1. Types of commands and their examples.

Tuesday, July 30, 2024 6:19 AM

1. Data Definition Language (DDL)
CREATE: Defines new database objects like tables.
ALTER: Modifies existing database structures.
DROP: Deletes database objects.
TRUNCATE: Removes all records from a table.
2. Data Manipulation Language (DML)
INSERT: Adds new records to tables.
UPDATE: Modifies existing records.
DELETE: Removes records from tables.
3. Data Query Language (DQL)
SELECT: Retrieves data from tables.
4. Transaction Control Language (TCL)
COMMIT: Saves all changes made during the transaction.
ROLLBACK: Reverts all changes made during the transaction.
SAVEPOINT: Sets a point within a transaction to which you can roll back.
RELEASE SAVEPOINT: Removes a savepoint.
5. Data Control Language (DCL)
GRANT: Gives users access privileges.
REVOKE: Removes access privileges.
6. Data Administration Commands
BACKUP DATABASE: Creates a backup of the database.
RESTORE DATABASE: Restores the database from a backup.

2. What is Normalization and denormalization?

Tuesday, July 30, 2024 9:43 AM

Normalization
Normalization is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. Its goal is to reduce data anomalies and improve data integrity. Normalization involves dividing a database into two or more tables and defining relationships between them. The process follows a series of steps called normal forms, each with specific rules to ensure database efficiency.

Denormalization
Denormalization is the process of combining tables to reduce the complexity of joins and to improve database read performance. This involves adding redundant data to tables to achieve faster query performance by reducing the number of joins. Denormalization is often used in read-heavy environments where read performance is more critical than write performance.

3. Explain 1NF, 2NF, 3NF.

Tuesday, July 30, 2024 9:49 AM

First Normal Form (1NF)
A table is in 1NF if:

All columns contain atomic (indivisible) values.
Each column contains values of a single type.
Each column has a unique name.
The order in which data is stored does not matter.

Second Normal Form (2NF)
A table is in 2NF if:

It is in 1NF.
All non-key attributes are fully functionally dependent on the primary key.

Third Normal Form (3NF)
A table is in 3NF if:

It is in 2NF.
All non-key attributes are not transitively dependent on the primary key.

4. Share use case where you had to do denormalization in database..

Tuesday, July 30, 2024 9:56 AM

Example of Denormalization:
We had three normalized tables: Sensors, Readings, and Events.

```
CREATE TABLE Sensors (
   SensorID INT PRIMARY KEY,
   SensorName VARCHAR(50),
   SensorType VARCHAR(50)
);

CREATE TABLE Readings (
   ReadingID INT PRIMARY KEY,
   SensorID INT,
   ReadingValue FLOAT,
   ReadingTime TIMESTAMP,
   FOREIGN KEY (SensorID) REFERENCES Sensors(SensorID)
);

CREATE TABLE Events (
   EventID INT PRIMARY KEY,
   SensorID INT,
   EventDescription VARCHAR(255),
   EventTime TIMESTAMP,
   FOREIGN KEY (SensorID) REFERENCES Sensors(SensorID)
);


CREATE TABLE SensorData (
   SensorID INT,
   SensorName VARCHAR(50),
   SensorType VARCHAR(50),
   ReadingValue FLOAT,
   ReadingTime TIMESTAMP,
   EventDescription VARCHAR(255),
   EventTime TIMESTAMP
);
```

5. What is primary key and foreign key?

Tuesday, July 30, 2024 10:14 AM

A primary key is a unique identifier for each record in a database table. It ensures that each record can be uniquely identified and is used to establish and enforce entity integrity. A primary key must have the following characteristics:

Unique: No two records can have the same primary key value.
Non-null: A primary key cannot contain NULL values.
Immutable: The values in a primary key should not change over time.

Foreign Key
A foreign key is a column or a set of columns in one table that establishes a link between the data in two tables. It is used to maintain referential integrity by ensuring that the value in one table must match a value in another table, typically the primary key of the other table.

6. what is alternate and candidate key?

Tuesday, July 30, 2024 10:16 AM

Candidate Key
A candidate key is a column, or a set of columns, in a database table that can uniquely identify any record in that table. A table can have multiple candidate keys, each of which can serve as a unique identifier. Each candidate key must have the following properties:

Unique: No two rows can have the same value(s) for the candidate key.
Non-null: The candidate key cannot contain NULL values.
Minimal: No subset of the candidate key can be unique; it is the smallest set of attributes that can uniquely identify a record.

Alternate Key
An alternate key is any candidate key that is not chosen as the primary key. In other words, alternate keys are candidate keys that could have been selected as the primary key but were not.

7. What are window functions?

Tuesday, July 30, 2024 10:25 AM

Window functions in SQL perform calculations across a set of table rows that are somehow related to the current row. Unlike aggregate functions, which return a single result for a group of rows, window functions can return multiple rows for each partition or frame.

8. Explain Ranking Functions? GIven a small table , write the output.

Tuesday, July 30, 2024 10:31 AM

Ranking Functions in SQL
Ranking functions in SQL assign a rank or row number to each row within a partition of a result set. These functions are useful for tasks like ordering results and determining the position of a row within a set.

Example and Output:
Assume we have the following Employees table:

EmployeeID	Name	Salary
1	Alice	50000
2	Bob	60000
3	Charlie	60000
4	David	55000
5	Eve	70000

Using RANK():

```
SELECT EmployeeID, Name, Salary,
    RANK() OVER (ORDER BY Salary DESC) AS Rank
FROM Employees;
```

Output:

EmployeeID	Name	Salary	Rank
5	Eve	70000	1
2	Bob	60000	2
3	Charlie	60000	2
4	David	55000	4
1	Alice	50000	5

9. Types of Joins? With example and use case. All the number of records return and exact records.

Tuesday, July 30, 2024 11:10 AM

Joins in SQL are used to combine rows from two or more tables based on a related column between them. Here are the main types of joins:

INNER JOIN
LEFT JOIN (LEFT OUTER JOIN)
RIGHT JOIN (RIGHT OUTER JOIN)
FULL OUTER JOIN
CROSS JOIN
SELF JOIN

1. INNER JOIN
Description: Returns only the rows where there is a match in both tables.

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

2. LEFT JOIN (LEFT OUTER JOIN)
Description: Returns all rows from the left table and matched rows from the right table. If no match is found, NULLs are returned for columns from the right table.

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

3. RIGHT JOIN (RIGHT OUTER JOIN)
Description: Returns all rows from the right table and matched rows from the left table. If no match is found, NULLs are returned for columns from the left table.

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName
FROM Employees
RIGHT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

4. FULL OUTER JOIN
Description: Returns all rows when there is a match in one of the tables. Rows that do not have a match in either table will have NULLs for missing columns.

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName
FROM Employees
FULL OUTER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

5. CROSS JOIN
Description: Returns the Cartesian product of the two tables, i.e., all possible combinations of rows.

```
SELECT Employees.EmployeeID, Employees.Name, Departments.DepartmentName
FROM Employees
CROSS JOIN Departments;
```

6. SELF JOIN
Description: Joins a table with itself. Useful for querying hierarchical data or comparing rows within the same table.

```
SELECT e1.EmployeeID AS Emp1ID, e1.Name AS Emp1Name, e2.EmployeeID AS Emp2ID, e2.Name
AS Emp2Name
FROM Employees e1
JOIN Employees e2 ON e1.EmployeeID <> e2.EmployeeID;
```

10. Use case when self-join is required.

Tuesday, July 30, 2024 11:28 AM

A self join is used to join a table with itself. This is particularly useful in scenarios where there is a need to compare rows within the same table, such as hierarchical data, comparisons between rows, and finding related records within the same table.

11. What is subquery?

Tuesday, July 30, 2024 11:31 AM

A subquery (or inner query) is a query nested within another query, such as a SELECT, INSERT, UPDATE, or DELETE statement, or inside another subquery. Subqueries are used to perform operations that need multiple steps or to break down complex queries into more manageable parts.

```
SELECT Name
FROM Employees
WHERE DepartmentID IN (SELECT DepartmentID FROM Departments WHERE
Location = 'New York');
```

12. What is corelated subquery?

Tuesday, July 30, 2024 11:34 AM

A correlated subquery is a subquery that references columns from the outer query. Because of this reference, the subquery is evaluated once for each row processed by the outer query. This makes correlated subqueries different from regular subqueries, which are executed only once and then provide a result to the outer query.

```
SELECT e1.Name, e1.Salary, e1.DepartmentID
FROM Employees e1
WHERE e1.Salary > (SELECT AVG(e2.Salary) FROM Employees e2
WHERE e2.DepartmentID = e1.DepartmentID);
```

13. What is CTE?

Tuesday, July 30, 2024 11:37 AM

A Common Table Expression (CTE) is a temporary result set defined within the execution scope of a SELECT, INSERT, UPDATE, or DELETE statement. It is particularly useful for simplifying complex queries, improving readability, and making recursive queries easier to write and understand.

```
WITH cte_name (column1, column2, ...)
AS (
   -- CTE definition
   SELECT ...
)
-- Using the CTE
SELECT ...
FROM cte_name
WHERE ...;
```

14. What is derived table?

Tuesday, July 30, 2024 11:41 AM

A derived table is a subquery that appears in the FROM clause of a SQL query. It is essentially a temporary table that is created and used within the context of a single query. Derived tables are useful for breaking down complex queries into simpler, more manageable parts and can often help improve the readability and structure of a query.

```
SELECT DepartmentID, TotalSales
FROM (
   SELECT DepartmentID, SUM(Amount) AS TotalSales
   FROM Sales
   GROUP BY DepartmentID
) AS DepartmentSales
WHERE TotalSales > 200;
```

15. Find third highest employee based on salary?

Tuesday, July 30, 2024 11:44 AM

```
SELECT Name, Salary
FROM Employees
ORDER BY Salary DESC
LIMIT 1 OFFSET 2;
```

16. Find third highest employee based on salary per department?

Tuesday, July 30, 2024   11:47 AM

```
WITH RankedSalaries AS (
   SELECT
      EmployeeID,
      Name,
      Salary,
      DepartmentID,
      DENSE_RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary
DESC) AS SalaryRank
   FROM Employees
)
SELECT
   EmployeeID,
   Name,
   Salary,
   DepartmentID
FROM RankedSalaries
WHERE SalaryRank = 3;
```

17. How to find duplicate values in a single column?

Tuesday, July 30, 2024 11:49 AM

```
SELECT *
FROM Employees
WHERE Name IN (
   SELECT Name
   FROM Employees
   GROUP BY Name
   HAVING COUNT(*) > 1
);
```

18. How to find duplicate values in a multiple column?

Tuesday, July 30, 2024 11:50 AM

```
SELECT *
FROM Employees
WHERE (Name, Salary) IN (
   SELECT Name, Salary
   FROM Employees
   GROUP BY Name, Salary
   HAVING COUNT(*) > 1
);
```

19. What are ACID properties?

Tuesday, July 30, 2024 5:13 PM

Consider a scenario where a user wants to transfer money from Account A to Account B.

Atomicity: The transaction involves debiting Account A and crediting Account B. Atomicity ensures that either both operations are completed or none.
Consistency: The total amount of money in the system remains unchanged. If the transfer amount is $100, Account A is debited by $100 and Account B is credited by $100, maintaining overall consistency.
Isolation: If another transaction is trying to access Account A or Account B during the transfer, isolation ensures that this concurrent transaction does not interfere and the final state is consistent as if the transactions were performed one after the other.
Durability: Once the transfer is completed and the transaction is committed, the changes (the debit and the credit) are permanently recorded in the database. Even if the system crashes right after the commit, the changes will not be lost.

Summary
Atomicity: All-or-nothing execution of transactions.
Consistency: Ensures data integrity before and after a transaction.
Isolation: Transactions are executed independently.
Durability: Committed transactions are permanently recorded.

20. Diff between union and union all

Tuesday, July 30, 2024 5:27 PM

UNION
Purpose: Combines the results of two or more SELECT statements into a single result set and removes duplicate rows.

UNION ALL
Purpose: Combines the results of two or more SELECT statements into a single result set, including all duplicate rows.

21. Diff between primary key and unique key

Tuesday, July 30, 2024 5:32 PM

Primary Key: Ensures uniqueness and non-nullability, creating a unique identifier for table rows. Only one primary key is allowed per table.

Unique Key: Ensures uniqueness but allows nullability. Multiple unique keys can be defined per table to enforce uniqueness on multiple columns.

22. Diff between truncate and delete

Tuesday, July 30, 2024 5:34 PM

TRUNCATE: Fast, minimal logging, resets identity columns, does not activate triggers, used for removing all rows.
DELETE: Slower, full logging, does not reset identity columns, activates triggers, used for removing specific rows based on conditions.

23. Diff between having and where

Tuesday, July 30, 2024 5:36 PM

WHERE: Filters rows before any grouping; cannot use aggregate functions directly.

HAVING: Filters groups after grouping; can use aggregate functions.

24. SQL query execution order.

Tuesday, July 30, 2024 5:38 PM

FROM: Identifies the source tables.
ON: Specifies join conditions.
JOIN: Combines data from multiple tables.
WHERE: Filters individual rows.
GROUP BY: Groups rows for aggregation.
HAVING: Filters groups based on aggregate conditions.
SELECT: Chooses the columns to display.
DISTINCT: Removes duplicate rows.
ORDER BY: Sorts the result set.
LIMIT / OFFSET: Limits the number of rows returned or sets the starting point.

25. What are indexes? Types of Indexes and their differences.

Tuesday, July 30, 2024 6:39 PM

Indexes are special database objects that enhance the speed of data retrieval operations on a database table at the cost of additional storage space and potential decrease in performance during data modification operations (INSERT, UPDATE, DELETE). They work similarly to indexes in books, allowing the database to find rows more quickly and efficiently.

Clustered Index:

Purpose: Determines physical order.
Count per Table: One.
Usage: Primary keys, range queries.

Non-Clustered Index:

Purpose: Separate index structure.
Count per Table: Multiple.
Usage: Improve query performance on columns other than the clustered index.

Unique Index:

Purpose: Enforce uniqueness.
Count per Table: Multiple.
Usage: Ensure unique column values.

Full-Text Index:

Purpose: Optimize text searches.
Count per Table: Depends on RDBMS.
Usage: Text search applications.

Spatial Index:

Purpose: Optimize spatial data queries.
Count per Table: Depends on RDBMS.
Usage: GIS, location-based services.

Bitmap Index:

Purpose: Efficient for low cardinality.
Count per Table: Multiple (depends on RDBMS).
Usage: Data warehousing.

26. What is surrogate key? Give example where you used it and how.

Tuesday, July 30, 2024 6:44 PM

A surrogate key is an artificially generated key used to uniquely identify each record in a database table. It is not derived from the application data and is typically implemented as an auto-incrementing integer or a universally unique identifier (UUID). Surrogate keys are used instead of natural keys (which are derived from the actual data) to simplify data management and ensure that each record has a unique, immutable identifier.

Characteristics of Surrogate Keys
Uniqueness: Guarantees that each record in the table has a unique identifier.
Immutability: Once assigned, the surrogate key does not change, even if other data in the record changes.
System-Generated: Typically auto-incremented integers or UUIDs generated by the database system.
Independence: Not derived from application data, which makes them immune to changes in business logic or data values.

27. Ways to optimize sql query?

Tuesday, July 30, 2024 6:45 PM

Optimizing SQL queries is crucial for improving the performance of database operations, especially when dealing with large datasets. Here are several strategies and techniques to optimize SQL queries:

1. Use Indexes
Create Indexes: Indexes can significantly speed up data retrieval operations.
Use Appropriate Index Types: Choose the right type of index (e.g., clustered, non-clustered) based on your query patterns.

2. Optimize Joins
Use Appropriate Join Types: Select the most efficient join type for your query (INNER JOIN, LEFT JOIN, RIGHT JOIN, etc.).
Ensure Proper Indexing on Join Columns: Index the columns used in join conditions.

3. Avoid Select *
Select Only Necessary Columns: Reduce the amount of data retrieved by selecting only the columns you need.

4. Use WHERE Clauses Effectively
Filter Data Early: Use WHERE clauses to filter rows as early as possible in the query execution process.

5. Use Proper Data Types
Match Data Types: Ensure that the data types of columns and variables match to avoid unnecessary conversions.
Use Efficient Data Types: Choose data types that are appropriate for the data being stored and that minimize storage space.

6. Avoid Functions on Indexed Columns
Do Not Use Functions on Indexed Columns in WHERE Clauses: This can prevent the use of indexes.

7. Use JOINs Instead of Subqueries
Rewrite Subqueries as Joins: In some cases, joins can be more efficient than subqueries.

8. Limit the Number of Rows Returned
Use LIMIT/OFFSET: Fetch only the necessary number of rows

9. Analyze Query Execution Plans
Use EXPLAIN/EXPLAIN ANALYZE: Analyze the query execution plan to identify bottlenecks.

10. Optimize ORDER BY and GROUP BY Clauses
Index Columns Used in ORDER BY and GROUP BY: Index the columns to speed up sorting and grouping operations.

11. Partition Large Tables
Use Table Partitioning: Partition large tables to improve performance and manageability

12. Cache Repeated Queries
Use Query Caching: Cache the results of frequently executed queries to reduce database load.

13. Optimize Insert and Update Operations
Batch Inserts and Updates: Batch multiple insert or update operations into a single query.

14. Avoid Unnecessary Columns in SELECT Clause
Select Only Necessary Columns: Retrieve only the columns needed for the operation.

15. Use EXISTS Instead of IN
Use EXISTS for Subqueries: Use EXISTS for better performance in certain cases

16. Denormalize When Necessary
Balance Normalization and Performance: In some cases, denormalization can improve query performance by reducing the number of joins needed.