Chapter 3

1. Assume the Relations given below.
Student(Enrno, name, courseId, emailId, cellno)
Course(courseId, course_nm, duration)

a. Find out the list of students who have enrolled in the "computer" course.

SQL Statement:

```
SELECT s.name
FROM Student s
INNER JOIN Course c ON s.courseId = c.courseId
WHERE c.course_nm = 'computer';
```

b. List the names of all courses with their duration.

SQL Statement:

```
SELECT course_nm, duration
FROM Course;
```

c. List the names of all students starting with "a".

SQL Statement:

```
SELECT name
FROM Student
WHERE name LIKE 'a%';
```

d. List the email IDs and cell numbers of all mechanical engineering students.

SQL Statement:

```
SELECT s.emailId, s.cellno
FROM Student s
INNER JOIN Course c ON s.courseId = c.courseId
WHERE c.course_nm = 'mechanical engineering';
```

2. Write the basic structure of SQL Queries. Explain working of each keyword in the structure.

1. SELECT: This keyword is used to specify the columns that you want to retrieve data from. You can either specify specific column names or use a wildcard (*) to select all columns.

2. FROM: This keyword is used to specify the table or tables from which you want to retrieve data. It indicates the source of the data you are querying.

3. WHERE: This keyword is used to filter the rows that are retrieved from the tables specified in the FROM clause. It allows you to specify conditions that must be met for a row to be included in the result set.

4. GROUP BY: This keyword is used to group the rows that have the same values in specified columns into summary rows. It's often used in conjunction with aggregate functions like SUM, COUNT, AVG, etc.

5. HAVING: This keyword is used to filter the groups that are generated by the GROUP BY clause. It's similar to the WHERE clause but operates on grouped rows rather than individual rows.

6. ORDER BY: This keyword is used to sort the rows retrieved from the tables specified in the FROM clause. It allows you to specify one or more columns and the sort order (ASC for ascending, DESC for descending).

7. LIMIT: This keyword is used to limit the number of rows returned by a query. It's typically used in conjunction with ORDER BY to retrieve the top N rows or to implement pagination.

Here's an example of a basic SQL query structure:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column1
HAVING condition
ORDER BY column1 ASC
LIMIT n;
```

3. How are the following integrity constraints implemented in SQL:

- **a. Domain constraint**
- **b. Referential integrity.**

Explain the above with appropriate syntax and example

a. Domain Constraint:

A domain constraint ensures that values stored in a column or attribute are of a specific data type and within a certain range or set of values.

In SQL, you can implement domain constraints using data types and check constraints.

Example:
Let's say you have a table called Employee with a column age, and you want to ensure that the age of an employee is between 18 and 65:

```sql
CREATE TABLE Employee (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT CHECK (age >= 18 AND age <= 65)
);
```

In the above example, the CHECK constraint ensures that the age column's value must be greater than or equal to 18 and less than or equal to 65, thereby enforcing the domain constraint on the age attribute.

b. Referential Integrity:

Referential integrity ensures that relationships between tables remain consistent. It is typically implemented using foreign key constraints, which enforce referential integrity by ensuring that values in one table's referencing column exist in another table's referenced column.

Example:
Suppose you have two tables, Orders and Customers, where each order is associated with a customer. You want to ensure that every order in the Orders table corresponds to a valid customer in the Customers table.

```sql
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(50)
);

CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

- In the Orders table, the customer_id column is a foreign key that references the customer_id column in the Customers table.
- This ensures that every customer_id value in the Orders table must exist in the Customers table, thereby maintaining referential integrity.
- If an attempt is made to insert a row into the Orders table with a customer_id that does not exist in the Customers table, it will result in a foreign key constraint violation error.

4. List and explain aggregate functions of SQL with appropriate examples

Aggregate functions in SQL are used to perform calculations on sets of values and return a single value as a result. Here are some common aggregate functions along with explanations and examples:

1. **COUNT()**:
   - Counts the number of rows in a result set.
   - Can be used with the asterisk (*) to count all rows or with a specific column to count non-null values in that column.

Example:

```
SELECT COUNT(*) AS total_students FROM Students;
```

2. **SUM()**:
   - Calculates the sum of values in a numeric column.
   - Ignores NULL values.

Example:

```
SELECT SUM(sales_amount) AS total_sales FROM Sales;
```

3. **AVG()**:
   - Calculates the average of values in a numeric column.
   - Ignores NULL values.

Example:

```sql
SELECT AVG(salary) AS average_salary FROM Employees;
```

4. **MIN():**
   - Returns the minimum value in a column.
   - Ignores NULL values.

   Example:

```sql
SELECT MIN(age) AS youngest_age FROM Employees;
```

5. **MAX():**
   - Returns the maximum value in a column.
   - Ignores NULL values.

   Example:

```sql
SELECT MAX(age) AS oldest_age FROM Employees;
```

5. List and explain the different DML statements in SQL

Data Manipulation Language (DML) statements in SQL are used to manipulate data stored in database tables. Here are the main DML statements along with explanations:

1. SELECT:
   - Retrieves data from one or more tables.
   - Used to query and retrieve rows and columns from tables based on specified criteria.

   Example:

```sql
SELECT column1, column2 FROM table_name WHERE condition;
```

2. INSERT:
   - Adds new rows of data into a table.

- Can insert data into specific columns or provide values for all columns.

Example:

```
INSERT INTO table_name (column1, column2) VALUES (value1, value2);
```

3. UPDATE:
- Modifies existing data in a table.
- Allows updating one or more columns in one or more rows based on specified conditions.

Example:

```
UPDATE table_name SET column1 = new_value1, column2 = new_value2 WHERE condition;
```

4. DELETE:
- Removes one or more rows from a table.
- Deletes rows based on specified conditions, or all rows if no condition is provided.

Example:

```
DELETE FROM table_name WHERE condition;
```

6. Explain the following SQL constructs with examples:
(1) order by,
(2) group by,
(3) having,
(4) as,
(5) in

1. ORDER BY:

- The ORDER BY clause is used to sort the result set based on one or more columns.
- It sorts the rows returned by a SELECT statement in ascending or descending order.
- By default, ORDER BY sorts in ascending order. To sort in descending order, use the DESC keyword.

Example:

```
SELECT column1, column2
FROM table_name
ORDER BY column1 DESC;
```

2. GROUP BY:
- The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows.
- It's often used in conjunction with aggregate functions like COUNT, SUM, AVG, etc.

Example:

```
SELECT department, COUNT(*) AS employee_count
FROM employees
GROUP BY department;
```

3. HAVING:
- The HAVING clause is used to filter the groups generated by the GROUP BY clause.
- It's similar to the WHERE clause but operates on grouped rows rather than individual rows.

Example:

```
SELECT department, COUNT(*) AS employee_count
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;
```

4. AS:

- The AS keyword is used to rename columns or tables in the result set.
- It's often used to make the output of a query more readable or to provide meaningful aliases for columns or tables.

Example:

```
SELECT first_name AS "First Name", last_name AS "Last Name"
FROM employees;
```

5. IN:

- The IN operator is used to specify multiple values in a WHERE clause.
- It allows you to specify a list of values against which to compare a column's value.

Example:

```
SELECT * FROM products
WHERE category_id IN (1, 2, 3);
```

7. Consider the following Database design

Customer (cid, custname, coststreet, custcity)
Account (accno, branchname, balance)
Loan (loanno, branchname, amount)
Borrower (cid, loanno)
Branch (branchname, branchcity, asset)
Depositor (cid, accno)

Solve the following queries in SQL

a. Display the names of customers who have both an account and a loan at the bank:

```
SELECT c.custname
```

```
FROM Customer c
INNER JOIN Depositor d ON c.cid = d.cid
INNER JOIN Account a ON d.accno = a.accno
INNER JOIN Borrower b ON c.cid = b.cid
INNER JOIN Loan l ON b.loanno = l.loanno;
```

b. Update the amount of the loan to 10000 where the loan number is "L-101":

```
UPDATE Loan
SET amount = 10000
WHERE loanno = 'L-101';
```

c. Change the column name custcity to ccity:

```
ALTER TABLE Customer
RENAME COLUMN custcity TO ccity;
```

d. Find all customers who have an account but no loan at the bank:

```
SELECT c.custname
FROM Customer c
LEFT JOIN Depositor d ON c.cid = d.cid
LEFT JOIN Borrower b ON c.cid = b.cid
WHERE b.cid IS NULL;
```

8. The following relations keep track of the Library Management system.

Book_info(bookid, bname, bauthor, price, edition, publication, pur_date,)
Student(lib_car_num, stud_name, class, branch, roll_no)
Issue_table(issue_date, sub_date, bookid, lib_car_num, due)

Write the following SQL queries:

a. Find the details of the books issued to library card number 1:

```
SELECT Book_info.*
FROM Book_info
JOIN Issue_table ON Book_info.bookid = Issue_table.bookid
WHERE Issue_table.lib_car_num = 1;
```

b. Give all the information about the student and the book issued with ascending order of library card number:

```
SELECT Student.*, Book_info.*
FROM Student
JOIN Issue_table ON Student.lib_car_num = Issue_table.lib_car_num
JOIN Book_info ON Issue_table.bookid = Book_info.bookid
ORDER BY Student.lib_car_num ASC;
```

c. Find the author, edition, and price of the book:

```
SELECT bauthor AS Author, edition AS Edition, price AS Price
FROM Book_info;
```

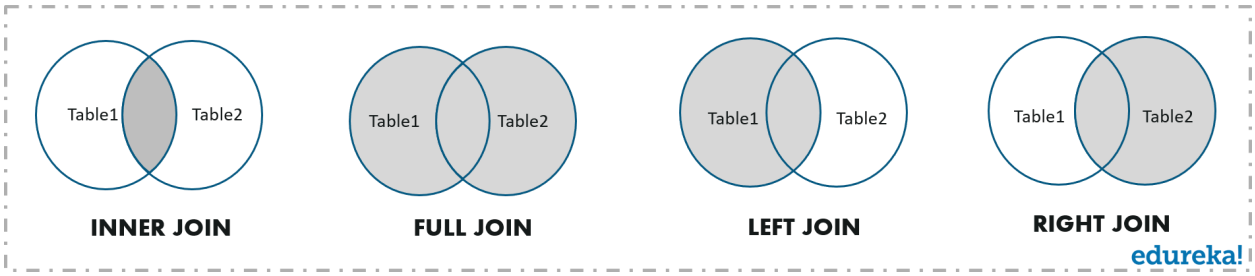
d. Find the names of the students with dues on the book issue:

```
SELECT Student.stud_name
FROM Student
JOIN Issue_table ON Student.lib_car_num = Issue_table.lib_car_num
WHERE Issue_table.due > 0;
```

9. List and explain the types of Join in SQL

There are mainly four types of joins that you need to understand. They are:

- INNER JOIN
- FULL JOIN
- LEFT JOIN
- RIGHT JOIN



considering the following three tables to show you how to perform the Join operations on such tables.

Employee Table:

EmpID	EmpFname	EmpLname	Age	EmailID	PhoneNo	Address
1	Vardhan	Kumar	22	vardy@abc.com	9876543210	Delhi
2	Himani	Sharma	32	himani@abc.com	9977554422	Mumbai
3	Aayushi	Shreshth	24	aayushi@abc.com	9977555121	Kolkata
4	Hemanth	Sharma	25	hemanth@abc.com	9876545666	Bengaluru
5	Swatee	Kapoor	26	swatee@abc.com	9544567777	Hyderabad

Project Table:

ProjectID	EmpID	ClientID	ProjectName	ProjectStartDate
111	1	3	Project1	2019-04-21
222	2	1	Project2	2019-02-12
333	3	5	Project3	2019-01-10
444	3	2	Project4	2019-04-16
555	5	4	Project5	2019-05-23
666	9	1	Project6	2019-01-12
777	7	2	Project7	2019-07-25
888	8	3	Project8	2019-08-20

Client Table:

Client ID	ClientFname	ClientLname	Age	ClientEmailID	PhoneNo	Address	EmpID
1	Susan	Smith	30	susan@adn.com	9765411231	Kolkata	3
2	Mois	Ali	27	mois@jsq.com	9876543561	Kolkata	3
3	Soma	Paul	22	soma@wja.com	9966332211	Delhi	1
4	Zainab	Daginawala	40	zainab@qkq.com	9955884422	Hyderabad	5
5	Bhaskar	Reddy	32	bhaskar@xyz.com	9636963269	Mumbai	2

INNER JOIN

This type of join returns those records which have matching values in both tables. So, if you perform an INNER join operation between the Employee table and the Projects table, all the tuples which have matching values in both the tables will be given as output.

Syntax:

```
SELECT Table1.Column1,Table1.Column2,Table2.Column1,....
FROM Table1
INNER JOIN Table2
ON Table1.MatchingColumnName = Table2.MatchingColumnName;
```

NOTE: You can either use the keyword INNER JOIN or JOIN to perform this operation.

Example:

```
SELECT Employee.EmpID, Employee.EmpFname, Employee.EmpLname,
Projects.ProjectID, Projects.ProjectName

FROM Employee

INNER JOIN Projects ON Employee.EmpID=Projects.EmpID;
```

Output:

EmpID	EmpFname	EmpLname	ProjectID	ProjectName
1	Vardhan	Kumar	111	Project1
2	Himani	Sharma	222	Project2
3	Aayushi	Shreshth	333	Project3

3	Aayushi	Shreshth	444	Project4
5	Swatee	Kapoor	555	Project5

FULL JOIN

Full Join or the Full Outer Join returns all those records which either have a match in the left(Table1) or the right(Table2) table.

Syntax:

```
SELECT Table1.Column1,Table1.Column2,Table2.Column1,....
FROM Table1
FULL JOIN Table2
ON Table1.MatchingColumnName = Table2.MatchingColumnName;
```

Example:

```
1SELECT Employee.EmpFname, Employee.EmpLname, Projects.ProjectID
2FROM Employee
3FULL JOIN Projects
4ON Employee.EmpID = Projects.EmpID;
```

Output:

EmpFname	EmpLname	ProjectID
Vardhan	Kumar	111
Himani	Sharma	222

Aayushi	Shreshth	333
Aayushi	Shreshth	444
Hemanth	Sharma	NULL
Swatee	Kapoor	555
NULL	NULL	666
NULL	NULL	777
NULL	NULL	888

LEFT JOIN

The LEFT JOIN or the LEFT OUTER JOIN returns all the records from the left table and also those records which satisfy a condition from the right table. Also, for the records having no matching values in the right table, the output or the result-set will contain the NULL values.

Syntax:

```
SELECT Table1.Column1,Table1.Column2,Table2.Column1,....
FROM Table1
LEFT JOIN Table2
ON Table1.MatchingColumnName = Table2.MatchingColumnName;
```

Example:

```
1 SELECT Employee.EmpFname, Employee.EmpLname, Projects.ProjectID,
  Projects.ProjectName
2 FROM Employee
3 LEFT JOIN
4 ON Employee.EmpID = Projects.EmpID ;
```

Output:

EmpFname	EmpLname	ProjectID	ProjectName
Vardhan	Kumar	111	Project1
Himani	Sharma	222	Project2
Aayushi	Shreshth	333	Project3
Aayushi	Shreshth	444	Project4
Swatee	Kapoor	555	Project5
Hemanth	Sharma	NULL	NULL

RIGHT JOIN

The RIGHT JOIN or the RIGHT OUTER JOIN returns all the records from the right table and also those records which satisfy a condition from the left table. Also, for the records having no matching values in the left table, the output or the result-set will contain the NULL values.

Syntax:

```
SELECT Table1.Column1,Table1.Column2,Table2.Column1,....
FROM Table1
RIGHT JOIN Table2
ON Table1.MatchingColumnName = Table2.MatchingColumnName;
```

Example:

```
SELECT Employee.EmpFname, Employee.EmpLname, Projects.ProjectID,
Projects.ProjectName

FROM Employee
```

4RIGHT JOIN

ON Employee.EmpID = Projects.EmpID;

Output:

EmpFname	EmpLname	ProjectID	ProjectName
Vardhan	Kumar	111	Project1
Himani	Sharma	222	Project2
Aayushi	Shreshth	333	Project3
Aayushi	Shreshth	444	Project4
Swatee	Kapoor	555	Project5
NULL	NULL	666	Project6
NULL	NULL	777	Project7
NULL	NULL	888	Project8

10. Why is the Domain Constraint called an elementary Database constraint?

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain
- The term "Domain Constraint" refers to a rule that restricts the values that can be taken by an attribute or column in a database table.
- It specifies the permissible data values for a given attribute. It is often considered an elementary or fundamental constraint in a database because it defines the basic data types and ranges allowed for each attribute.
- The term "elementary" in this context implies that domain constraints are foundational or essential building blocks of a database schema.
- They establish the basic rules for what kind of data can be stored in each column, ensuring data integrity and consistency.

- Without domain constraints, databases would be prone to accepting incorrect or inconsistent data, leading to errors and issues in data management and analysis.
- In essence, domain constraints serve as the bedrock of database design, defining the characteristics and limitations of data elements within the system.
- They are fundamental to maintaining the accuracy, reliability, and usability of the database.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute