**LAB 5: Advanced DQL with Joins and Set Operations**

**OBJECTIVE:**

To delve into advanced Data Query Language (DQL) concepts, concentrating on joins, relationships, and set operations, to efficiently retrieve and analyze data from relational databases.

**THEORY:**

Data Query Language (DQL) is a part of SQL specifically designed for data retrieval and analysis. This lab builds on complex querying techniques by incorporating joins, relationships, and set operations to effectively query relational databases. The main concepts include:

* **Joins:**  
  Joins combine rows from two or more tables based on related columns, allowing for complex data retrieval. Types of joins include:
  1. **Inner Join:** Retrieves records that match in both tables.
  2. **Left/Right Join:** Includes records from one table even if they don’t have a match in the other table.
  3. **Full Outer Join:** Combines all matching and non-matching records from both tables.
  4. **Self-Join:** Joins a table with itself to manage hierarchical data.
* **Set Operations:**  
  These operations are used to combine, intersect, or subtract data from different result sets:
  1. **UNION:** Merges the results of multiple queries while removing duplicates.
  2. **INTERSECT:** Retrieves the common data between two result sets.
  3. **EXCEPT:** Displays the differences between two result sets.
* **Relationships:**  
  Establishes associations between tables using foreign keys to ensure data integrity. Examples include one-to-many and many-to-many relationships. Cascading actions (e.g., ON DELETE CASCADE) help maintain data consistency.

**Key Features in Advanced Queries:**

* **Joins for Data Combination:**  
  Combines data from related tables (e.g., linking students with issued books) to retrieve meaningful results.
* **Set Operations for Data Comparison:**  
  Compares datasets to find common, unique, or differing data between queries.
* **Hierarchical Query Logic:**  
  Utilizes self-joins and subqueries to analyze hierarchical data, such as teacher-supervisor relationships.
* **Referential Integrity with Relationships:**  
  Uses foreign keys and cascading actions to maintain consistency and integrity in the database.

These advanced DQL techniques are critical for querying relational databases, enabling efficient data retrieval and analysis for real-world applications, such as library management systems.

**SQL QUERY:**

--CREATE DATABASE LAB5;

CREATE DATABASE LAB5;

USE LAB5;

DROP TABLE issuebt;

DROP TABLE issuebs;

DROP TABLE book;

DROP TABLE student;

DROP TABLE teacher;

CREATE TABLE book(

bid INT PRIMARY KEY,

bname VARCHAR(50),

publication VARCHAR(50),

author VARCHAR(50),

price DECIMAL(8,2)

);

CREATE TABLE student(

sid INT PRIMARY KEY,

sfname VARCHAR(50),

slname VARCHAR(50),

sbranch VARCHAR(50),

address VARCHAR(50)

);

CREATE TABLE teacher(

tid INT PRIMARY KEY,

tfname VARCHAR(50),

tlname VARCHAR(50),

tbranch VARCHAR(50),

tsalary BIGINT,

hid INT FOREIGN KEY REFERENCES teacher(tid),

specialization VARCHAR(50)

);

CREATE TABLE issuebs(

bid INT FOREIGN KEY REFERENCES book(bid),

sid INT FOREIGN KEY REFERENCES student(sid),

dateofIssue DATE,

PRIMARY KEY(bid, sid)

);

CREATE TABLE issuebt(

bid INT FOREIGN KEY REFERENCES book(bid),

tid INT FOREIGN KEY REFERENCES teacher(tid),

dateofIssue DATE,

PRIMARY KEY(bid, tid)

);

-- Modified book data

INSERT INTO book VALUES

(101, 'Advanced Databases', 'TechPub', 'John Smith', 550.00),

(102, 'Cloud Computing Basics', 'Innovative Press', 'Alice Brown', 650.00),

(103, 'Data Science for Beginners', 'DataTech', 'Bruce Lee', 470.00),

(104, 'Network Security', 'CyberTech', 'Emma Watson', 750.00),

(105, 'Discrete Mathematics', 'EduPub', 'Alan Turing', 820.00),

(106, 'Calculus for Engineers', 'MathWorld', 'Isaac Newton', 950.00),

(107, 'Biology for Students', 'BioBooks', 'Charles Darwin', 210.00),

(108, 'Social Studies', 'HistoryPress', 'Leo Tolstoy', 320.00),

(109, 'General Knowledge', 'MindBooks', 'Albert Einstein', 430.00),

(110, 'Nepali Literature', 'Gpub', 'Rabindranath Tagore', 520.00),

(111, 'New Horizons', 'FutureBooks', 'J.K. Rowling', 540.00),

(112, 'Lost Secrets', 'Mystic Pub', 'George Orwell', 540.00),

(113, 'Ancient History', 'OldPress', 'Homer', 550.00),

(114, 'Artificial Intelligence', 'SmartPub', 'Sujan Sharma', 500.00);

-- Modified student data

INSERT INTO student VALUES

(1, 'Mohan', 'Sharma', 'BCT', 'Kathmandu'),

(2, 'Sita', 'Khadka', 'BCE', 'Pokhara'),

(3, 'Ravi', 'Thapa', 'BCT', 'Chitwan'),

(4, 'Priya', 'Subedi', 'BEX', 'Bhairahawa'),

(5, 'Sushant', 'Paudel', 'BEI', 'Lalitpur'),

(6, 'Gita', 'Bista', 'BCT', 'Bhaktapur'),

(7, 'Pradeep', 'Rai', 'BCT', 'Pokhara'),

(8, 'Shree', 'Singh', 'BEX', 'Kathmandu'),

(9, 'Arjun', 'Gurung', 'BCT', 'Chitwan'),

(10, 'Nina', 'Joshi', 'BAG', 'Bhaktapur');

-- Modified teacher data

INSERT INTO teacher VALUES

(1, 'Amit', 'Kumar', 'BCT', 110000, 1, 'Data Science'),

(2, 'Vijay', 'Shrestha', 'BCE', 210000, 1, 'Networking'),

(3, 'Priya', 'Poudel', 'BCT', 310000, 2, 'Cloud Computing'),

(4, 'Ayesha', 'Ghimire', 'BEX', 420000, 3, 'Artificial Intelligence'),

(5, 'Nikita', 'Nepal', 'BEI', 510000, 4, 'Software Engineering'),

(6, 'Suman', 'Thapa', 'BCT', 600000, 5, 'Machine Learning'),

(7, 'Niraj', 'Rai', 'BEI', 700000, 6, 'Big Data'),

(8, 'Kiran', 'Gurung', 'BEX', 800000, 7, 'Embedded Systems'),

(9, 'Prakash', 'Adhikari', 'BCT', 900000, 8, 'Data Structures'),

(10, 'Rina', 'Maharjan', 'BAG', 1000000, 9, 'Database Management');

-- Modified issuebs data

INSERT INTO issuebs VALUES

(102, 1, '2024-07-01'),

(101, 2, '2024-07-05'),

(103, 3, '2024-07-07'),

(104, 4, '2024-07-10'),

(105, 5, '2024-07-15'),

(106, 6, '2024-07-20'),

(107, 7, '2024-07-25'),

(108, 8, '2024-07-30'),

(109, 9, '2024-08-02'),

(110, 10, '2024-08-05'),

(111, 2, '2024-08-10'),

(112, 3, '2024-08-12'),

(113, 4, '2024-08-15'),

(114, 5, '2024-08-20');

-- Modified issuebt data

INSERT INTO issuebt VALUES

(102, 1, '2024-06-01'),

(101, 2, '2024-06-03'),

(103, 3, '2024-06-05'),

(104, 4, '2024-06-08'),

(105, 5, '2024-06-12'),

(106, 6, '2024-06-14'),

(107, 7, '2024-06-18'),

(108, 8, '2024-06-22'),

(109, 9, '2024-06-25'),

(110, 10, '2024-06-28'),

(111, 1, '2024-07-01'),

(112, 2, '2024-07-03'),

(113, 3, '2024-07-06'),

(114, 4, '2024-07-09');

SELECT \*FROM book;

SELECT \*FROM student

JOIN issuebs ON student.sid=issuebs.sid;

SELECT \*FROM student, issuebs

WHERE student.sid=issuebs.sid;

SELECT \*FROM student s

JOIN issuebs i ON s.sid=i.sid;

SELECT \*FROM student s

JOIN issuebs i ON s.sid=i.sid

WHERE sfname='Mohan';

SELECT \*FROM student s

LEFT OUTER JOIN issuebs i ON s.sid=i.sid;

SELECT \*FROM student s

LEFT OUTER JOIN issuebs i ON s.sid=i.sid

WHERE i.sid IS NULL;

--Right Outer JOIN

SELECT \*FROM student s

RIGHT OUTER JOIN issuebs i ON s.sid=i.sid;

SELECT \*FROM student s

FULL OUTER JOIN issuebs i ON s.sid=i.sid;

SELECT \*FROM teacher t

JOIN teacher h ON t.hid=h.tid;

SELECT t.tfname AS teachername,h.tfname AS headName FROM teacher t JOIN teacher h ON t.hid=h.tid;

SELECT \* FROM teacher t

JOIN book b on t.tfname=b.author;

SELECT bname,sfname,slname,dateofissue FROM issuebs ibs

JOIN book b ON b.bid=ibs.bid

JOIN student s ON ibs.sid=s.sid;

SELECT bname,sfname,slname,dateofissue FROM issuebs ibs

JOIN book b ON b.bid=ibs.bid

JOIN student s ON ibs.sid=s.sid

WHERE dateofIssue='2024-08-20';

SELECT sfname FROM student

UNION

SELECT tfname FROM teacher;

INSERT INTO teacher VALUES (15, 'Dipson','Adhikari', 'BCT', 100000,1,'AI');

SELECT sfname FROM student

INTERSECT

SELECT tfname FROM teacher;

SELECT tfname FROM teacher

EXCEPT

SELECT sfname FROM student;

SELECT bname AS bookName,sfname AS personName FROM issuebs ibs

JOIN book b ON b.bid=ibs.bid

JOIN student s ON ibs.sid=s.sid

WHERE dateofIssue='2024-06-01'

UNION

SELECT bname,tfname FROM issuebt ibt

JOIN book b ON b.bid=ibt.bid

JOIN teacher t ON ibt.tid=t.tid

WHERE dateofIssue='2024-06-01';

**Questions:**

1. Do the cross-product between book and issuebs table.

Ans: SELECT \*FROM book, issuebs;

* The above query does the cross product of the book table and issuebs table and since there are 15 rows(tuples) in book table and 14 tuples in issuebs table hence the cross product of these two tables results in a table with 15\*14=210

tuples with all the columns(fields/attributes) from both book and issuebs table.

Output:



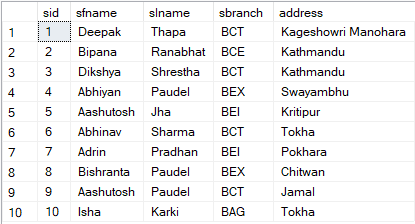
* Since there were 210 rows/tuples it wasn’t possible to show all of them. So,

only few are shown above.

2. Display the detail of all the student who has issued book.

SELECT \*FROM student

JOIN issuebs ON student.sid=issuebs.sid;



* This is the proper method of showing the detail of all students who has issued book. This method uses join: if student’s sid matches with issuebs’s sid then that student must have taken the book.

We could also do it like this using Cross-product:

SELECT \*FROM student, issuebs

WHERE student.sid=issuebs.sid;

This query is not good because: The JOIN syntax is better than the WHERE clause for joining tables because it improves readability, makes the join intent clearer, and allows the database to optimize the query more efficiently.

1. Use of Aliases:

**Aliases**: Aliases in DBMS are temporary names given to tables or columns to simplify queries and improve readability.

Example of it:

SELECT \*FROM student s

JOIN issuebs i ON s.sid=i.sid;

Does the same job as the above but we can now use s instead of student and i instead of issuebs.

4. Display the detail of student (Aashutosh only) who has taken book.

SELECT \*FROM student s

JOIN issuebs i ON s.sid=i.sid

WHERE sfname='Aashutosh';



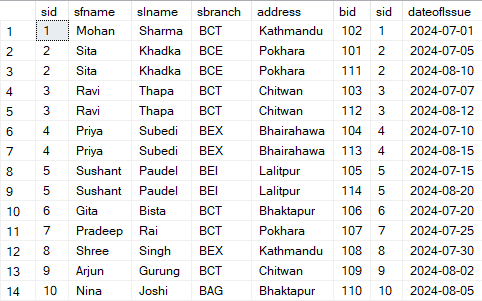
1. Display the detail of the student who haven’t taken the book.

→We do outer join for this because: it outputs unmatched data from student table who are the student who haven’t taken the book along with the students

who have taken the book.

SELECT \*FROM student s

LEFT OUTER JOIN issuebs i ON s.sid=i.sid;



To display only the detail of student who haven’t taken the book we can do this:

SELECT \*FROM student s

LEFT OUTER JOIN issuebs i ON s.sid=i.sid

WHERE i.sid IS NULL;

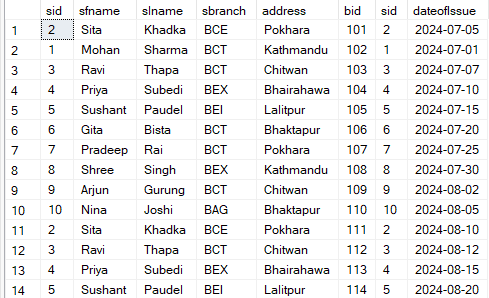


6. To do Right outer join and full outer join between student and issuebs table.

--Right Outer JOIN

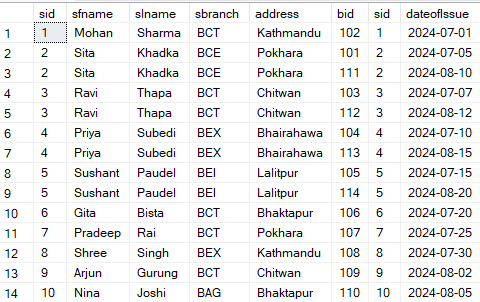
SELECT \*FROM student s

RIGHT OUTER JOIN issuebs i ON s.sid=i.sid;



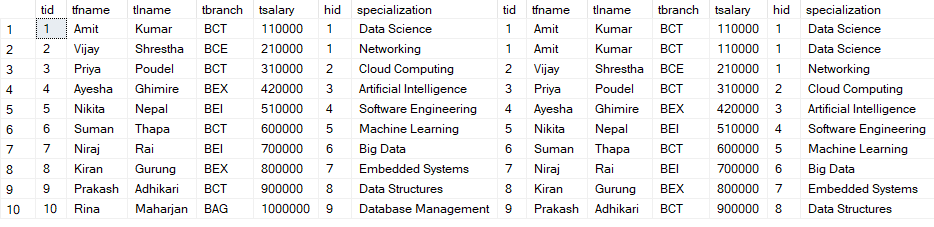
SELECT \*FROM student s

FULL OUTER JOIN issuebs i ON s.sid=i.sid;

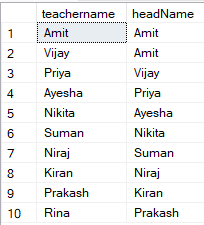


1. SELF JOIN to: DISPLAY TEACHER NAME ALONG WITH HEAD NAME SELECT \*FROM teacher t

JOIN teacher h ON t.hid=h.tid;



SELECT t.tfname AS teachername,h.tfname AS headName FROM teacher t JOIN teacher h ON t.hid=h.tid;



--display the name of teacher who has also issued a book SELECT \* FROM teacher t

JOIN book b on t.tfname=b.author;

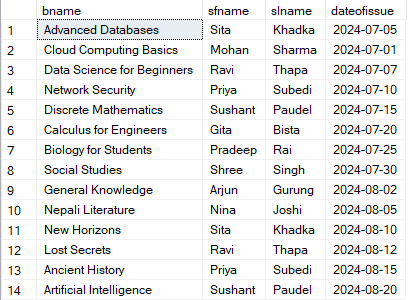


Display student name and book name who had issued book with date of issue.

SELECT bname,sfname,slname,dateofissue FROM issuebs ibs

JOIN book b ON b.bid=ibs.bid

JOIN student s ON ibs.sid=s.sid;



Display the name of the book and student name who has issued the book on ‘2024-08-20’.

SELECT bname,sfname,slname,dateofissue FROM issuebs ibs

JOIN book b ON b.bid=ibs.bid

JOIN student s ON ibs.sid=s.sid

WHERE dateofIssue='2024-08-20';



--display all teacher name and student name

SELECT sfname FROM student

UNION

SELECT tfname FROM teacher;



--student as well as teacher intersect

INSERT INTO teacher VALUES (15, 'Dipson','Adhikari', 'BCT', 100000,1,'AI'); SELECT sfname FROM student

INTERSECT

SELECT tfname FROM teacher;



--TEACHER BUT NOT STUDENT

SELECT tfname FROM teacher

EXCEPT

SELECT sfname FROM student;



--DISPLAY ALL THE TEACHER AND STUDENT NAME WHO HAS ISSUED BOOK ON THIS DATE SELECT bname AS bookName,sfname AS personName FROM issuebs ibs

JOIN book b ON b.bid=ibs.bid

JOIN student s ON ibs.sid=s.sid

WHERE dateofIssue='2024-06-01'

UNION

SELECT bname,tfname FROM issuebt ibt

JOIN book b ON b.bid=ibt.bid

JOIN teacher t ON ibt.tid=t.tid

WHERE dateofIssue='2024-06-01';



Assignment: Database of Realistic System

CREATE DATABASE ECommerce;

USE ECommerce;

* Drop tables if they already exist DROP TABLE OrderItems;

DROP TABLE Orders; DROP TABLE Customers; DROP TABLE Products;

* Create Products table

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10,2),

Stock INT

);

* Create Customers table CREATE TABLE Customers (

CustomerID INT PRIMARY KEY, FirstName VARCHAR(50), LastName VARCHAR(50), Email VARCHAR(100), Address VARCHAR(200)

);

* Create Orders table

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID) ON DELETE CASCADE ON UPDATE CASCADE,

OrderDate DATE,

TotalAmount DECIMAL(10,2)

);

* Create OrderItems table CREATE TABLE OrderItems (

OrderID INT FOREIGN KEY REFERENCES Orders(OrderID) ON DELETE CASCADE ON UPDATE CASCADE,

ProductID INT FOREIGN KEY REFERENCES Products(ProductID), Quantity INT,

SubTotal DECIMAL(10,2),

PRIMARY KEY (OrderID, ProductID)

);

* Insert sample data into Products

INSERT INTO Products VALUES

(1, 'Laptop', 'Electronics', 800.00, 50),

(2, 'Smartphone', 'Electronics', 500.00, 200), (3, 'Headphones', 'Accessories', 50.00, 150), (4, 'Desk Chair', 'Furniture', 120.00, 30), (5, 'Notebook', 'Stationery', 2.50, 500);

* Insert sample data into Customers INSERT INTO Customers VALUES

(1, 'John', 'Doe', 'john.doe@example.com', '123 Maple St, Springfield'), (2, 'Jane', 'Smith', 'jane.smith@example.com', '456 Oak St, Metropolis'), (3, 'Emily', 'Johnson', 'emily.j@example.com', '789 Pine St, Gotham');

* Insert sample data into Orders

INSERT INTO Orders VALUES

(101, 1, '2025-01-01', 850.00),

(102, 2, '2025-01-02', 550.00),

(103, 3, '2025-01-03', 170.00);

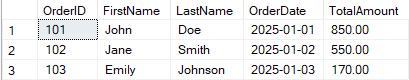
* Insert sample data into OrderItems INSERT INTO OrderItems VALUES

(101, 1, 1, 800.00), (101, 3, 1, 50.00), (102, 2, 1, 500.00), (102, 3, 1, 50.00), (103, 4, 1, 120.00), (103, 5, 20, 50.00);

* 1. Display all orders with customer details

SELECT o.OrderID, c.FirstName, c.LastName, o.OrderDate, o.TotalAmount FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID;



* 2. Display all products that are out of stock SELECT \* FROM Products WHERE Stock = 0;
  + Currently no product out of stock.
* 3. Show the total number of orders placed by each customer SELECT c.FirstName, c.LastName, COUNT(o.OrderID) AS TotalOrders FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID GROUP BY c.CustomerID , c.FirstName, c.LastName;

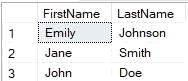


* 4. Show all customers who placed orders in January 2025 SELECT DISTINCT c.FirstName, c.LastName

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

WHERE MONTH(o.OrderDate) = 1 AND YEAR(o.OrderDate) = 2025;



* 5. Show details of products and their total sales quantity SELECT p.ProductName, p.Category, SUM(oi.Quantity) AS TotalSold FROM Products p

JOIN OrderItems oi ON p.ProductID = oi.ProductID GROUP BY p.ProductID, p.ProductName,p.Category;



* 6. Show products that have never been sold

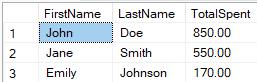
SELECT \*

FROM Products p

LEFT JOIN OrderItems oi ON p.ProductID = oi.ProductID WHERE oi.ProductID IS NULL;

* + All products have been sold.
* 7. Show customer names along with the total amount they spent SELECT c.FirstName, c.LastName, SUM(o.TotalAmount) AS TotalSpent FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID GROUP BY c.CustomerID,c.FirstName,c.LastName;



DISCUSSION:

This lab provided an in-depth exploration of advanced Data Query Language (DQL) techniques, focusing on the use of subqueries and joins. We practiced merging data from multiple tables using various types of joins, such as INNER JOIN, LEFT JOIN, and RIGHT JOIN, enabling us to extract valuable insights from relational databases. These joins allowed us to identify and analyze relationships between different data points, such as between authors and publishers or sales records.

Subqueries were also employed to add another layer of complexity and flexibility to our queries. By using subqueries, we were able to structure queries with conditions based on nested results, allowing for more precise and organized data extraction. This method proved especially useful when the necessary data was not directly available from the main query’s tables.

The exercises demonstrated the importance of joins for combining related data and how subqueries can enhance query results by introducing nested logic. Together, these techniques form the basis for building efficient and comprehensive database systems that are essential for handling complex data interactions.

CONCLUSION:

This lab enhanced our understanding of advanced DQL techniques, particularly the application of subqueries and joins. Through hands-on experience, we learned how to design queries that efficiently handle complex relationships and refine data retrieval. The ability to merge data from multiple tables using joins and filter results with subqueries adds depth and flexibility to our querying capabilities. Mastering these techniques is crucial for addressing real-world database problems, especially in professional environments that require managing large, interconnected datasets.