Lecture 21

Database Processing
SQL and noSQL

Several programming languages and query languages are commonly used to interact with and process queries in databases and data warehouses. The choice of language often depends on the specific database management system (DBMS) or data warehouse being used. Here are some common languages for querying and processing data:

SQL (Structured Query Language):
Description: SQL is the standard language for managing and querying relational databases. It allows users to define, manipulate, and query data in a relational database.
Common Use: Used with relational database management systems (RDBMS) like MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, and others.

PL/pgSQL (Procedural Language/PostgreSQL):
Description: An extension of SQL that adds procedural programming features. It is specific to the PostgreSQL database system.
Common Use: Used for writing stored procedures and functions in PostgreSQL.

T-SQL (Transact-SQL):
Description: An extension of SQL developed by Microsoft. It includes additional procedural programming constructs and is used with Microsoft SQL Server.
Common Use: Used for writing stored procedures, triggers, and other procedural code in SQL Server.

PL/SQL (Procedural Language/SQL):
Description: Oracle's extension of SQL with procedural language features. It allows for the creation of stored procedures, functions, and triggers.
Common Use: Used with Oracle Database for writing procedural code.

MDX (Multidimensional Expressions):
Description: A query language for querying multidimensional databases, often associated with Online Analytical Processing (OLAP) systems.
Common Use: Used in conjunction with databases that support OLAP, such as Microsoft SQL Server Analysis Services and others.

DAX (Data Analysis Expressions):
Description: A formula language used in Power BI, Excel Power Pivot, and SQL Server Analysis Services. It is used for creating custom formulas and expressions.
Common Use: Commonly used in business intelligence and analytics scenarios.

CQL (Cassandra Query Language):
Description: A query language for Apache Cassandra, a NoSQL database. It is similar to SQL but tailored for the Cassandra data model.
Common Use: Used for querying and interacting with Apache Cassandra databases.

HiveQL:
Description: The query language for Apache Hive, a data warehouse infrastructure built on top of Hadoop. It provides SQL-like queries for data stored in Hadoop Distributed File System (HDFS).
Common Use: Used in the context of big data processing with Apache Hive.

Spark SQL:
Description: Part of the Apache Spark ecosystem, Spark SQL provides a programming interface for data manipulation using SQL-like queries. It allows users to query structured and semi-structured data.
Common Use: Used for querying data within Apache Spark applications.

KQL (Kusto Query Language):
Description: The query language used in Azure Data Explorer (Kusto), a service for exploring and analyzing large volumes of data quickly.
Common Use: Used for querying and analyzing data in Azure Data Explorer.

GraphQL:
Description: A query language for APIs that enables clients to request only the data they need. It provides a more flexible and efficient alternative to traditional REST APIs.
Common Use: Widely used in web development for client-server communication.

These languages serve different purposes, and the choice depends on the type of database or data warehouse, as well as the specific requirements of the application or analysis being performed.

SQL (Structured Query Language) and **NoSQL (Not Only SQL)** are two different types of database management systems, each with its own set of characteristics and use cases. Here are the key differences between SQL and NoSQL databases:

1. Data Model:
SQL: SQL databases are relational and use a structured schema, where data is organized into tables with predefined columns and relationships between tables.
NoSQL: NoSQL databases can be non-relational and offer various data models, including document-oriented, key-value pairs, wide-column stores, and graph databases. The schema is more flexible, allowing for dynamic and hierarchical data.

2. Schema:
SQL: SQL databases have a fixed schema, and any changes to the structure require altering the entire database.
NoSQL: NoSQL databases have a dynamic or schema-less structure, allowing for easy modification and adaptation to evolving data requirements.

3. Scalability:
SQL: Traditional SQL databases are typically scaled vertically by adding more resources (CPU, RAM) to a single server.
NoSQL: NoSQL databases are often designed for horizontal scalability, allowing for distributed and scalable architectures by adding more servers to a database cluster.

4. ACID Properties:
SQL: SQL databases adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring data integrity and transactional consistency.
NoSQL: NoSQL databases may relax ACID properties for improved performance and scalability. They often prioritize eventual consistency and partition tolerance over immediate consistency.

5. Query Language:
SQL: SQL databases use SQL as the standard query language for defining and manipulating relational data.
NoSQL: NoSQL databases use various query languages or APIs, depending on the specific type of database. Examples include MongoDB's query language for document stores or Cassandra Query Language (CQL) for wide-column stores.

6. Use Cases:
SQL: Well-suited for applications with complex relationships and structured data, such as financial systems, ERP (Enterprise Resource Planning) systems, and traditional business applications.
NoSQL: Well-suited for scenarios with large volumes of unstructured or semi-structured data, such as content management systems, real-time big data applications, and IoT (Internet of Things) platforms.

7. Schema Evolution:
SQL: Changes to the schema can be complex and may require significant planning and downtime for migration.
NoSQL: NoSQL databases allow for dynamic and flexible schema evolution, making it easier to adapt to changing data requirements without downtime.

8. Examples:
SQL: MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.
NoSQL: MongoDB (document store), Cassandra (wide-column store), Redis (key-value store), Neo4j (graph database).

9. Consistency Model:
SQL: Emphasizes strong consistency, ensuring that transactions are executed in a manner that preserves the integrity of the data.
NoSQL: Offers various consistency models, including eventual consistency, where data consistency is guaranteed at some point in time but not necessarily immediately.

It's important to note that the choice between SQL and NoSQL depends on specific project requirements, scalability needs, data structure, and the nature of the application. Some projects may even use a combination of both types of databases to meet different needs within the same system (polyglot persistence).

Since most query languages are extensions of SQL, it’s useful to see how SQL works in its basic structure.

Let’s look at some specific examples of SQL code and what it does.

Database Creation:
SQL starts with creating a database using the CREATE DATABASE statement.

```
CREATE DATABASE MyDatabase;
```

Table Creation:
Tables hold the data in SQL databases. They are created using the CREATE TABLE statement.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT,
    Department VARCHAR(50)
);
```

Data Insertion:
Data is inserted into tables using the INSERT INTO statement.

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, Age,
Department)
VALUES (1, 'John', 'Doe', 30, 'HR');
```

Data Retrieval (Simple Query):
Basic data retrieval is done using the SELECT statement.

```
SELECT FirstName, LastName FROM Employees;
```

Filtering Data (WHERE Clause):
Data can be filtered using the WHERE clause.

```
SELECT * FROM Employees WHERE Age > 25;
```

Sorting Data (ORDER BY Clause):
Sorting is achieved using the ORDER BY clause.

```
SELECT * FROM Employees ORDER BY LastName ASC;
```

Updating Data (UPDATE Statement):
Data can be updated using the UPDATE statement.

```
UPDATE Employees SET Department = 'Finance' WHERE EmployeeID = 1;
```

Deleting Data (DELETE Statement):
Data can be deleted using the DELETE statement.

```
DELETE FROM Employees WHERE Age < 30;
```

Joining Tables (INNER JOIN):
Tables can be joined to combine data using the JOIN clause.

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

Other kinds of joins are also possible, such as left joins, right joins and outer joins.

Subqueries:
Subqueries are queries nested inside other queries.

```
SELECT FirstName, LastName
FROM Employees
WHERE DepartmentID IN (SELECT DepartmentID FROM Departments WHERE
DepartmentName = 'IT');
```

Examples:

Simple SELECT:
```
SELECT FirstName, LastName FROM Employees;
```

Filtering with WHERE:
```
SELECT * FROM Employees WHERE Age > 25;
```

Sorting with ORDER BY:
```
SELECT * FROM Employees ORDER BY LastName ASC;
```

Joining Tables:
```
SELECT Employees.EmployeeID, Employees.FirstName, Employees.LastName,
Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;
```

Grouping and Aggregation:
```
SELECT Department, AVG(Age) AS AverageAge, COUNT(*) AS EmployeeCount
FROM Employees
GROUP BY Department;
```

Subqueries in WHERE Clause:
```
SELECT FirstName, LastName
FROM Employees
WHERE DepartmentID IN (SELECT DepartmentID FROM Departments WHERE
DepartmentName = 'IT');
```

Using HAVING Clause:
```
SELECT Department, AVG(Age) AS AverageAge
FROM Employees
GROUP BY Department
HAVING AVG(Age) > 30;
```

Window Functions (ROW_NUMBER):

```
SELECT FirstName, LastName, Age, ROW_NUMBER() OVER (PARTITION BY
Department ORDER BY Age DESC) AS Rank
FROM Employees;
```

Common Table Expressions (CTE):

```
WITH HighSalaryEmployees AS (
    SELECT FirstName, LastName, Salary
    FROM Employees
    WHERE Salary > 80000
)
SELECT * FROM HighSalaryEmployees;
```

Dynamic SQL with Stored Procedures:

```
CREATE PROCEDURE GetEmployeesByDepartment(IN departmentName
VARCHAR(50))
BEGIN
    SET @sql = 'SELECT * FROM Employees WHERE Department = ?';
    PREPARE stmt FROM @sql;
    EXECUTE stmt USING departmentName;
    DEALLOCATE PREPARE stmt;
END;
```

These examples showcase the progression from basic SQL operations to more advanced features, including joins, subqueries, aggregations, window functions, and stored procedures. As the complexity increases, SQL becomes a powerful tool for managing and analyzing relational databases.

Resources:

1. https://www.talend.com/resources/what-is-data-processing/
2. https://www.simplilearn.com/what-is-data-processing-article
3. https://www.referenceforbusiness.com/management/Comp-De/Data-Processing-and-Data-Management.html
4. https://support.microsoft.com/en-gb/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5
5. https://brewminate.com/a-brief-history-of-database-processing-since-the-1960s/
6. https://www.geeksforgeeks.org/types-of-databases/
7. https://www.matillion.com/blog/the-types-of-databases-with-examples
8. https://www.javatpoint.com/dbms-language
9. https://www.integrate.io/blog/the-sql-vs-nosql-difference/
10. https://www.mongodb.com/nosql-explained/nosql-vs-sql
11. https://www.ibm.com/blog/sql-vs-nosql/
12. https://www.geeksforgeeks.org/difference-between-sql-and-nosql/