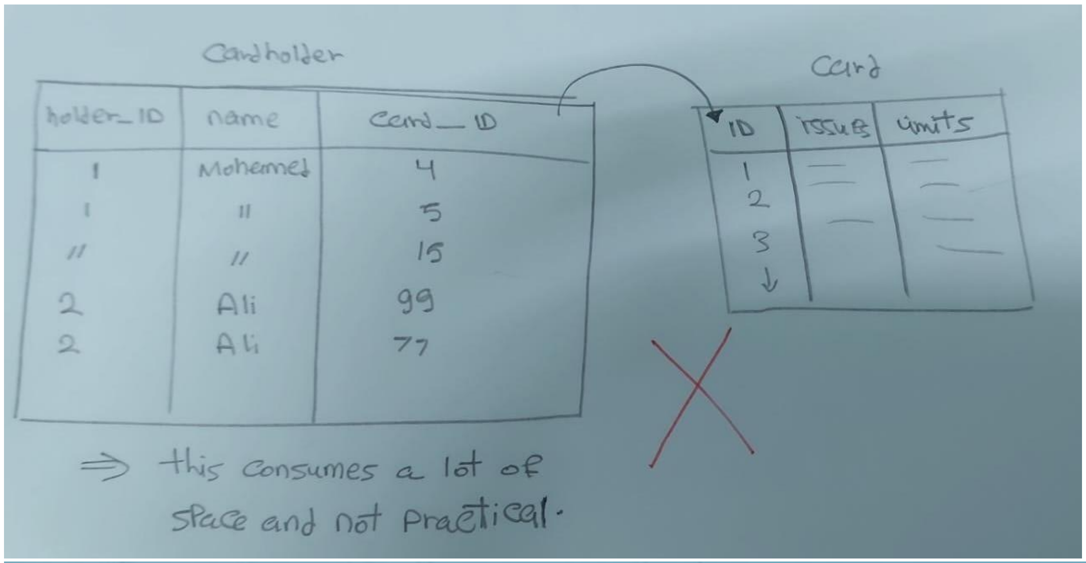
1) Designing one to one relationship:

>> you put things related to each other in one table and just connect between them. An example (A cardholder in some country have a rule that person must have only one ID card).



2)Designing one to many relationships:

The cardholder may have many cards, but it doesn't make sense to put them in the table of cardholder like (card_id1, card_id2, card_id3, …). So, we put the cardholder ID field to the card table (child) to connect it with cardholder table (parent). With this way every card is connected with it's owner instead of typing the owner many times and consuming a lot of space from database.



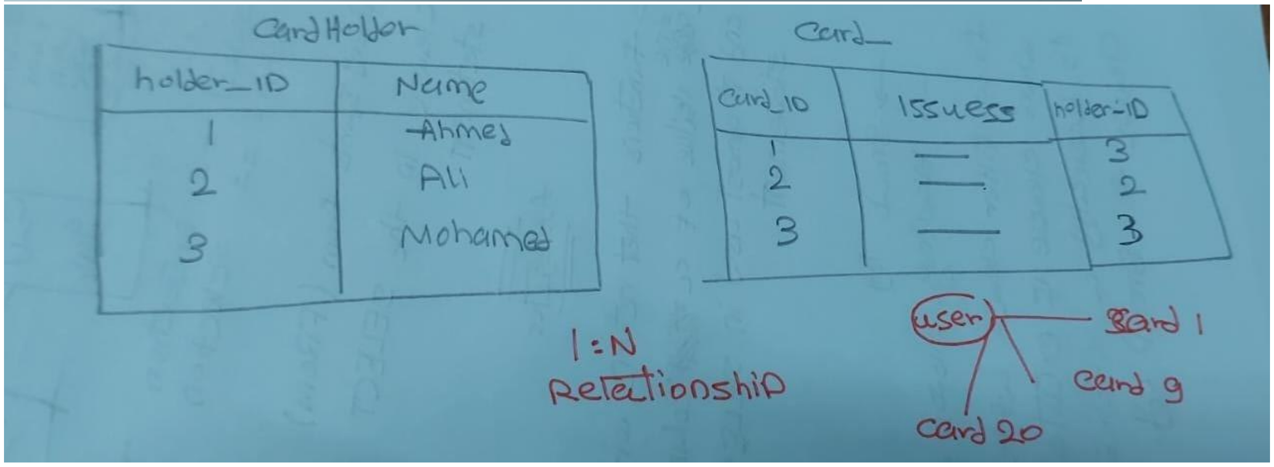
Cardholder

holder_ID	name	Card_ID
1	Mohamed	4
1	"	5
//	//	15
2	Ali	99
2	Ali	77

Card

ID	issue	limits
1	—	—
2	—	—
3	—	—
↓		

⇒ this consumes a lot of space and not practical.



CardHolder

holder_ID	Name
1	~~Ahmed~~
2	Ali
3	Mohamed

Card

Card_ID	Issuess	holder_ID
1	—	3
2	—	2
3	—	3

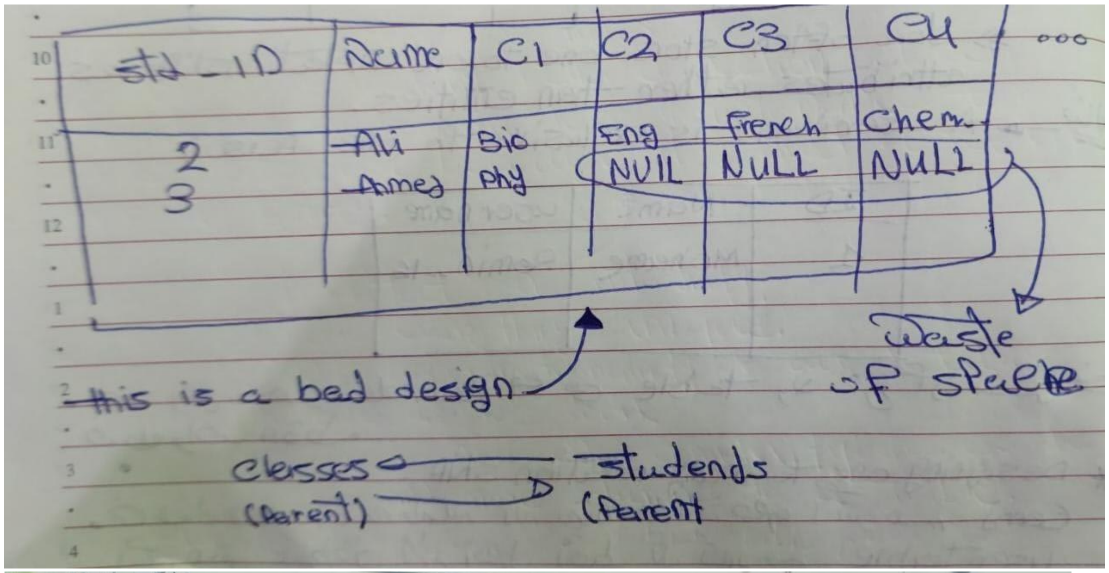
1:N Relationship
user
card 1
card 9
card 20

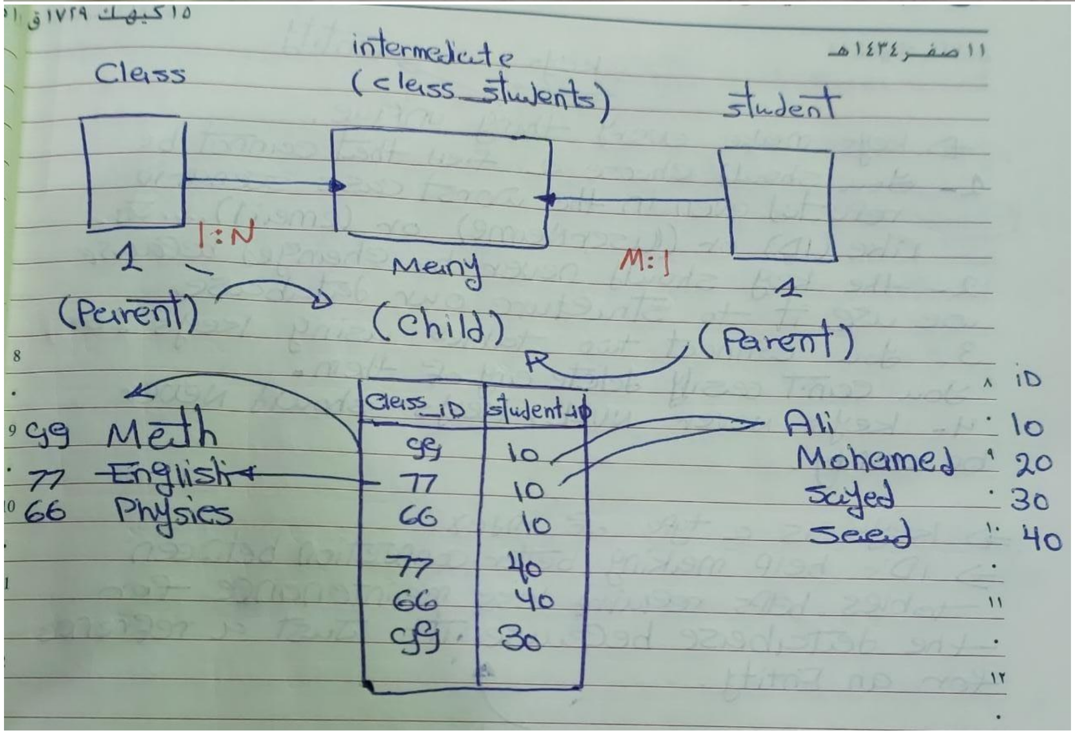
3) Designing many to many relationship:

Usually, N:M relationships are divided into to 1:N using intermediate table because it has many issues like wasting of memory and difficulty of implementing.

Example) a class contains many students, a student can have many classes.

Assume “Mohamed” is the student, he takes 3 classes (“Physics”, “English”, “Chemistry”). “Ali” is another one but he take 1 class(“English”). So there are two places in memory have NULL values and no one will use. So this is not a good workarounds when dealing with a huge database. The solution for this issue is to make intermediate table acts like child with two parents (student, class) and inherits from them.





* You choose the type of relationship depending on what you want for your application. Example (class and professor)

1. (1:1) a prof can teach only one class. A class can only taught by one prof.
2. (1: N) a prof can teach many classes. A class can taught by one prof.
3. (M: N) a prof can teach many classes. A class can taught by many profs.

Keys

Keys make everything unique.

- You should choose key that cannot be repeated like ID, username.
- The key should never be changed because we use it to structure our database.
- You should never delete any of them because you connect two tables through it.
- Keys never be Null. It should never be empty.

The importance of keys:

1. Protect our integrity.
2. Keeps everything unique.
3. Improves functionality of our database.
4. Gives us less work.
5. Allows for added complexity because you can add more columns to a table.

Super Key

Any number of columns that forces every single row to be unique.

Candidate keys

They are selected from a set of super keys which there's no redundancy. It's the least number of columns to make a row unique.

- Super keys are not practical they are for designing your database only, by asking – Can every row be unique with these columns?
    - How many candidate keys we have to pick good primary key?

Primary key

A primary key is chosen from a set of candidate keys.

Rules to choose a primary key:

- UNIQUE
- NOT NULL
- NEVER CHANGES

And the other candidate keys we don't choose are known as alternate keys.

Types of primary key:

1) Surrogate key (like ID).
   They are completely private. No one expect people working with database will know anything about them because the number hasn't real world meaning.
   Pros:
   They are numbers. So they easy for using.
   Cons:
   Adding a new column for your database.

2) Natural key (like username).
   They are about something in real life like username or email.
   Pros:
   You don't have to define a new data. So you have a smaller database And it have a real world meaning.
   Cons:
   Actual value might change and this requires updates all connections and this requires a lot of resources from our server. Also, sometimes you might not be able to choose a perfect natural key.