University Institute of Engineering

Department of Computer Science & Engineering

Experiment:1

Date of Experiment: 28-07-2025
EASY

1. Aim of the practical: Author-Book Relationship Using Joins and Basic SQL Operations

2. Problem Statement:

- Design two tables to store author and book details.
- Establish a foreign key relationship from books to authors.
- Insert sample data and perform an INNER JOIN.

3. Theory:

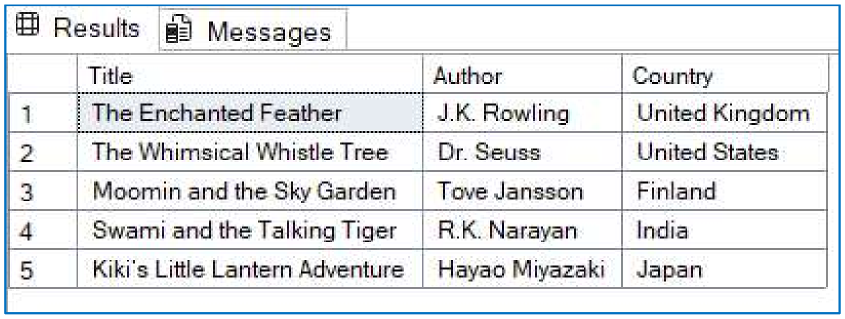
A foreign key is a field in one table that refers to the primary key in another table. It is used to establish a relationship between the two tables and ensures referential integrity, i.e., the foreign key value must exist in the referenced table. This allows us to logically connect data across different tables.
Joins in SQL are used to combine rows from two or more tables based on a related column between them. The necessary condition for a join is that there must be a common column between the tables involved — the names of the columns do not need to be the same, but the data must be logically related (e.g., author ID in both author and book tables).
Although joins are mostly used between two tables, joins can also be performed on a single table, such as in a self-join, where a table is joined with itself.
Among various types of joins, the most common is the INNER JOIN, which returns only the records that have matching values in both tables. In our case, the BOOKS table is joined with the AUTHORS table on the related author ID column to fetch combined details like book title, author name, and country.

4. Queries:

```
/* 1. Create the database */

CREATE DATABASE BookCafe;
USE BookCafe;

/*2. Create the Authors table */

CREATE TABLE Authors(
                author_id INT PRIMARY KEY,
                author_name VARCHAR(150),
                nationality VARCHAR(60)
 );
```

```
/* 3. Create the Books table with FK to Authors */

CREATE TABLE Books(
        book_code INT PRIMARY KEY,
        book_title VARCHAR(120),
        author_id INT, FOREIGN KEY (author_id) REFERENCES Authors(author_id) );

/* 4. Insert authors (from 5 different countries) */

INSERT INTO Authors (author_id, author_name, nationality) VALUES
        (101, 'J.K. Rowling', 'United Kingdom'), (102, 'Dr. Seuss', 'United States'),
        (103, 'Tove Jansson', 'Finland'), (104, 'R.K. Narayan', 'India'),
        (105, 'Hayao Miyazaki', 'Japan');

/* 5. Insert books */ INSERT INTO Books (book_code, book_title, author_id) VALUES

        (101,'The Enchanted Feather',101), (102, 'The Whimsical Whistle Tree', 102),
        (103,'Moomin and the Sky Garden',103),(104, 'Swamiandthe Talking Tiger',104),
        (105, 'Kiki’s Little Lantern Adventure', 105);

/* 6. Final SELECT query to display result */

        SELECT B.book_title AS Title, A.author_name AS Author,
        A.nationality AS Country
        FROM Books B JOIN Authors A ON B.author_id = A.author_id;
```

5. Output:



	Title	Author	Country
1	The Enchanted Feather	J.K. Rowling	United Kingdom
2	The Whimsical Whistle Tree	Dr. Seuss	United States
3	Moomin and the Sky Garden	Tove Jansson	Finland
4	Swami and the Talking Tiger	R.K. Narayan	India
5	Kiki’s Little Lantern Adventure	Hayao Miyazaki	Japan

Result:

The INNER JOIN query successfully retrieved each book along with its corresponding author's name and country. The output confirms the correct relationship between the BOOKS and AUTHORS tables using the AuthorID.

Learning outcomes (What I have learnt):

1. I have learnt how to design relational tables using primary and foreign keys.
2. I have learnt to insert multiple records efficiently into SQL tables.
3. I have learnt how to use INNER JOIN to combine related data from different tables.
4. I have learnt the importance of maintaining referential integrity in databases.
5. I have learnt to retrieve meaningful combined information using SQL queries.

Experiment:1

Date of Experiment: 28-07-2025
MEDIUM

1. Aim of the practical: Department-Course Subquery and Access Control

2. Problem Statement:

- Design normalized tables for departments and the courses they offer, maintaining a foreign key relationship.
- Insert five departments and at least ten courses across those departments.
- Use a subquery to count the number of courses under each department.
- Filter and retrieve only those departments that offer more than two courses.
- Grant SELECT-only access on the courses table to a specific user.

3. Theory:

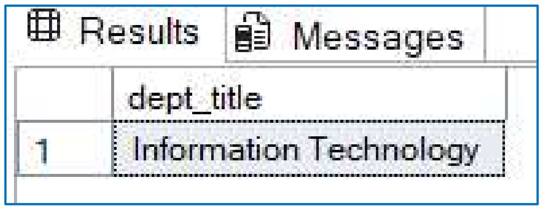
A subquery (or nested query) is a query embedded within another SQL query. Subqueries are used to filter, compute, or transform data dynamically based on the result of another query. They are especially useful when you need to apply conditions based on aggregated results, such as counts or averages.
In this experiment, a subquery is used to count how many courses belong to each department and filter only those departments that offer more than two courses using GROUP BY and HAVING.
Access control in SQL allows you to manage what actions a user can perform on database objects. The GRANT command is used to give privileges like SELECT, INSERT, UPDATE or DELETE to users. In this case, SELECT-only access ensures a user can read data from the table without modifying it.

4. Queries:

```
/* 1. Create the database */

        CREATE DATABASE AcademicDB;
        USE AcademicDB;

/* 2. Create the Departments table */

CREATE TABLE Departments (
        dept_id INT PRIMARY KEY,
        dept_title VARCHAR(80)
    );
```

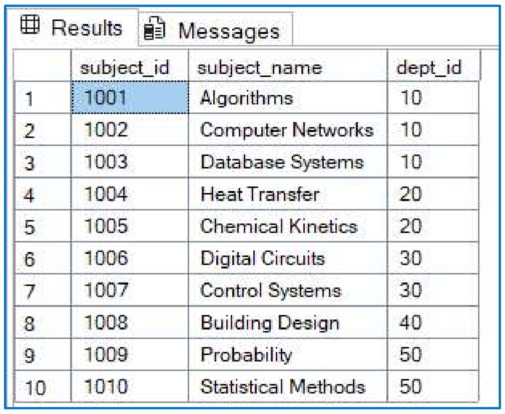
```
/* 3. Create the Subjects table with foreign key reference */

CREATE TABLE Subjects (
        subject_id INT PRIMARY KEY,
        subject_name VARCHAR(120),
        dept_id INT, FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
    );

/* 4. Insert department data */

INSERT INTO Departments (dept_id, dept_title) VALUES
      (10, 'Information Technology'), (20, 'Chemical Engineering'), (30, 'Electronics'),
      (40, 'Architecture'), (50, 'Statistics');

/* 5. Insert subject data */

INSERT INTO Subjects (subject_id, subject_name, dept_id) VALUES

      (1001, 'Algorithms', 10), (1002, 'Computer Networks', 10), (1003, 'Database Systems', 10),
      (1004, 'Heat Transfer', 20), (1005, 'Chemical Kinetics', 20), (1006, 'Digital Circuits', 30),
      (1007, 'Control Systems', 30), (1008, 'Building Design', 40), (1009, 'Probability', 50),
      (1010, 'Statistical Methods', 50);

/* 6. Final SELECT: Departments with at least 3 subjects */

      SELECT dept_title FROM Departments WHERE dept_id IN (
         SELECT dept_id FROM Subjects GROUP BY dept_id HAVING COUNT(subject_id) >= 3
      );

/* 7. Granting SELECT-Only Access to a User on the Subjects Table in SQL Server */

      CREATE LOGIN Adi WITH PASSWORD = 'Test@123';
      CREATE USER Miku FOR LOGIN Adi;
      GRANT SELECT ON Subjects TO Miku;
      EXECUTE AS USER = 'Miku';
      SELECT * FROM Subjects;
      REVERT;
```

6. Output:



Results | Messages

	dept_title
1	Information Technology



Results | Messages

	subject_id	subject_name	dept_id
1	1001	Algorithms	10
2	1002	Computer Networks	10
3	1003	Database Systems	10
4	1004	Heat Transfer	20
5	1005	Chemical Kinetics	20
6	1006	Digital Circuits	30
7	1007	Control Systems	30
8	1008	Building Design	40
9	1009	Probability	50
10	1010	Statistical Methods	50

Result:

The subquery successfully retrieved only those departments offering more than two courses, with Computer Science appearing as the department with the highest number. The user readonly_user was created and granted SELECT-only access to ensure controlled read-only visibility of the TBL_COURSE table.

Learning outcomes (What I have learnt):

1. I have learnt how to write subqueries using GROUP BY and HAVING clauses.
2. I have learnt to filter records based on aggregated conditions.
3. I have learnt to insert multiple records using single INSERT statements.
4. I have learnt to create and manage users and access permissions in SQL Server.
5. I have learnt to implement basic access control using the GRANT command.