DATA ALCHEMY SHAPING PERCEPTIVITY THROUGH THE LANGUAGE OF SQL

Preface

Structured Query Language, or SQL, is a particular kind of programming language used for handling and modifying relational databases. Its main thing is to give druggies a standardised and effective system of interacting with databases so they may manipulate the data that's stored in them. SQL is used to interact with relational databases and to administer them. Data querying, database structure update and revision, fresh data insertion, and data omission are among its main operations. With several database operation systems, including MySQL, PostgreSQL, Microsoft SQL Garçon, and Oracle, SQL offers a standardised interface for communication.

ORIGIN AND HISTORY OF SQL

1970	• Origins at IBM, SEQUEL developed by Chamberlin and Boyce
1986	• ANSI SQL-86 • Official standardization of SQL
1989	• SQL-89 • Additional features, widespread adoption
1992	• SQL-92 (SQL2) • Outer joins, character string support, integrity constraints
1999	• SQL-99 (SQL3) • Object-relational features, recursive queries
2003	• SQL:2003 • Window functions, improved LOB support, triggers, stored procedures
2008	• SQL:2008 • MERGE statement, enhanced temporal support
2011	• SQL:2011 • Clarifications and improvements
2016	• SQL:2016 • JSON support, polymorphic table functions, temporal table enhancements
2019	• SQL:2019 • SQL/JSON path language, SQL/MDA support, analytics function improvements

PURPOSE

- **Database Management:**
  Database Management Relational database systems are managed using SQL. Data is arranged in tables with rows and columns using a relational database. These tables and their connections can be created, altered, and modified by druggies using SQL.

- **Data Querying:**
  Data Querying Getting particular data out of a database is one of SQL's primary uses. The SQL SELECT statement allows druggies to produce queries that filter, sort, and aggregate data according to certain criteria. As a result, it's simple to prize precious data from big datasets.

- **Modification of Data:**
  Revision of Data SQL offers commands for barring gratuitous data, modifying records that formerly live, and adding new data. As a result, druggies can save the applicability and delicacy of the data kept in a database.

- **Database Schema Definition:**
  Database Schema Definition druggies can define and alter a database's structure using SQL statements written in the Data Definition Language(DDL). To maintain data integrity, this entails defining data types, establishing constraints, and creating and modifying tables.

- **Constraints and Data Integrity:**
  Constraints and Data Integrity SQL allows for the operation of a number of constraints, including check, unique, and primary keys. These limitations guarantee the thickness and delicacy of the data stored in the database.

- **Security and Access Control:**
  Security and Access Control Data Control Language(DCL) statements in SQL give directors the capability to manage database access. This include granting or removing boons, conforming warrants, and making sure that only individualities with authorization are suitable to carry out specific tasks.

- **Transaction Management:**
  Sale operation Sequences of one or further SQL statements that are carried out as a single unit are supported by SQL. Through the successful completion of all statements or the rolling reverse of changes in the event of an error, deals guarantee the thickness of the database.

All effects considered, SQL is a strong and standardised tool for working with relational databases. It's a vital language for database directors, inventors, and judges because of its capability to query and recoup data, alter database armature, save data integrity, and manage access control.

Significance of SQL

Relational database operation requires the use of Structured Query Language(SQL), which is essential for managing, modifying, and querying databases. Some crucial points that's important in SQL are :

1. **Data Management:** Data Management In relational databases, SQL is used to organise and manage massive quantities of data. In order to guarantee thickness and integrity, it offers a standardised system for defining, modifying, and reacquiring data.

2. **Data Retrieval:** Data Retrieval SQL queries enable druggies to gain particular data from databases. This eliminates the need to manually search through large databases in order to recoup material information.

3. **Modification of Data:** Revision of Data druggies can add, update, and remove records from a database using SQL. This allows for changes in the underpinning data, icing the database stays correct and over to date.

4. **Data Integrity:** Data Integrity SQL has capabilities that support data integrity, including primary keys, foreign keys, and constraints. These systems put guidelines that stop inconsistent or inaccurate data from being entered into the database.

5. **Data security:** Data security SQL enables database directors to manage access to sensitive data by supporting stoner authentication and authorization. This guarantees that particular data can only be seen or modified by those who are authorised.

6. **Database Design:** Database Design To produce and alter a database's structure, SQL is needed. It helps with the effective organisation of data by letting druggies make tables, connections, and indicators.

7. **Scalability:** Scalability SQL databases have the capacity to manage big datasets and expand to accommodate the requirements of expanding operations. Because of this, it can be used in a variety of surrounds, ranging from small- scale enterprise to enterprise- position systems.

8. **Compatibility:** The SQL language is extensively utilised across a range of database operation systems(DBMS), including extensively habituated options similar as Microsoft SQL Garçon, Oracle, PostgreSQL, and MySQL. Because of this standardisation, colourful database systems can be reciprocated and transported across them.

9. **Business Intelligence:** Business Intelligence The capability to decide significant perceptivity from data requires SQL. When used in confluence with business

intelligence technologies, it's constantly utilised for decision- making processes, reporting, and analytics.

10. **Online development:** Online development SQL databases are used by numerous online operations to store and recoup data stoutly. To construct dynamic and interactive websites, SQL is constantly used in confluence with computer languages like PHP, Python, or Java

A QUICK TOUR TO SQL BASICS

TYPES OF SQL COMMANDS

DDL (DATA DEFINITION LANGUAGE)	DML (DATA MANIPULATION LANGUAGE)	DCL (DATA CONTROL LANGUAGE)	TCL (TRANSACTION CONTROL LANGUAGE)	DQL (DATA QUERY LANGUAGE)
The SQL Data Definition Language (DDL) is in charge of specifying and controlling a database's structure	To interact with and manipulate data inside a database, utilise SQL's Data Manipulation Language (DML).	SQL's Data Control Language (DCL) is in charge of overseeing database security by limiting data access.	A database's transactions are managed using SQL's Transaction Control Language (TCL).	It is udes to highlight the quering capabilities of SQL, particularly through select command.
It includes commands as follows: 1. CREATE 2. ALTER 3. DROP.	It includes commands as follows: 1. SELECT 2. INSERT 3. DELETE 4. UPDATE	It includes commands as follows: 1. GRANT 2. REVOKE	It includes commands as follows: 1. ROLLBACK 2 .COMMIT 3. SAVEPOINT	It includes command as follows: 1. SELECT

COMMANDS IN SQL

1. **Create Command**

   This command is used to create a table. This is the basic SQL command to create a table which is important to perform various functions on it and to retrieve the required information from the table. So, this is the first step in SQL i.e. to create a table.

   Syntax :

   Create table table_name(column1 datatype not null, column2 datatype not null, ……);

   Where,

   Create table table_name : it create table of required name
   Column1, column2,…. : the name of the column which we want to add in the table.
   Datatype : the type of the data i.e. integer, float, char, etc.

   Example:

   If we want to create a table named 'products' with product_id, product_name, product_price as its attributes then the query for the same will be

   Query:

```
Create table products(
product_id int not null,
product_name varchar(30) not null,
product_price float not null );
```

2. **Select Command**

   A basic SQL (Structured Query Language) command for retrieving data from one or more database tables is the SELECT statement. It may be expanded with more clauses to filter and arrange the results, and it has a simple syntax.
   Syntax for select command:

   Select column1, column2, …….
   from table_name;

   where,

   Select: The query's starting keyword that retrieves data.
   column1, column2, ….: The columns from the given table that you wish to obtain.
   from: A keyword designating the table that the data should be retrieved from.
   table_name: The table name that has the required columns in it.

Example:
Lets take an example of student table which contains Roll_no, Name, Department as columns. Now suppose we want to retrieve Name and Department columns only
from the student table then we can make use of select command.

Student table		
Roll_no	Name	Department
1	Ram	DS
2	Siya	CT
3	Sam	DS
4	Sita	CT

Solution:

Query:

```
select Name, Department
from student;
```

Output:

Name	Department
Ram	DS
Siya	CT
Sam	DS
Sita	CT

3. Where Clause

In SQL, the WHERE clause is used to filter the rows that a SELECT operation returns according to predetermined criteria. You can only obtain the rows that match the given criteria.

Syntax for where clause :

```
Select column1, column2, .....
from table_name
where condition;
```

where,

Select: The query's starting keyword that retrieves data.
column1, column2,...: The columns from the given table that you wish to obtain.
from: A keyword designating the table that the data should be retrieved from.
table_name: The table name that has the required columns in it.
where: clause that lays out the criteria for data filtering.
condition: The prerequisite that a row has to satisfy in order to be part of the result set.

Example:

Lets take an example of 'student' table which contains Roll_no, Name, Department as columns. Now suppose we want to retrieve Name and Roll_no of students whose department is 'DS' from the student table then we can make use of where clause.

Student table		
Roll_no	Name	Department
1	Ram	DS
2	Siya	CT
3	Sam	DS
4	Sita	CT

Solution:

Query:

```
 Select Name, Roll_no
from student
 where Department='DS';
```

Output:

Roll_no	Name
1	Ram
3	Sam

4. **Order by Clause**
The SQL ORDER BY clause. A query's result set can be sorted using the ORDER BY clause according to one or more columns. The output can be arranged using it in either descending (DESC) or ascending (ASC) order.

Syntax for order by clause:

```
Select column1, column2, …..
from table_name
where condition
order by column1[asc/desc], column[asc/desc], ……;
```

where,

Select: The query's starting keyword that retrieves data.
column1, column2,…: The columns from the given table that you wish to obtain.
from: A keyword designating the table that the data should be retrieved from.
table_name: The table name that has the required columns in it.
where: clause that lays out the criteria for data filtering.

condition: The prerequisite that a row has to satisfy in order to be part of the result set.
Order by: sort the result based on the specified column either ascending or descending order.

Example:
Lets take an example of 'student' table which contains Roll_no, Name, Department as columns. Now suppose we want to retrieve Name and Roll_no of students whose department is 'CT' from the student table and the name should be in ascending order then for that we can make use of where clause followed by order by clause.

Output:

Student table		
Roll_no	Name	Department
1	Ram	DS
2	Siya	CT
3	Sam	DS
4	Sita	CT

Solution:

Query:

```
 Select Name, Roll_no
from student
 where Department='CT'
 order by name asc;
```

Roll_no	Name
4	Sita
2	Siya

AGGREGATE FUNCTIONS IN SQL

Aggregate functions in SQL are used to calculate values on sets and produce a single result. These routines take a set of rows as input and output a single value. There are various aggregate functions used in SQL.

To understand these functions let's take an example.

Suppose there is a 'student' table which has following attribute i.e. Roll_no, Name, Marks.

And now we have to find the sum, average, minimum, maximum of the marks obtained then we will make use of aggregate functions as follows:

Roll_no	Name	Marks
1	Raghu	45
2	Adhya	
3	Minal	67
4	Sita	23
5	Ram	80

1. **SUM():**
   This function calculate the sum of all numeric values of the mentioned column.
   Syntax:
   Select sum(column_name) from table_name;

   Example:

   To calculate the sum of the marks of all student in the above example we use the aggregate function sum().

   Query:

   Select sum(Marks) from student;

   Output:

sum(Marks)
215

2. **AVERAGE():**
   This function calculate the average of all numeric values of the mentioned column. We can say average is nothing but to calculate the mean of all values in the column.
   Syntax:
   Select avg(column_name) from table_name;

   Example:

   To calculate the average of the marks of all student in the above example we use the aggregate function avg().

   Query:

   Select avg(Marks) from student;

   Output:

avg(Marks)
53.75

3. **MIN():**

This function is used to calculate the minimum value from the column. It return the smallest value from the column.

Syntax:

Select min(column_name) from table_name;

Example:

In the above example to find the minimum value from the marks column we can make use of min() function.

Query:

Select min(Marks) from student;

Output:

min(Marks)
23

4. **MAX():**

This function is used to calculate the minimum value from the column. It return the smallest value from the column.

Syntax:

Select min(column_name) from table_name;

Example:

In the above example to find the minimum value from the marks column we can make use of min() function.

Query:

Select min(Marks) from student;

Output:

min(Marks)
23

5. **COUNT():**

This function is used to count the number of rows in the table. In other words we can say it is used to give the number of non-null values in the tables.

Syntax:

Select count(*) from table_name;

Example:

In the above example in order to calculate the total number of rows in the table we can use count() function.

Query:

Select count(*) from student;

Output:

min(Marks)
23

UNDERSTANDING SQL JOINS

A join in SQL is a technique that enables you to merge rows from two or more tables together according to a shared column. Joining tables serves the function of retrieving and combining data from various tables in a database to satisfy a particular query or reporting need. Joins link similar data that is stored in separate tables, allowing you to generate more meaningful and complicated result sets.

In short we can say join is used to combine two or more tables based on common columns in the table.

Basic Syntax for join:

```
Select columns
from table_name1, table_name2
where table_name1.column=table_name2.column;
```

There are various types of joins so let's see through an example.

Example:

Suppose we have two tables 'Student' and 'Department' which has Id, Name, Marks as attributes in student table and Id, dep _no., dep_name as attributes in department table.

Student table		
Id	Name	Marks
1	Ram	55
2	Sita	67
3	Gita	64
4	Mona	87

Department table		
Id	dep_no	dep_name
3	101	DS
4	102	CT
5	103	DS
6	104	CT

Now preform various types of joins on these two tables as follows:

1. **Inner Join**
   When the required columns in two or more tables match, an INNER JOIN in SQL is used to get rows from those tables. Only the rows that meet the join requirement are returned.

Syntax:

```
Select columns
from table1
Inner join table2 on table1.column=table2.column;
```

Where,

Select: Choose which columns from the joined tables you wish to get by using the select command.

from table1: Indicates which table to join initially in the join process.
Table 2: Indicates the second table to be joined and that an INNER JOIN is desired.
on table1.column = table2.column: Indicates the join's condition. For the rows to be included in the result set, the columns that are being compared must have the same data type and have matching values.

Example:

In the above example we find the rows from student table which matches the rows in department table based on common column i.e. Id. For this we preform Inner Join.

Query:

```
Select * from Student
Inner join Department on Student.Id=Department.Id;
```

2. **Left Join OR Left Outer Join**
It returns all the rows from the left table i.e. table1 and the rows that matches from the right table i.e. table2. If there is no match then null values are return for columns from the right table.

Syntax:

```
Select columns
From table1
Left join table2 on table1.column=table2.column;
```

Where,

Select : Choose which columns from the joined tables you wish to get by using the select command.
from table1: Indicates which table is the first (left) in the join process.
left join table2: Identifies the second (right) table and declares that an left outer join is desired.
on table1.column = table2.column: Indicates the join's condition. For the rows to be included in the result set, the columns that are being compared must have the same data type and have matching values.

Example:

In the above example, the LEFT OUTER JOIN will be performed on the Id column. All rows from the 'Student' table will be included in the result set, and for each matching row in the 'Department' table, the corresponding department information will be included. If there is no match in the 'Department ' table for a particular employee, the columns from the 'Department' table will contain NULL values in the result set.

Query:

```
Select Roll_no, Name, Marks
From Student
Left join Department on Student.Id=Department.Id;
```

3. **Right Outer Join**
   It returns all the rows from the left table i.e. table1 and the rows that matches from the right table i.e. table2. If there is no match then null values are return for columns rom the right table.

Syntax:

```
Select columns
From table1
Right join table2 on table1.column=table2.column;
```

Where,
Select : Choose which columns from the joined tables you wish to get by using the select command.
from table1: Indicates which table is the first (left) in the join process.
right join table2: Identifies the second (right) table and declares that an right outer join is desired.
on table1.column = table2.column: Indicates the join's condition. For the rows to be included in the result set, the columns that are being compared must have the same data type and have matching values.

Example:
In the above example, the RIGHT OUTER JOIN will be performed on the Id column. All rows from the 'Department' table will be included in the result set, and for each matching row in the 'Student' table, the corresponding department information will be included. If there is no match in the 'Student' table for a particular employee, the columns from the 'Student' table will contain NULL values in the result set.

Query:

```
Select Roll_no, Name, Marks
From Student
Right  join Department on Student.Id=Department.Id;
```

4. **Full Outer Join**
   It return all the rows where there is match from both the left and right table. If there is no match then it returns null values for columns from the tables where there is match.

Syntax:

```
Select columns
From table1
Full outer join table2 on table1.column=table2.column;
```

Where,
Select columns : Choose which columns from the joined tables you wish to get by using the select command.
from table1: Indicates which table is the first (left) in the join process.
full outer join table2: Identifies the second (right) table and declares that a full outer join is desired.
on table1.column = table2.column: Indicates the join's condition. For the rows to be included in the result set, the columns that are being compared must have the same data type and have matching values.

Example:

In this example, the FULL OUTER JOIN is performed on the Id column. All rows from both the STUDENT and DEPARTMENT tables will be included in the result set. For matching rows, you get a combined result with columns from both tables. If there is no match in either the STUDENT or DEPARTMENT table for a particular department or student, the corresponding columns from the non-matching table will contain NULL values in the result set.

Query:

```
Select Roll_no, Name, Marks
From Student
Full outer join Department on Student.Id=Department.Id;
```

5. **Cross Join**
   The Cartesian product of two tables is produced by a SQL CROSS JOIN, sometimes referred to as a Cartesian Join. This indicates that it returns every conceivable set of rows from both tables in any combination.

Syntax:

```
Select columns
From table1
Cross join table2;
```

Where,

Select columns: The command "SELECT columns" tells the computer which columns to extract from the final Cartesian product.
from table1: Indicates which table to join initially in the join process.
cross join table2: Indicates that a cross join is desired and provides the second table.

Example:

In this example, the CROSS JOIN returns all possible combinations of colors and sizes. If there are three colours and four sizes, the result set will have a total of twelve rows, representing every combination of colour and size.

Query:

```
Select Roll_no, Name, Marks
From Student
Cross join Department
```

CONSTRAINTS

Within a relational database, constraints are essential for maintaining data integrity. By assisting in the enforcement of table relationships and rules, they help stop the entry of erroneous or inconsistent data.
Some common types of constraints are:

1. **Primary Key**
    - It guarantees that every entry in a table is distinct and not null.
    - It gives each record in the table a unique identity.
    - It aids in creating connections between tables.

Example :
```
Create table employee
(employee_id int Primary Key,
Employee_name varchar(20) not null);
```

2. **Foreign Key**
    - It creates a link based on a column between two tables.
    - It enforces the need for values from one table to exist in another, protecting referential integrity.
    - It maintains uniformity throughout linked tables.

Example:
```
Create table orders(
Order_id int Primary Key,
Customer_id int,
Order_date date,
Foreign Key(Customer_id) references Customer(Customer_id));
```

3. **Unique Key**
   - It guarantees the uniqueness of the values in a column (or a combination of columns).
   - It keeps data from being duplicated, ensuring data integrity.

Example:
```
Create table student(
Student_id int Primary Key,
Student_name varchar(20) not null,
Student_email varchar(30) Unique);
```

4. **Not Null**
   - It Verifies that there are no null values in a column.
   - It ensures that every record has a value for that column that is legitimate.

Example:
```
Create table products(
Product_id int Primary Key,
Product_name varchar(30) not null,
Price decimal(10,2) not null);
```

TRANSACTIONS : ACID PROPERTIES

In database management systems (DBMS), transactions are essential for maintaining data integrity and consistency. The attributes that transactions must possess in order to preserve data integrity are outlined by the ACID properties: atomicity, consistency, isolation, and durability.

1. **Atomicity:**
   A transaction is regarded as a single, indivisible unit of labour when it is atomic. The transaction's operations can either all be completed successfully or none at all.
   The database is kept in a consistent state by rolling the entire transaction back to its initial state if any portion of it fails.

2. **Consistency:**
   A transaction moving the database from one consistent state to another is guaranteed by consistency. Prior to and following the completion of a transaction, the database must adhere to a set of integrity constraints.
   To keep the database from going into an inconsistent state, all transactions that break integrity constraints are rolled back in full.

3. **Isolation:**
   The process of isolation makes sure that one transaction is carried out independently of the other transactions. The result should be the same whether or not numerous transactions are running simultaneously, just like if they were running serially.
   By averting conflicts and guaranteeing that every transaction sees an identical snapshot of the database, isolation helps preserve data integrity and prevents interference between transactions.

4. **Durability:**
   When a transaction is durable, it ensures that its consequences remain intact even in the event of future failures, including system crashes or power outages. Committed transaction modifications are kept in non-volatile memory, typically on disc, so they continue to exist in the event of a system failure. This is necessary to keep the database's long-term integrity intact.

ADVANTAGES OF SQL

Structured Query Language(SQL) provides colourful benefits for maintaining and manipulating relational databases

1. **Easy to Use:**
   - SQL is intended to be stoner-friendly and simple to learn.
   - Its declarative design enables druggies to concentrate on expressing what data they want rather than how to get it.

1. **Scalability:**
   - SQL databases can manage vast volumes of data and scale efficiently as the database size increases.
   - Indeed in huge datasets, well- optimized quests and indexing help to recoup data efficiently.

2. **Data integrity:**
   - SQL enforces data integrity restrictions similar as primary keys, foreign keys, and unique constraints, icing that data is accurate and dependable.
   - The ACID rates(Atomicity, thickness, insulation, and continuity) give a solid foundation for sale operation.

3. **Security:**
    - SQL databases have grainy access control, which enables directors to produce and manage stoner warrants at colourful situations.
    - Authentication and encryption features ameliorate data security.

4. **Interoperability:**
    - SQL is a standardised language that allows for interoperability across different database systems.
    - operations can snappily switch between SQL- biddable databases with minimum law changes.   .

5. **Data reclaimation and manipulation:**
    - SQL has robust querying capabilities, enabling druggies to get specific data or execute sophisticated aggregations and joins.
    - smut operations(produce, read, update, and cancel) give for effective data processing.

6. **Data Consistency:**
    - SQL deals insure that changes to the database are harmonious and reliable.
    - Rollback procedures allow you to return changes in the event of an error, icing data thickness.

7. **Data analysis:**
    - SQL can handle ad hoc queries, making it ideal for data processing and reporting.
    - Integration with business intelligence technologies enables enhanced analytics and decision- timber.

8. **Concurrency Control:**
    - SQL databases handle concurrent data access using features like locking, insulation situations, and sale control.
    - This allows multitudinous druggies to interact with the database contemporaneously without jeopardising data integrity.

9. **Backup & Recovery:**
    - SQL databases include tools for regular data backups and recovery, which cover against data loss or system failure.
    - Point- in- time recovery options allow you to restore databases to precise timestamps.

RECENTS TRENDS IN SQL

SQL (Structured Query Language) is a foundational and extensively used language for managing and modifying relational databases as of my most recent knowledge update in January 2022. It's possible that since then, new trends and advancements in the SQL environment have evolved. The following trends may still be applicable in 2022 and may develop further:

1. **SQL databases for analysis:**
   Analytical databases like Snowflake, Amazon Redshift, and Google BigQuery are using an increasing amount of SQL. Large dataset analysis and high-performance querying are made possible by these databases, which are designed with analytical workloads in mind.

2. **Integration of Machine Learning:**
   The integration of machine learning capabilities with SQL is becoming more and more popular. Data scientists can carry out machine learning tasks within the SQL environment with the help of SQL-based machine learning libraries and frameworks, such as SQL Server Machine Learning Services and BigQuery ML.

3. **Extensions for Graph Databases:**
   Graph database features are being directly integrated into SQL in certain relational databases. The need to rapidly query and analyse complex relationships in data is what's driving this development.

4. **Time Travel and Temporal Tables:**
   The use of temporal tables, which show how data has changed over time, has grown in popularity. Time-travel searches and working with temporal data have been made simpler by features added to databases such as SQL Server.

5. **JSON Assistance:**
   JSON data handling is now better supported by many relational databases. This is especially crucial since JSON is starting to be used as a standard data format for many different applications, such as web and mobile development.

6. **Improvements to SQL Standards:**
   The goal of ongoing SQL standard updates is to increase the language's functionality and compatibility with various database systems. New features and enhancements were introduced in SQL:2016 and SQL:2019.

7. **Cloud-Based Native SQL:**
   With the opportunity to leverage cloud platforms' scalability, flexibility, and managed services, SQL databases are becoming more and more cloud-native in design. Integration between serverless and containerised architectures with cloud-native databases is frequently smooth.

8. **Predictive Performance Optimisation:**
   More automated performance tweaking options are being added to SQL

databases. These consist of indexing, query optimisation, and other performance-related duties that the database system itself can handle.

9. **A Greater Focus on Security**
   Enhancing SQL databases' security features is a constant focus due to increased worries about data security. This covers auditing capabilities, access controls, and encryption.

10. **Orchestration and Containerisation:**
    An increasing number of SQL databases are being orchestrated with Kubernetes and deployed in containerised environments with technologies like Docker. More scalability and portability are made possible by this.

REAL LIFE APLLICATION OF SQL

Structured Query Language (SQL) is a powerful tool used for managing and manipulating relational databases.
Some of the real-life examples in which SQL is used are as follows:

1. **Retail Management System**
   - Scenario: It helps to track inventory and sales.
   - SQL query: To retrieve the total sales of a product from the sales table.

```
Select product_name, sum(quantity* unit_price) as total_sales
from sales
where product_id=’ABC123’
order by product_name;
```

2. **Human Resources Database**
   - SQL can be used in human resources database to manage employee information.
   - SQL query: For example it can be used to retrieve the names and positions of the employees hired in the last month.

```
Select employee_name, position
from employee
where hire_date>=’2024-03-01’ and hire_date<’2024-05-01’;
```

3. **Social Media Platforms**
   - It can be used for storing user data and posts.

- SQL query: For example if we want to retrieve the number of likes for a post.

```
Select post_title, count(user_id) as likes
From likes
Where post_id='123456'
Group by post_title;
```

4. **E-Commerce Website**
   - SQL can be used in E-Commerce for managing customer orders and shipping
   - SQL query: For example we can find all the pending orders along with customer details.

```
Select customer_name, order_id, order_date
From customers
join orders on customers.customer_id=orders.customer_id;
```

5. **Financial System**
   - For financial system, SQL can be used to record financial transactions.
   - SQL query: We can take an example of to find the total balance for a specific amount.

```
Select account_no, sum(amount) as total_balance
from transactions
where account_no='123456';
```

6. **Healthcare Database**
   - SQL can be used for storing records and medical history.
   - SQL query: for example, retrieving all appointments for a specific doctor.

```
Select patient_name, appointment_date
From appointments
Where doctor_id='AB123';
```

These are the few illustrations that shows how SQL may be used to retrieve, manipulate, and manage data within relational databases in a variety of contexts and industries. Other applications where SQL can be used are banking sector, hotel reservation system, CRM and so on. SQL is a flexible language that may be used for a wide range of activities, from basic data retrieval to intricate data administration and analysis.

DRAWBACKS OF SQL

1. **Complexity for Beginners**
   - For beginners, SQL syntax and database principles can be difficult.
   - It could be too much for beginners to comprehend and utilise SQL efficiently.

2. **Vendor-Specific Implementations**
   - The way SQL is implemented may differ throughout database management systems.
   - Because of these differences, writing portable, vendor-independent code can be challenging.

3. **Performance Concerns**
   - Slow query execution times might be caused by poorly optimised queries.
   - Performance problems may arise from inefficient database design and indexing.

4. **Scalability Challenges**
   - It is difficult to maintain scalability and performance when dealing with big datasets or heavy transaction volumes.
   - Scalability optimisation in SQL databases can be difficult.

5. **Limited support for Hierarchical Data**
   - Because relational databases are the primary focus of SQL, managing hierarchical data is less natural.
   - Even with advancements like Common Table Expressions, problems could still arise.

6. **Learning Curve for Advanced Features**
   - It's possible that more complex SQL capabilities like triggers and stored procedures are less obvious.
   - It will take more time and effort to become proficient with these features.

7. **Security Concerns**
   - One well-known security flaw is SQL injection.
   - Although hazards can be reduced with parameterized queries, security concerns must be understood.

8. **Limited Support for Unstructured Data**
   - Structured data is best suited for SQL databases.
   - NoSQL databases might be more appropriate for handling unstructured or semi-structured data.

9. **Lack of Standardization for Some Operations**
   - Challenges may arise from differences in how specific operations are implemented amongst databases.

- Adjustments could be necessary when migrating between different database platforms.

10. **Cost of Implementation and Maintenance**
    - Systems involving SQL databases might require a lot of resources to implement and maintain.
    - Licencing fees, necessary hardware, and continuous maintenance are among the costs.

CONCLUSION

To sum up, SQL (Structured Query Language) is an essential tool for relational database management and manipulation. Its value stems from its capacity to offer a standardised and effective means of interacting with databases, enabling users to easily define, query, and manipulate data. Because SQL guarantees data security, consistency, and integrity, it is essential for every organisation that uses structured data.

SQL remains relevant in the ever changing technology landscape for a number of reasons. First off, SQL remains at the forefront of data management due to the widespread use of relational databases across numerous industries. Furthermore, the expansion of big data and the rising need for analytics demonstrate SQL's versatility in managing sizable datasets and intricate queries. Furthermore, SQL is essential for database management in cloud environments as more and more enterprises adopt cloud computing.

To put it briefly, SQL's lasting significance comes from its fundamental position in relational database management, its flexibility in responding to changing data difficulties, and its ongoing relevance in the face of technology improvements, all of which contribute to its status as a necessary skill for professionals in a variety of industries.