SQL Queries Assignment

SATYAJIT NAYAK

1. SQL Basics

- **Define SQL. Explain its importance in database management.**

Introduction:

SQL, or Structured Query Language, is a standard programming language designed for managing and manipulating data in relational database management systems (RDBMS). It serves as the primary means to interact with databases, allowing users to define, query, update, and administer data.

Importance of SQL in database Management-

SQL has several importances in database management system such as-

Data Definition:

SQL's Data Definition Language (DDL) allows for the creation, modification, and deletion of database objects like tables, indexes, and views, defining the structure of the database.

Data Manipulation:

SQL's Data Manipulation Language (DML) enables users to insert, update, delete, and retrieve data within the defined database structures. This is crucial for managing the actual information stored.

Data Control:

SQL's Data Control Language (DCL) provides mechanisms for managing database access permissions and security, ensuring data integrity and confidentiality.

Standardization:

As an ANSI standard, SQL provides a consistent and widely adopted language across various RDBMS platforms, simplifying data migration and reducing vendor lock-in.

Efficiency and Scalability:

SQL queries are optimized for efficient data retrieval and manipulation, making it suitable for managing large datasets and supporting high-volume transactions.

- Describe basic SQL data types.

SQL data types define the kind of data that can be stored in a column, local variable, expression, or parameter. Common basic data types include:

1. Numeric Data Types:

a.INT (or INTEGER): Stores whole numbers (integers).

b.DECIMAL (or NUMERIC): Stores exact numeric values with a specified precision and scale, suitable for monetary values or precise calculations.

c. FLOAT (or REAL, DOUBLE PRECISION): Stores approximate floating-point numbers, suitable for scientific or less precise calculations.

2.String Data Types:

a. VARCHAR(n): Stores variable-length strings of characters, with n specifying the maximum length.

b. CHAR(n): Stores fixed-length strings of characters, padded with spaces if the string is shorter than n.

C. TEXT:Stores large blocks of text.

3.Date and Time Data Types:

a.DATE: Stores dates (year, month, day).

b.TIME: Stores times (hour, minute, second).

c.DATE & TIME (or TIME purpose stamp): Stores both date and time values.

4.Boolean Data Type:

a. BOOLEAN : Stores logical values, typically True or False.

2. SQL Queries

- Given a schema for a library database (books, authors, and borrowers), write SQL queries to:

A.**Create Table with constraints :-**

SQL query for create Table Authors:

```
CREATE TABLE Authors (
  author_id INT PRIMARY KEY,
  author_name VARCHAR(255) NOT NULL
);
```

SQL query for create table Books:

```
CREATE TABLE Books (
   book_id INT PRIMARY KEY,
   title VARCHAR(255) NOT NULL,
   author_id INT NOT NULL,
   publication_year INT,
   CONSTRAINT fk_author
      FOREIGN KEY (author_id)
      REFERENCES Authors(author_id)
      ON DELETE CASCADE
);
```

SQL query for create table Borrowers:

```
CREATE TABLE Borrowers (
   borrower_id INT PRIMARY KEY,
   borrower_name VARCHAR(255) NOT NULL,
   address VARCHAR(255),
   phone_number VARCHAR(20) UNIQUE
);
```

SQL query for create table Issued:

```
CREATE TABLE Issued (
   issue_id INT PRIMARY KEY,
   book_id INT,
   borrower_id INT,
   issue_date DATE NOT NULL,
   return_date DATE,
   FOREIGN KEY (book_id) REFERENCES Books(book_id),
 FOREIGN KEY (borrower_id) REFERENCES Borrowers(borrower_id)
);
```

B.Insert data Into Tables:

```
INSERT INTO Authors (author_id, first_name, last_name) VALUES
(1, 'Rohit', 'Nayak'),
(2, 'Sonam', 'Das'),
(3, 'Nihar', 'Sahoo');

INSERT INTO Books (book_id, title, author_id, publication_year) VALUES
(101, 'Human Science', 1, 1945),
(102, 'Village Story', 2, 1946),
(103, 'To whom I trust', 3, 2002),
(104, 'Animal Biology', 2, 2012);

INSERT INTO Borrowers (borrower_id, first_name, last_name, email) VALUES
(1, 'Pinkey', 'Nayak', 'pinky7@gmail.com'),
(2, 'Sahil', 'Ahmed', 'sahil6@gmail.com');

INSERT INTO Issued (issue_id, book_id, borrower_id, issue_date, return_date) VALUES
(1, 101, 1, '2025-07-01', '2025-07-15'),
(2, 102, 2, '2025-07-05', NULL),
(3, 103, 1, '2025-08-01', '2025-08-15');
```

C. Retrieve Data using SELECT with Conditions:

1. Where

```
SELECT title, publication_year FROM Books WHERE publication_year < 2000;
```

2. And

```
SELECT B.title, A.first_name, A.last_name
FROM Books B JOIN Authors A ON B.author_id = A.author_id
WHERE A.first_name = 'Sonam' AND A.last_name = 'Das' AND B.publication_year > 1945;
```

3.OR

```
SELECT B.title, A.first_name, A.last_name
FROM Books B JOIN Authors A ON B.author_id = A.author_id
WHERE (A.first_name = 'Rohit' AND A.last_name = 'Nayak') OR B.publication_year
= 2002;
```

D. Use Aggregate Functions:

1.Count

```
SELECT COUNT(*) AS TotalBooks FROM Books;
```

2.SUM

```
SELECT A.first_name, A.last_name, COUNT(B.book_id) AS NumberOfBooks
FROM Authors A LEFT JOIN Books B ON A.author_id = B.author_id
GROUP BY A.author_id, A.first_name, A.last_name;
```

3. AVERAGE

```
SELECT AVG(publication_year) AS AveragePublicationYear FROM Books;
```

3. JOINS AND SUBQUERIES

1. INNER JOIN to fetch related data:

A.This query retrieves the title of books and the names of their authors.

Query-

```
SELECT  b.title,  a.author_name FROM books AS b INNER JOIN  authors AS a ON b.author_id
= a.author_id;
```

B.This query retrieves the names of borrowers and the titles of books they have borrowed.

Query-

```
SELECT br.borrower_name, b.title FROM borrowers AS br INNER JOIN borrowed_books AS
bb ON br.borrower_id = bb.borrower_id INNER JOIN  books AS b ON bb.book_id =
b.book_id;
```

2. Subqueries to fetch data based on conditions:

A.This query finds the titles of books written by a specific author (e.g., 'Rohit Nayak').

Query-

```
SELECT title FROM  books WHERE author_id IN (SELECT author_id FROM authors WHERE author_name = 'Rohit Nayak');
```

B.This query finds the names of borrowers who have borrowed a specific book (e.g., 'Human Science').

Query-

```
SELECT borrower_name FROM  borrowers WHERE  borrower_id IN (
    SELECT borrower_id  FROM  borrowed_books WHERE book_id = (SELECT book_id FROM books WHERE title = 'Human Science')
);
```

4. SQL Constraints and Indexes

A.Explain the use of constraints (PRIMARY KEY, FOREIGN KEY).

SQL constraints are rules applied to columns or tables in a database to enforce data integrity and maintain data consistency.

A.1.PRIMARY KEY Constraint:

A Primary Key uniquely identifies each row in a table. Its characteristics include:

- **Uniqueness:** No two rows can have the same primary key value.
- **Non-Nullability :** A primary key column cannot contain NULL values.
- **Single per Table:** A table can have only one primary key.
- **Purpose:** It ensures entity integrity, meaning each entity (row) in the table is uniquely identifiable. It is also commonly used to establish relationships with other tables through foreign keys.

A.2.FOREIGN KEY Constraint:

A Foreign Key establishes a link between two tables, enforcing referential integrity. Its characteristics include:

- **Referential Integrity:**

  The value in the foreign key column of the referencing table must exist as a primary key (or unique key) in the referenced table.

- **Relationship Enforcement:**

  It ensures that relationships between tables are maintained, preventing orphaned records.

- **Purpose:**
  It maintains consistency across related tables, ensuring that data referenced in one table actually exists in the other.

B. Describe the purpose of indexes in SQL.

Indexes are database objects that improve the speed of data retrieval operations on a table.

- **Faster Data Retrieval:**

  Indexes create a sorted structure on one or more columns, allowing the database system to quickly locate specific rows without scanning the entire table (full table scan). This is particularly beneficial for SELECT statements with WHERE clauses, JOIN operations, and ORDER BY clauses.

- **Performance Optimization:**

  By reducing the amount of data the database needs to examine, indexes significantly enhance query performance, especially in large tables.

- **Trade-offs:**
  While indexes speed up read operations, they can slightly slow down write operations (inserts, updates, and deletes) because the index itself needs to be updated whenever the underlying data changes. Therefore, indexes should be created strategically on columns that are frequently used in search conditions or for sorting.

Thank You