What is a Database?

A Database is a software used to store data in any form.

Why we need special software?

Database lives in cloud

A cloud is a renting pc Cloud Storage providers are not the cloud. A server is a group of PC's

Linux is used in cloud

Advantages of Linux

1. Free
2. Open-Source - All are travelling on the same boat, thats why its important
3. Security
4. Smaller footprint
5. Automation - Can be controlled all operations from the terminal, you wont even need a mouse.

Distro of linux :

- Linux Mint
- Ubuntu etc

Features of Database

1. Database - Frequently asked data, it will have it in the ram
2. Querying becomes easy
3. CRUD - easy
4. Backups are inbuilt
5. Undo - easily (time limit)
6. Performance (As storing in the RAM)

SQL Theory

Keys

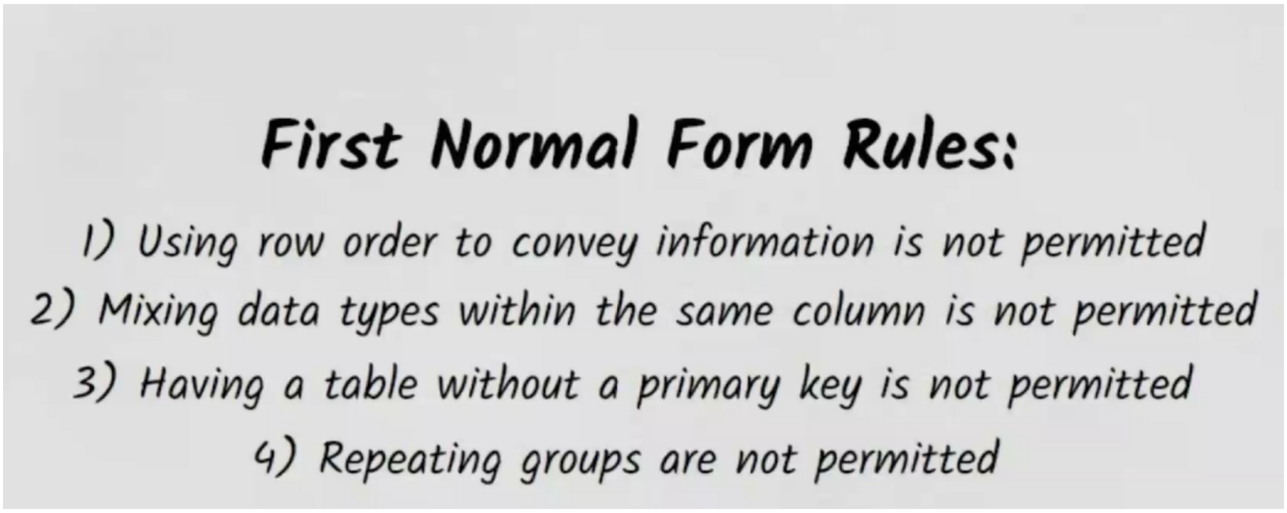
- Primary Key :
    1. Should be UNIQUE
    2. Should be NOT NULL
    3. Only one column a table should be PK
-

Normalization

To avoid anomolies we do normalize the tables or To increase the safety of the data

Types of Normal Forms:

1. 1st Normal Form

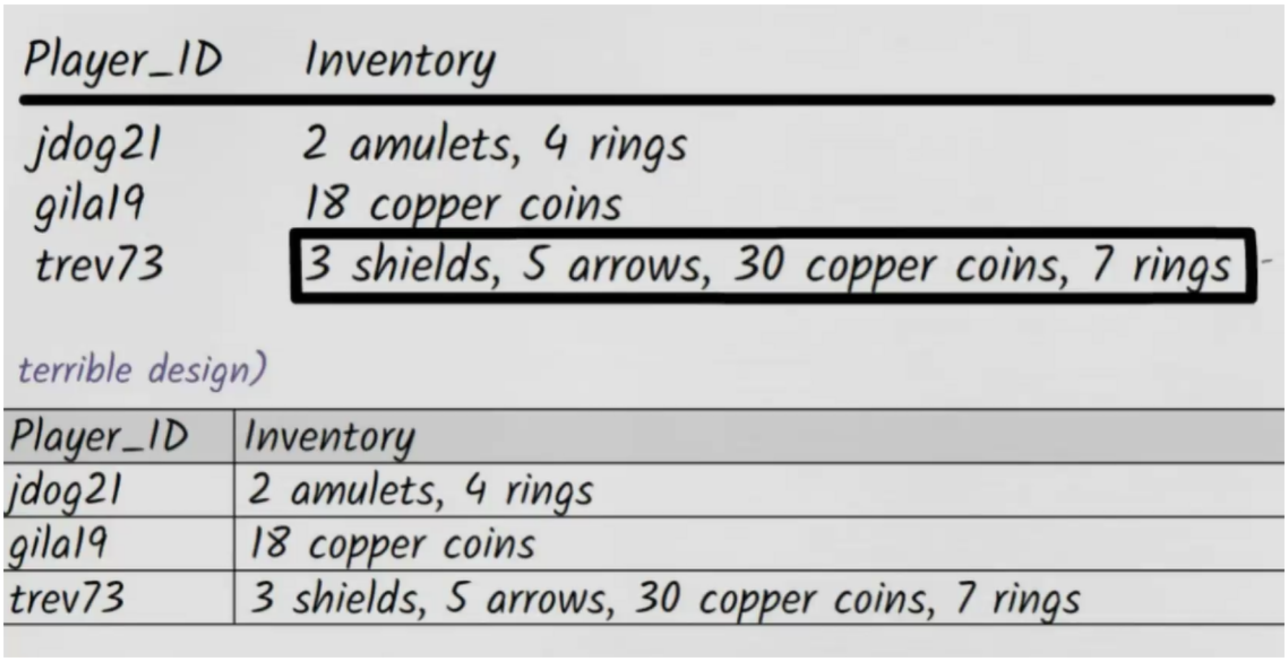
   

   First Normal Form Rules:

   1) Using row order to convey information is not permitted
   2) Mixing data types within the same column is not permitted
   3) Having a table without a primary key is not permitted
   4) Repeating groups are not permitted

   Row order should not contain any information. Dont mix Data types within the same column Table without primarykey is not allowed Repeating groups are not permitted

   Task 1 (Convert to NF)

Player_ID	Inventory
jdog21	2 amulets, 4 rings
gila19	18 copper coins
trev73	3 shields, 5 arrows, 30 copper coins, 7 rings

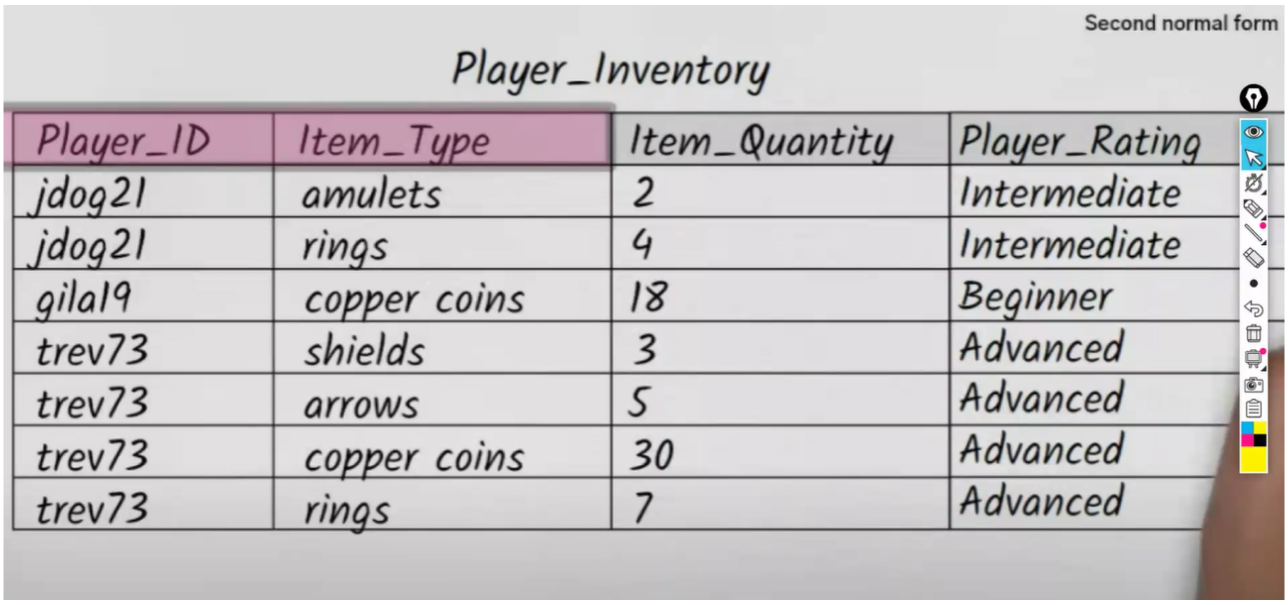
   terrible design)

Player_ID	Inventory
jdog21	2 amulets, 4 rings
gila19	18 copper coins
trev73	3 shields, 5 arrows, 30 copper coins, 7 rings

player_id	Inventory	Quantity
abc	rings	2
xyz	shields	3

   Primary Key - Combinely id and inventory (Composite Key) or else we can create a new primary key column.

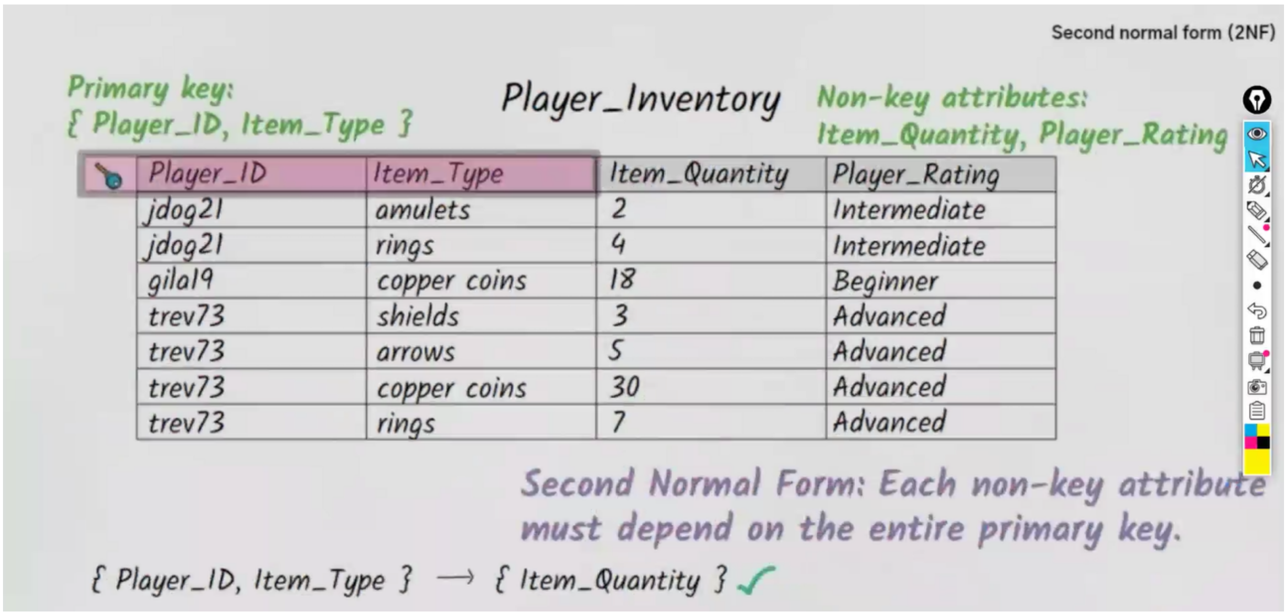
2. 2nd Normal Form
   Insertion, Updation & Deletion Anamoly



Second normal form

Player_Inventory

Player_ID	Item_Type	Item_Quantity	Player_Rating
jdog21	amulets	2	Intermediate
jdog21	rings	4	Intermediate
gila19	copper coins	18	Beginner
trev73	shields	3	Advanced
trev73	arrows	5	Advanced
trev73	copper coins	30	Advanced
trev73	rings	7	Advanced



Second normal form (2NF)

Primary key:
{ Player_ID, Item_Type }

Player_Inventory

Non-key attributes:
Item_Quantity, Player_Rating

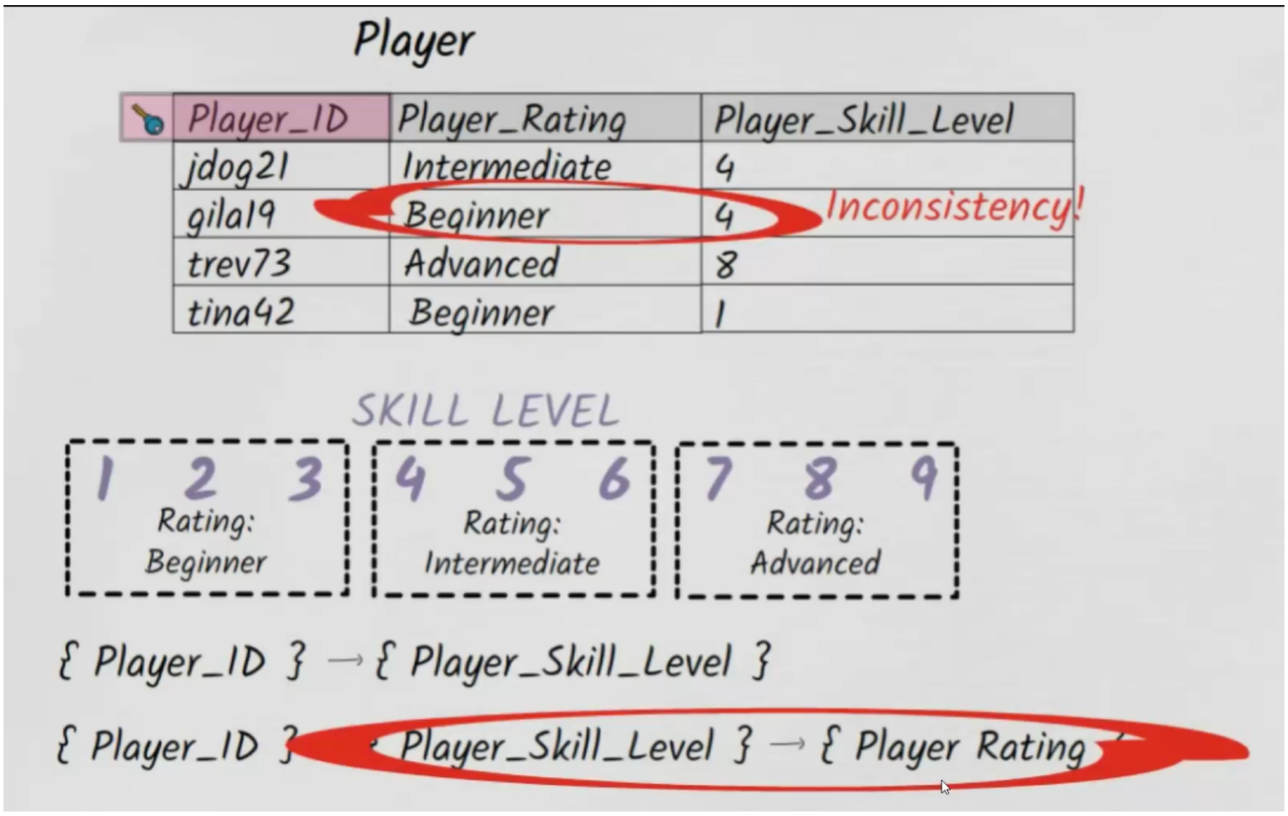
Player_ID	Item_Type	Item_Quantity	Player_Rating
jdog21	amulets	2	Intermediate
jdog21	rings	4	Intermediate
gila19	copper coins	18	Beginner
trev73	shields	3	Advanced
trev73	arrows	5	Advanced
trev73	copper coins	30	Advanced
trev73	rings	7	Advanced

Second Normal Form: Each non-key attribute must depend on the entire primary key.

{ Player_ID, Item_Type } $\longrightarrow$ { Item_Quantity } ✓

Note - No partial Dependency (Non-key Attribute must depend on the entire Primary Key)

3. 3rd Normal Form
   Lesser updates, More safer data.



Player

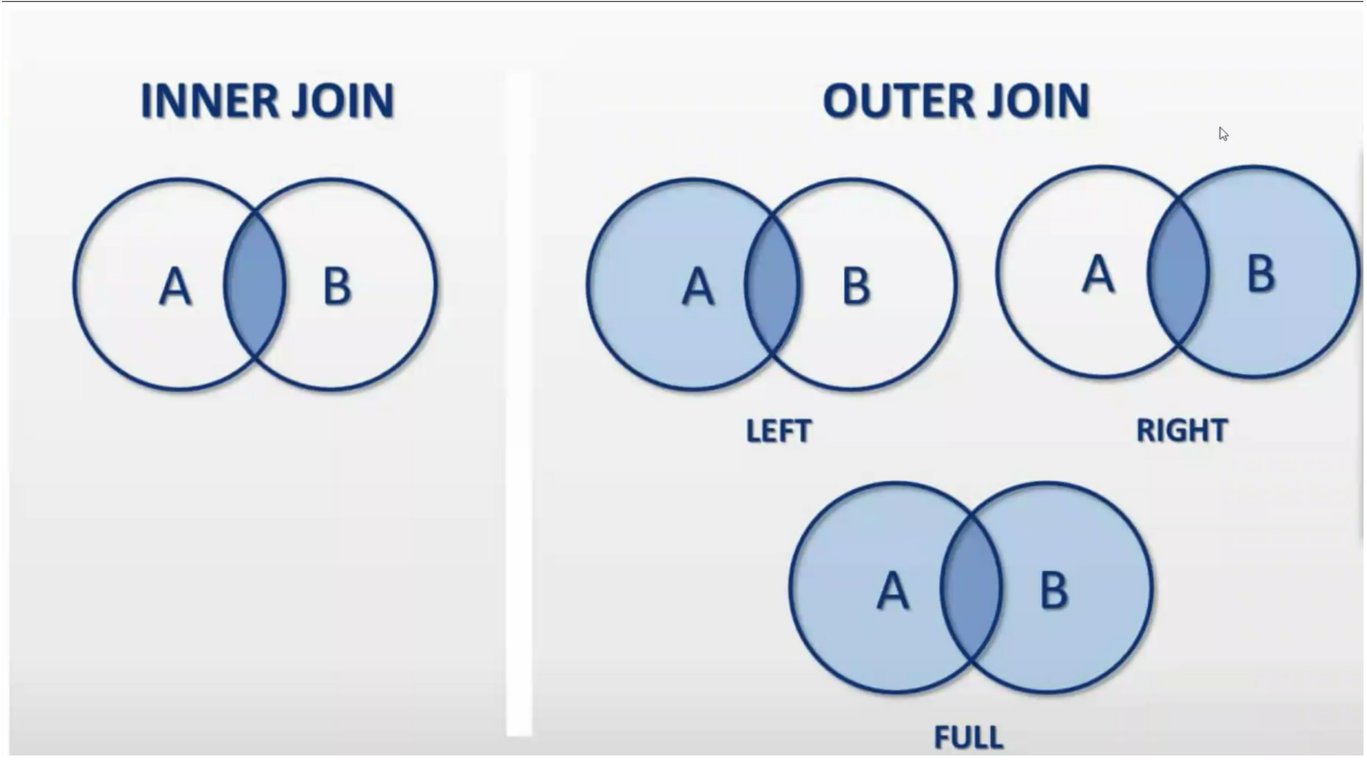
Player_ID	Player_Rating	Player_Skill_Level
jdog21	Intermediate	4
gila19	Beginner	4
trev73	Advanced	8
tina42	Beginner	1

$\{ Player\_ID \} \rightarrow \{ Player\_Skill\_Level \}$

$\{ Player\_ID \} \rightarrow \{ Player\_Skill\_Level \} \rightarrow \{ Player\ Rating \}$

(3.5 NF)BCNF - Every attribute should depend on Key attribute

JOINS



```
-- Syntax/Formula for join
-- A JOIN B ON Pk = Fk;
```

Why Joins required?

- For safety we are dividing the tables(Normalization), to fetch the data from other tables we are joining the tables(De-Normalization).

Types of JOINS

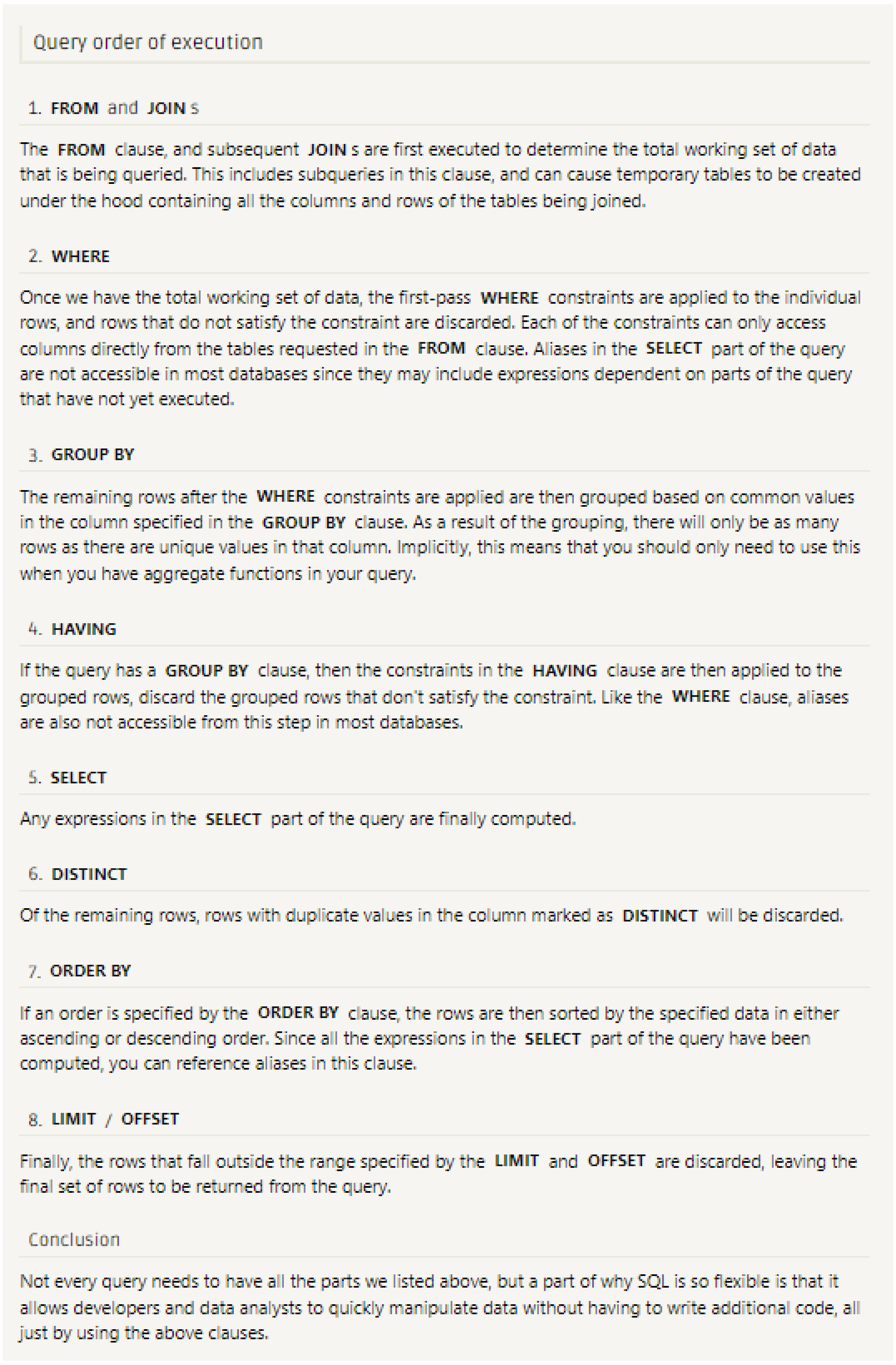
1. Inner Join
2. Outer Join
    - Left Outer Join
    - Right Outer Join
    - Full Join

Aggregates

Aggregation nothing but summarization

Clue - If there is every x in the question then GROUP BY x comes into picture. It is used to drill down to the next level

Query Order of Execution



Query order of execution

1. **FROM** and **JOIN** s

The **FROM** clause, and subsequent **JOIN** s are first executed to determine the total working set of data that is being queried. This includes subqueries in this clause, and can cause temporary tables to be created under the hood containing all the columns and rows of the tables being joined.

2. **WHERE**

Once we have the total working set of data, the first-pass **WHERE** constraints are applied to the individual rows, and rows that do not satisfy the constraint are discarded. Each of the constraints can only access columns directly from the tables requested in the **FROM** clause. Aliases in the **SELECT** part of the query are not accessible in most databases since they may include expressions dependent on parts of the query that have not yet executed.

3. **GROUP BY**

The remaining rows after the **WHERE** constraints are applied are then grouped based on common values in the column specified in the **GROUP BY** clause. As a result of the grouping, there will only be as many rows as there are unique values in that column. Implicitly, this means that you should only need to use this when you have aggregate functions in your query.

4. **HAVING**

If the query has a **GROUP BY** clause, then the constraints in the **HAVING** clause are then applied to the grouped rows, discard the grouped rows that don't satisfy the constraint. Like the **WHERE** clause, aliases are also not accessible from this step in most databases.

5. **SELECT**

Any expressions in the **SELECT** part of the query are finally computed.

6. **DISTINCT**

Of the remaining rows, rows with duplicate values in the column marked as **DISTINCT** will be discarded.

7. **ORDER BY**

If an order is specified by the **ORDER BY** clause, the rows are then sorted by the specified data in either ascending or descending order. Since all the expressions in the **SELECT** part of the query have been computed, you can reference aliases in this clause.

8. **LIMIT** / **OFFSET**

Finally, the rows that fall outside the range specified by the **LIMIT** and **OFFSET** are discarded, leaving the final set of rows to be returned from the query.

Conclusion

Not every query needs to have all the parts we listed above, but a part of why SQL is so flexible is that it allows developers and data analysts to quickly manipulate data without having to write additional code, all just by using the above clauses.

WHERE vs HAVING

WHERE checks for the condition in row wise whereas HAVING is used to more drill down the problem after grouping the table with a condition.