# Navigating SQL Interviews (Part I)

**Key points to remember when facing SQL interviews:**

**Master Aggregation Functions**
- Be comfortable using aggregate functions like SUM, AVG, COUNT, MIN, and MAX.
- Understand how to use GROUP BY and HAVING clauses effectively.

**Know Your Joins**
- Understand various types of joins (INNER, LEFT, RIGHT, FULL) and when to use each.
- Be able to explain how joins work and the differences between them.

**Understand the Basics**
- Ensure a solid grasp of fundamental SQL concepts such as SELECT statements, joins, aggregations, and filtering.

**Optimize Queries**
- Demonstrate proficiency in writing efficient SQL queries.
- Understand indexing, query execution plans, and how to optimize query performance.

**Handle NULL Values**
- Understand how SQL treats NULL values in expressions and comparisons.
- Be able to use IS NULL and IS NOT NULL appropriately.

**Practice, Practice, Practice !!**
- Work on sample databases and real-world scenarios to reinforce your skills.
- Familiarize yourself with different SQL dialects (e.g., MySQL, PostgreSQL, SQL Server) if possible.

**Show Problem-Solving Skills**
- Approach SQL problems methodically, breaking them down into smaller steps.
- Demonstrate your problem-solving skills by thinking logically and strategically.

**Be Clear and Concise**
- Articulate your thoughts clearly when explaining SQL concepts or solving problems.
- Use simple and precise language to communicate your answers.

**Ask Clarifying Questions**
- Don't hesitate to ask for clarification if you don't understand a question.
- Seek additional context to ensure you're addressing the interviewer's concerns effectively.

**Practice SQL Coding**

- Be prepared to write SQL queries on a whiteboard, paper, or in a text editor during the interview.
- Practice writing syntactically correct SQL code without relying heavily on autocomplete features.

**Review Past Work**

- Be ready to discuss SQL-related projects or experiences listed on your resume.
- Highlight any achievements or challenges you faced while working with SQL.

**Stay Calm and Confident**

- Maintain composure during the interview, even if you encounter difficult questions.
- Confidence in your abilities will reassure the interviewer of your competence.

The subsequent questions are commonly posed during interviews for roles in data analysis and data science. The complexity level fluctuates based on the interviewer's preferences and the candidate's responses to the questions. This repository serves as a valuable resource for promptly reviewing pertinent topics and ensuring readiness for the interview process.

# SQL and Types of SQL Commands

**What are the different types of SQL commands?**

SQL commands are divided into five categories:

1. Data Query Language (DQL)
2. Data Definition Language (DDL)
3. Data Manipulation Language (DML)
4. Data Control Language (DCL)
5. Transaction Control Language (TCL)

**What is DQL?**

- Data Query Language (DQL) is a subset of SQL used to query and retrieve data from a database.
- The primary command in DQL is `SELECT`, which is employed to fetch data from tables, allowing users to specify the columns to retrieve and the conditions for filtering the data.
- DQL enables data extraction and analysis without modifying the data itself.

**What does DDL stand for? Give an example.**

- DDL stands for Data Definition Language.
- It includes SQL commands used to define and modify the structure of database objects like tables, indexes, and schemas.
- Examples include CREATE, ALTER, DROP.

**What is DML?**

- DML (Data Manipulation Language) includes commands such as INSERT, UPDATE, DELETE which are used to manipulate data in a database.

**What is the purpose of DCL commands?**

- The purpose of Data Control Language (DCL) commands is to control access to data in the database.
- DCL commands, such as `GRANT` and `REVOKE`, are used to grant or revoke permissions to users or roles, thereby managing security and access rights.
- For example, `GRANT SELECT ON employees TO user1` allows `user1` to perform `SELECT` queries on the `employees` table.

**Explain TCL.**

- TCL stands for Transaction Control Language.
- It includes SQL commands that manage transactions within a database to ensure data integrity and consistency.
- The primary TCL commands are `COMMIT`, which saves all changes made during the current transaction, and `ROLLBACK`, which undoes all changes since the last `COMMIT`, restoring the database to its previous state.

# Data Types

**Name some common numerical data types in SQL.**

- INT, FLOAT, DOUBLE, DECIMAL.

**What data type would you use for a large integer value?**

- BIGINT.

**Name some common string data types in SQL.**

- VARCHAR, CHAR, TEXT

**What is the difference between CHAR and VARCHAR?**

- `CHAR` and `VARCHAR` are both used to store string data in SQL, but they differ in storage and usage.

- `CHAR` is a fixed-length data type, meaning it always reserves the specified number of characters, padding with spaces if necessary.
- In contrast, `VARCHAR` is a variable-length data type, storing only the actual characters entered, up to the defined limit, without additional padding.
- `CHAR` is more efficient for storing data of consistent length, while `VARCHAR` is more space-efficient for data of varying lengths.

**What is the difference between DATETIME AND TIMESTAMP?**

| DATETIME | TIMESTAMP |
|---|---|
| DATETIME usually has a wide range of years but less precision. | TIMESTAMP does not have a wide range of years but is more precise, often down to a fraction of seconds. |
| DATETIME values are stored without time zone offset. | TIMESTAMP stores value with time zone offset or in UTC (Coordinated Universal Time). So, it is suitable for handling time conversions according to time zones. |

# DQL

**Explain order of execution of SQL.**

1. FROM
2. JOIN
3. WHERE
4. GROUP BY
5. HAVING
6. SELECT
7. DISTINCT
8. ORDER BY
9. LIMIT / OFFSET (if applicable)

**What is the difference between WHERE and HAVING?**

- The `WHERE` and `HAVING` clauses in SQL are both used to filter records, but they serve different purposes.
- The `WHERE` clause filters individual rows based on specified conditions before any groupings are made, and it is applied to non-aggregated columns.
- In contrast, the `HAVING` clause filters groups after the `GROUP BY` clause has been applied, and it is used with aggregate functions.
- Essentially, `WHERE` is for row-level filtering, and `HAVING` is for group-level filtering.
- For instance, `WHERE` might filter employees with a salary over 50000, while `HAVING` could then filter departments that have more than 5 such employees.

**What is the use of group by?**

- The GROUP BY clause in SQL is used to group rows that have the same values in specified columns into summary rows. It's particularly useful for performing aggregate functions, such as SUM, COUNT, AVG, MIN, or MAX, on groups of data.
- Essentially, GROUP BY allows you to condense rows with similar attributes into summary rows, enabling you to analyze data at different levels of granularity and gain insights into patterns or trends within the dataset.

**How does the UNION operator work?**

- UNION: The Union clause/operator is used to combine the results of two or more SELECT Statements with Identical columns without returning any duplicate rows.
- There should be the same number of columns in both SELECT statements.
- The column names from the first SELECT statement in the UNION operator are used as the column names for the result set

**What is the difference between UNION and UNION ALL?**

- UNION returns the common members from the result of two select queries. The common members appear only once in the resultant set.
- UNION ALL returns the common members in the result of two select queries without removing duplicates in the result.

**Does MySQL support INTERSECT or MINUS operators?**

- MySQL DOES NOT support INTERSECT operators. However, you can emulate the INTERSECT operator using JOINS.

**What is the result of the IFNULL(exp1, exp2) function?**

- If exp1 is null then the function returns exp2. Otherwise it returns exp1.

**What is the use of the NULLIF(exp1, exp2) function?**

- If exp1 is the same as exp2 then the function returns null. Otherwise it returns exp1.

**What is the use of the str_to_date function?**

- It is used to convert date in string format into standard format of date. Only after conversion, we can use functions such as adddate, subdate.

**What is the use of a CAST function?**

- The CAST() function converts a value (of any type) into the specified datatype.
- SELECT CAST(150 AS CHAR)

**What is the use of COALESCE? How is it different from IFNULL?**

- COALESCE fetches the first non-null entry in the series of variables.
- COALESCE (p_email_id, sec_email_id, gurdian_email_id, friend_email_id) will return the first non-null entry in the series of variables mentioned.
- IFNULL is used to return the first non-null entry in case of only two variables.

**Why is it advisable to use the WHERE clause before GROUP BY?**

- It becomes less computationally expensive if the records are already filtered based on primary business entities such as month, department, etc., before performing grouping operations.

# DDL

**What is the difference between DROP and TRUNCATE?**

- When using `DROP`, the entire table, including its structure and data, is permanently deleted from the database. Due to its irreversible nature, executing a `DROP` command typically requires elevated permissions and careful consideration, as it effectively eliminates all traces of the table.
- On the other hand, `TRUNCATE` is designed to remove all rows from a table while preserving its structure. This command swiftly deletes data without affecting the table's schema, making it a faster and less resource-intensive alternative to `DROP`.
- While both commands effectively clear a table's contents, `DROP` is a more drastic measure suited for situations where the table itself needs to be eliminated, whereas `TRUNCATE` is ideal for quickly purging data while retaining the table structure for future use.

**What is the difference between CHANGE and MODIFY?**

- Change command can be used to rename a column of a table and change its datatype.
- Modify command can be used to change its datatype only.

# DML

**What are DML commands in SQL?**

- DML stands for Data Manipulation Language, and it includes SQL commands like INSERT, UPDATE, DELETE, and SELECT.
- These commands are used to manipulate data stored in database tables.

**What is the purpose of the INSERT statement in SQL?**

- The INSERT statement is used to add new rows of data into a table.
- It allows you to specify the values to be inserted for each column or retrieve values from another table using a SELECT statement.

**How does the UPDATE statement work in SQL?**

- The UPDATE statement is used to modify existing data in a table.
- It allows you to change the values of one or more columns for specific rows based on specified conditions using a WHERE clause.

**Explain the role of the DELETE statement in SQL.**

- The DELETE statement is used to remove one or more rows from a table.
- It allows you to specify conditions to determine which rows to delete using a WHERE clause. If no WHERE clause is specified, all rows in the table are deleted.

**What is the difference between the DELETE and TRUNCATE commands?**

- DELETE is a DML command used to remove specific rows from a table, allowing the use of a WHERE clause to specify criteria for deletion.
- TRUNCATE is a DDL command that removes all rows from a table without logging individual row deletions, resulting in faster performance but without the ability to specify conditions for deletion.

**How can you use DML commands to ensure data integrity in a database?**

- DML commands can be used with constraints such as PRIMARY KEY, FOREIGN KEY, and CHECK constraints to enforce data integrity rules.
- For example, you can use INSERT, UPDATE, and DELETE statements to ensure that only valid data is stored in the database tables.

# Subqueries

**What is a subquery in SQL?**

- A subquery is a nested query within another SQL statement, such as SELECT, INSERT, UPDATE, or DELETE.
- Subqueries can return a single value, a single row, multiple rows, or even an entire result set, depending on the context and usage.

**Explain the difference between a correlated and a non-correlated subquery.**

- A non-correlated subquery is independent of the outer query and can be executed separately.
- In contrast, a correlated subquery depends on the outer query and is executed once for each row processed by the outer query.
- Correlated subqueries typically involve a performance overhead compared to non-correlated subqueries.

**How can you use a subquery in the WHERE clause?**

- Subqueries in the WHERE clause are used to filter rows based on a condition evaluated in the subquery.
- For example, you can use a subquery to find all employees whose salary exceeds the average salary in the department.
- The result of the subquery is evaluated against each row in the outer query's result set to determine inclusion.

**How do you use a subquery to perform a comparison in SQL?**

- Subqueries can be used to compare values in different tables or within the same table.
- For instance, you can use a subquery to find all customers who have placed orders exceeding a certain amount.
- The subquery returns the necessary values, which are then compared using comparison operators like >, <, =, etc., in the outer query.

**Explain the concept of a correlated subquery with an example.**

- A correlated subquery references columns from the outer query, creating a dependency between the inner and outer queries.
- For example, to find all employees whose salary exceeds the average salary in their department, you would use a correlated subquery.
- The subquery calculates the average salary for each department, and the outer query filters employees based on this correlated condition.

**What is the difference between a single-row and a multi-row subquery?**

- A single-row subquery returns only one row of results, typically used in scenarios where you expect only one result.
- In contrast, a multi-row subquery returns multiple rows of results and can be used to compare values across multiple rows in a table.

- The choice between single-row and multi-row subqueries depends on the specific requirements of the query.

**Explain the concept of a nested subquery.**

- A nested subquery is a subquery that is nested within another subquery.
- This allows for complex filtering and comparison logic by combining multiple levels of subqueries.
- Each nested subquery is executed sequentially, with the results of the innermost subquery used as input for the next outer subquery.

**How do you use a subquery to perform an EXISTS or NOT EXISTS check?**

- Subqueries can be used with the EXISTS and NOT EXISTS operators to check for the existence or absence of rows that meet certain criteria.
- For example, you can use a subquery with EXISTS to find all employees who have at least one order in the Orders table.
- The EXISTS operator returns true if the subquery returns any rows, and false otherwise.

**What are the advantages of using subqueries in SQL?**

- Subqueries provide a powerful mechanism for expressing complex logic in a concise and readable manner.
- They enable you to perform operations that would otherwise require multiple queries or complex joins.
- Subqueries also enhance the flexibility and adaptability of SQL queries by allowing dynamic referencing of data from different tables.

**Explain types of subqueries in short.**

<u>Outer reference: Subquery is written in FROM clause</u>

A Sub-query is executed independently and its records are referred to as a derived table in the FROM clause of a Main Query.

```
SELECT A.Acct_Num,
       Sub_T.Tran_Amount
FROM   ACCOUNT A ,
       (
         SELECT Acct_Num,
                Tran_Amount
         FROM Transaction
         Where Tran_amount > 23000
       )  Sub_T
WHERE A.Acct_Num = Sub_T.Acct_Num
```

Subquery for comparison: Subquery is written in WHERE clause with single row operator

```
SELECT
    Acct_Num,
    Tran_Amount,
    Tran_Date
FROM
    TRANSACTION
WHERE Tran_Date = ( SELECT MAX(event_dt )
                    FROM MESSAGE
                    WHERE   Event = 'Holiday'
                  )
```

Subquery for comparison: Subquery is written in WHERE clause with multiple row operator

```
SELECT A.Acct_Num,
       A.Balance,
       A.Acct_type,
        A.Acct_status
FROM ACCOUNT A
WHERE A.Acct_Num IN
(
 SELECT Acct_Num FROM Transaction
 WHERE  Channel = 'ATM withdrawal'
)
```

Subquery for existence: Subquery is written in WHERE EXIST clause

In subquery, ‘yes’ is a constant value. But, internally, the subquery returns true or false when the record is available.

```
SELECT A.Acct_Num,
          A.Balance  ,
      A.acct_type,
          A.acct_status
FROM   ACCOUNT A
WHERE Exists (SELECT 'yes'
              FROM Transaction T
              WHERE T.Acct_Num = A.Acct_Num)
```

Nested Subquery: Data from multiple tables

```
SELECT    *
FROM CUSTOMER
WHERE Cust_Id IN (SELECT Cust_Id
                  FROM ACCOUNT
                  WHERE Acct_Num IN
                         (SELECT Acct_Num
                          FROM Transaction
                          WHERE Tran_Date > (SELECT MAX(Event_dt )
                                             FROM Message)
                          )
                 )
```

WITH clause plays a major role in complex queries when there is a need for calling the subqueries multiple times.

Always use the WITH clause when there is a need to invoke the subquery multiple times in different parts of the query.

```
WITH    I_RATE AS
         (SELECT  Acct_type, rate FROM Interest
     )
SELECT A.Acct_num, I_RATE.Acct_type,
I_RATE.rate
FROM ACCOUNT  A
JOIN I_RATE
ON A.Acct_type = I_RATE.Acct_type
```

# JOINS

**Explain: Self join**

When we want to compare the records with the value from the same table, self joins are used.

**What is a natural join? Write advantages and limitations.**

Advantages:

- A Natural JOIN maps the rows implicitly among common columns in both the tables defined in the FROM clause.
- It is best to use when there is a need for joining with all of the common columns between the two tables and retrieve a full set of columns from both the tables.

- Developers need not be aware of the columns that are used for Joining the tables.

Limitations:

- All the common columns should have unique data without duplicates.
- Natural joins cannot handle NULL values in Joining key columns since it matches the column values implicitly by SQL and are not written in the SQL queries.

**What is the difference between Equi join and natural join?**

- Joining key columns are explicitly specified in equi-join, whereas, in case of natural join, we do not have to specify joining key columns. The query searches for common column names with the same datatype.
- Equi join is a type of inner join where we have an equality (=) operator in the join condition.
- Inner join can have equality (=) and other operators (like <,>,<>) in the join condition.

**What are the advantages of joins over subqueries?**

| Joins | Subqueries |
|---|---|
| More efficient for large dataset | Less efficient for large dataset |
| Easy to understand the logic in a query | Difficult to understand |
| More flexibility and control performing aggregations or grouping operations | Less flexibility and control while aggregations or grouping |
| Joins allow you to handle duplicate data more effectively using techniques like DISTINCT. | There's a higher chance of returning duplicate data, especially if the subquery is correlated. |
| JOIN query performs the calculation in the main SELECT query. | Calculations can be performed in the subquery and returned as a single value. |

# Constraints

**What are constraints in SQL?**
Constraints are rules applied to columns to enforce data integrity.

**Name some common types of constraints.**
PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK.

**What does the PRIMARY KEY constraint do?**
It ensures that a column (or a combination of columns) uniquely identifies each row in a table.

**What is the purpose of the UNIQUE constraint?**
It ensures that all values in a column are unique.

**Explain the difference between PRIMARY KEY and UNIQUE KEY.**

Primary Key:

- Ensures that each value in the primary key column(s) is unique.
- Does not allow `NULL` values.
- Uniquely identifies each row in a table.
- A table can have only one primary key.

Unique Key:

- Ensures that each value in the unique key column(s) is unique.
- Allows `NULL` values, but each `NULL` value must be unique (some databases may allow multiple `NULL`s, but they are still considered distinct).
- Enforces uniqueness for specific columns, but does not necessarily uniquely identify each row.
- A table can have multiple unique keys.

## Explain following keys in SQL.

· Candidate key ● Primary key ● Foreign key ● Unique key ● Alternate key

**Super key :**

- Super key is a superset of all keys.
- It can be any single column or combination of columns which can uniquely identify the records in a table.
- It may contain extraneous attributes.

**Candidate key :**

- Minimal super keys are candidate keys.
- Candidate key can be any column or a combination of columns that can qualify as a unique key in the database.
- There can be multiple candidate keys in one table.
- Candidate key can have a null value.

**Primary key:**

- It is a special case of candidate keys.
- It should be unique and does not allow null value.
- There is only one primary key declared in a table.

**Foreign key:**

- Foreign keys are the columns of a table that points to the primary key of another table.

**Alternate key:**

- An alternate key is a candidate key that is not considered as a primary key.

**Unique key:**

- An attribute or a set of attributes to uniquely record in a table.
- This is similar to a primary key but can contain a null value.

**Composite key:**

- Any key with more than one attribute is a composite key.

**What is the difference between a candidate key and primary key?**

- A candidate key can have null entries but the primary key cannot.
- A table can have multiple candidate keys but there can be only one primary key for each table.
- Primary key is a special case of candidate key.

**What is the purpose of foreign key constraints?**

- Referential integrity is a database concept that ensures the consistency and accuracy of data relationships between tables.
- It is enforced through foreign key constraints, ensuring that values in a column (the foreign key) referencing another table (the primary key) always point to valid rows in that referenced table. This means that any action (insert, update, or delete) that affects the referenced table will be restricted or cascaded to maintain consistency.
- For example, if a row with a primary key value is deleted, any rows referencing that value via foreign keys will also be affected, either by preventing the deletion or by cascading the deletion to maintain integrity.
- In essence, referential integrity guarantees that relationships between tables are valid and maintain data consistency throughout the database.

**What is the CHECK constraint?**

- The CHECK constraint ensures that all values in a column meet a specific condition.

**What is the DEFAULT constraint?**

- The DEFAULT constraint provides a default value for a column if no value is specified during an INSERT operation.

## WINDOW FUNCTIONS

**What is a window function in SQL, and how does it differ from regular aggregate functions?**

- Window functions perform calculations across a set of table rows related to the current row, without collapsing the rows into a single result.
- Regular aggregate functions group the result set into a single output row for each group of rows.

**Explain the components and purpose of the `OVER` clause in window functions.**

- The `OVER` clause defines the window or set of rows that the window function operates on. It can include `PARTITION BY` to divide the result set into partitions, `ORDER BY` to order rows within each partition, and window frame clauses (`ROWS` or `RANGE`) to limit the rows within the partition.

**What is the `PARTITION BY` clause in window functions, and why is it used?**

- The `PARTITION BY` clause divides the result set into partitions to which the window function is applied.
- Each partition is processed separately by the window function, allowing calculations like running totals or ranks to be reset for each partition.

**How does the `ORDER BY` clause within the `OVER` clause affect window functions?**

- The `ORDER BY` clause within the `OVER` clause specifies the order of rows within each partition.
- It is essential for functions like `ROW_NUMBER()`, `RANK()`, `LAG()`, and `LEAD()`, as it determines the sequence in which calculations are performed.

**Describe the difference between `ROW_NUMBER()`, `RANK()`, and `DENSE_RANK()` functions.**

- `ROW_NUMBER()` assigns a unique sequential integer to each row within a partition.
- `RANK()` assigns ranks to rows within a partition, leaving gaps for ties.

- `DENSE_RANK()` assigns ranks without gaps for ties, ensuring consecutive rank numbers.

**What is the purpose of the `LAG()` and `LEAD()` functions in SQL?**

- The `LAG()` function accesses data from a previous row in the same result set, while the `LEAD()` function accesses data from a subsequent row.
- They are useful for comparing values in different rows without using self-joins.

**Explain how the `NTILE()` function works and provide an example scenario where it might be useful.**

- The `NTILE()` function divides rows in an ordered partition into a specified number of approximately equal groups or buckets.
- It is useful in scenarios like distributing tasks or data into equal parts, such as dividing sales data into quartiles.

**What is the difference between `FIRST_VALUE()` and `LAST_VALUE()` functions?**

- `FIRST_VALUE()` returns the first value in an ordered partition, while `LAST_VALUE()` returns the last value in an ordered partition.
- They are used to fetch specific values from the beginning or end of the partition.

**How does the `CUME_DIST()` function calculate cumulative distribution, and when would you use it?**

- The `CUME_DIST()` function calculates the cumulative distribution of a value within a partition by determining the proportion of rows with values less than or equal to the current row's value.
- It is useful for percentile calculations and statistical analysis.

**What is the `PERCENT_RANK()` function, and how does it differ from `RANK()`?**

- `PERCENT_RANK()` calculates the relative rank of a row as a percentage of the total number of rows, returning a value between 0 and 1.
- Unlike `RANK()`, which assigns integer ranks, `PERCENT_RANK()` provides the rank as a fraction, useful for understanding the relative standing of rows.

**Discuss the `NTH_VALUE()` function and a scenario where it might be applied.**

- The `NTH_VALUE()` function returns the value of the nth row in an ordered partition.
- It is useful in scenarios where specific positional data is needed, such as retrieving the second-highest sales amount within each region.

**How do `ROWS` and `RANGE` clauses within the `OVER` clause affect window functions?**

- The `ROWS` clause defines a physical set of rows relative to the current row, while the `RANGE` clause defines a logical range of rows based on value comparisons.
- They control the window frame for functions like moving averages and cumulative sums.

**What are the performance considerations when using window functions in SQL queries?**

- Window functions can be resource-intensive as they may require sorting and additional computation over large datasets.
- Proper indexing and query optimization techniques are crucial to maintain performance, especially for large result sets.

**Can window functions be used in `WHERE`, `HAVING`, or `GROUP BY` clauses? Explain why or why not.**

- Window functions cannot be used directly in `WHERE`, `HAVING`, or `GROUP BY` clauses because they operate on the result set after these clauses have been processed.
- They are typically used in the `SELECT` or `ORDER BY` clauses.

**What are some common use cases for window functions in real-world SQL applications?**

- Common use cases include calculating running totals, moving averages, ranking rows, finding gaps in data, performing year-over-year comparisons, and generating cumulative distributions.
- They simplify complex analytical queries and provide insights without requiring multiple subqueries or joins.

# Data Integrity

**What is the importance of data integrity?**

- Data integrity is necessary to ensure that the data is consistent, accurate, reliable and relevant (CARR).
- It is required to ensure smooth flow of business without losses helping in making good predictions.

**What will happen if data integrity is not achieved?**

- Information loss
- Incorrect predictions
- Storage space requirement increases

**What are the levels of integrity?**

- **Row level integrity:** Every row must be uniquely identified with the help of primary key.
- **Column level integrity:** Datatype of all the entries in a column should be same.
- **Referential integrity**: It ensures consistency of records between two related tables. It establishes parent-child relationships between two different tables using - FOREIGN KEY and PRIMARY KEY. (Entries in the child table cannot be updated without making changes in primary key data in the parent table. If entries in the parent table are updated then at the same time, foreign key data in the child table has to be updated simultaneously.)

**What are the ACID properties?**

In a transaction processing, database application follows the standard four properties to ensure data integrity.

1. **Atomicity**: Complete the data transaction by 100% otherwise retain the original status of transaction. (It makes sure that the transaction is either complete **or** reversed.)
2. **Consistency**: Consistency of data ensures the data is not leaked in any case of success or failure of transactions. (It makes sure that the transaction is complete **and** successful.)
3. **Isolation**: Database application ensures every transaction is individual, secured and is hidden from other transactions. It avoids un-ethical hacking.
4. **Durability**: Data should be persistent (should be same) after before and after a transaction.

**Explain ACID properties with an example.**

**Atomicity:**

- Example: A pays B $1000.

- Explanation: Atomicity ensures that all parts of a transaction are completed successfully or none are. If $1000 is debited from A's account, it must be credited to B's account for the transaction to be successful. If any part of this transaction fails (e.g., crediting B's account), the entire transaction is rolled back, and both A and B retain their original balances. This ensures that partial transactions do not occur, maintaining data integrity.

**Consistency:**

- Example: If A pays B $1000, but B receives only $500.

- Explanation: Consistency ensures that a transaction brings the database from one valid state to another. A transaction that leaves B with only $500 instead of $1000 violates consistency rules. Consistency guarantees that the database remains in a consistent state before and after the transaction, following all predefined rules, such as balance constraints.

**Isolation:**

- Example: A pays B $1000. Meanwhile, Edward has a balance of $5000 and writes two checks for $3000 and $2700.

- Explanation: Isolation ensures that concurrent transactions do not interfere with each other. A's payment to B should be isolated such that the debit from A and credit to B are not visible to other transactions until completed. Similarly, Edward's two checks should not impact each other, even though his balance would be insufficient if both were processed simultaneously. Isolation prevents such anomalies by ensuring transactions are executed in isolation.

**Durability:**

- Example: Edward has an account balance of $4500 and swipes $2000 for a purchase. The network goes down before the order is completed.

- Explanation: Durability ensures that once a transaction is committed, it remains so, even in the event of a system failure. If Edward's transaction is committed, his balance should reflect the debit of $2000, even if the network fails afterward. The database must maintain the state of the data before and after the transaction to ensure that no loss occurs, preserving data integrity.

Summary of ACID Properties:

-Atomicity: Ensures all-or-nothing transactions.

- Consistency: Guarantees that database rules are not violated.

- Isolation: Keeps transactions separate and unaffected by others.

- Durability: Ensures that committed transactions persist despite failures.

**List down the anomalies due to redundant data.**

· Insertion anomalies

· Updation anomalies

· Deletion anomalies

**<u>What is normalisation?</u>**

- Normalization is the process of organizing the data attributes with their relationships so that we can reduce the redundancy in data. It improves data integrity.
- Normalization minimizes the redundancy of data rows.
- Normalization minimizes the dependency of columns.
- It eliminates the undesirable characteristics like Insertion, Update and delete of anomalies ( flaws).
- Normalization divides the larger table into the smaller tables and establishes entity-relationship among those tables.

**1st NF**

- A table is said to be in 1st normalization form if the table has scalar atomic value (multiple values are not written by comma).
- Values in columns should have the same business term.
- All column names should be unique.

**2NF**

- Table should not contain partial dependency.
- A table is said to be in 2nd normalization form, if all non-key columns and independent attributes are fully functional on primary key.

**3NF**

- No transition dependency exists means if a table has columns a, b, c, d where a is primary key, and columns b, c, d are non-prime attributes then tables should be as follows: ab, ac, ad.  It should not be ab, bc, cd.
- Bigger table is broken down to a smaller table so that it is easy to fetch data by joining the tables.

**4NF (Boyce Codd Normalization Form)**

- For every dependency A >>> B, the 4NF form ensures that A has to be the super key of that table.
- In simpler terms, BCNF ensures that there are no partial dependencies of non-prime attributes on candidate keys.