Experiment No.6

Implement various join operations

Aim :- Write simple query to implement join operations(equi join, natural join, inner join, outer joins).

Objective :- To apply different types of join to retrieve queries from the database management system.

Theory:

SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

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

A. INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table. table2: Second table

matching_column: Column common to both the tables.

B. LEFT JOIN

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column; table1:

First table.

table2: Second table matching_column: Column

common to both the tables.

C. RIGHT JOIN

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table. table2: Second table

matching_column: Column common to both the tables.

D. FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table. table2: Second table

matching_column: Column common to both the tables.

Implementation:

MY SQL CODE:

```
CREATE TABLE documents (
   doc_id INT PRIMARY KEY,
   doc_name VARCHAR(255),
   category_id INT,
   user_id INT
);

CREATE TABLE categories (
   category_id INT PRIMARY KEY,
   category_name VARCHAR(50)
);

CREATE TABLE users (
   user_id INT PRIMARY KEY,
   username VARCHAR(50),
   email VARCHAR(100)
);
INSERT INTO categories (category_id, category_name)
VALUES
   (1, 'Contracts'),
   (2, 'Reports'),
   (3, 'Invoices');

INSERT INTO users (user_id, username, email)
VALUES
   (1, 'john_doe', 'john@example.com'),
```

```
   (2, 'jane_smith', 'jane@example.com');


INSERT INTO documents (doc_id, doc_name, category_id, user_id)
VALUES
   (101, 'Contract_A', 1, 1),
   (102, 'Report_X', 2, 2),
   (103, 'Invoice_B', 3, 1);
SELECT d.doc_name, c.category_name
FROM documents d
INNER JOIN categories c ON d.category_id = c.category_id;
SELECT d.doc_name, u.username
FROM documents d
LEFT JOIN users u ON d.user_id = u.user_id;
SELECT u.username, d.doc_name
FROM users u
RIGHT JOIN documents d ON u.user_id = d.user_id;
SELECT IFNULL(u.username, 'N/A') AS username, d.doc_name
FROM users u
JOIN documents d ON u.user_id = d.user_id;
```

Output:

doc_name	category_name
Contract_A	Contracts
Report_X	Reports
Invoice_B	Invoices

doc_name	username
Contract_A	john_doe
Report_X	jane_smith
Invoice_B	john_doe

username	doc_name
john_doe	Contract_A
jane_smith	Report_X
john_doe	Invoice_B

username	doc_name
john_doe	Contract_A
jane_smith	Report_X
john_doe	Invoice_B

Conclusion:

1. Illustrate how to perform natural join for the joining attributes with different names with a suitable example.

   ➔ Natural Join with Different Attribute Names

- A **natural join** automatically joins tables based on columns with the same name.
- If you have different attribute names but want to perform a natural join, you can use aliases to rename the columns temporarily.
- Let's consider two tables: `employees` and `departments`.

1. **Create Sample Tables**:

```
2. CREATE TABLE employees (
3.     emp_id INT PRIMARY KEY,
4.     emp_name VARCHAR(50),
5.     dept_id INT
6. );
7.
```

```
CREATE TABLE departments (
    dept_id INT PRIMARY KEY,
     dept_name VARCHAR(50)
);
```

Insert Sample Data:

```
INSERT INTO employees (emp_id, emp_name, dept_id)
VALUES
    (1, 'John', 101),
    (2, 'Jane', 102);

INSERT INTO departments (dept_id, dept_name)
VALUES
    (101, 'HR'),
    (102, 'IT');
```

Perform Natural Join with Aliases:

```
SELECT e.emp_id, e.emp_name, d.dept_name
FROM employees AS e
NATURAL JOIN departments AS d;
```

In this example, we use aliases (`e` for `employees` and `d` for `departments`) to rename the columns temporarily. The natural join will match the `dept_id` columns even though they have different names.

2. Illustrate significant differences between natural join equi join and inner join.

➔ Differences between Natural Join, Equi Join, and Inner Join

- **Natural Join**:
  - Automatically joins tables based on columns with the same name.
  - Removes duplicate columns involved in the equality comparison.
  - Works well when the attribute names are consistent across tables.
  - May lead to unexpected results if column names change or if there are additional columns with the same name.
- **Equi Join**:
  - A general form of join using the equality operator (=) in the `ON` clause.
  - Explicitly specifies the joining condition.
  - Returns rows where the specified columns have equal values.
  - More flexible than natural join.
- **Inner Join**:
  - Joins two or more tables based on a specified condition.
  - Returns only the rows that satisfy the join condition.
  - Does not remove duplicate columns; all columns from both tables are included in the result.