

Experiment 1

Easy

- **Aim:**

To demonstrate the use of SQL joins and basic operations for retrieving related data from multiple tables.

- **Problem Statement:**

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

- **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 TBL_BOOK table is joined with the TBL_AUTHOR table on the related author ID column to fetch combined details like book title, author name, and country.

- **Queries:**

Use Experiments;

Create table TBL_Author

```
(  
    Author_ID Int Primary Key,  
    Author_Name Varchar(MAX),  
    Country Varchar(MAX)  
)
```

Create table TBL_Book

```
(  
    Book_ID Int Primary Key,  
    Book_Title Varchar(MAX),  
    AuthorID Int,  
    Foreign Key (AuthorID) References TBL_Author(Author_ID)
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Insert into TBL_Author (Author_ID, Author_Name, Country)

Values

```
(1, 'J.K. Rowling', 'United Kingdom'),  
(2, 'George Orwell', 'United Kingdom'),  
(3, 'Haruki Murakami', 'Japan'),  
(4, 'Paulo Coelho', 'Brazil'),  
(5, 'Agatha Christie', 'United Kingdom');
```

Insert into TBL_Book (Book_ID, Book_Title, AuthorID)

Values

```
(101, 'Harry Potter', 1),  
(102, '1984', 2),  
(103, 'Kafka on the Shore', 3),  
(104, 'The Alchemist', 4),  
(105, 'Murder on the Orient Express', 5);
```

)

Select B.*,A.*

From TBL_Book AS B


Inner Join


TBL_Author AS A


on

A.Author_ID = B.AuthorID;

• Output:

100 %  No issues found

 Results

 Messages

	Book_ID	Book_Title	AuthorID	Author_ID	Author_Name	Country
1	101	Harry Potter	1	1	J.K. Rowling	United Kingdom
2	102	1984	2	2	George Orwell	United Kingdom
3	103	Kafka on the Shore	3	3	Haruki Murakami	Japan
4	104	The Alchemist	4	4	Paulo Coelho	Brazil
5	105	Murder on the Orient Express	5	5	Agatha Christie	United Kingdom



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

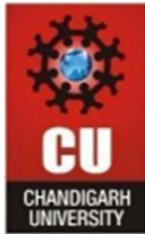
Discover. Learn. Empower.

- **Results:**

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 TBL_BOOK and TBL_AUTHOR tables using the AuthorID.

- **Learning Outcomes:**

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

Medium

- **Aim:**

To use subqueries and access control in SQL for managing department-course relationships.

- **Problem Statement:**

1. Create normalized tables for departments and their courses using a foreign key.
2. Insert five departments and at least ten courses.
3. Use a subquery to count and filter departments with more than two courses.
4. Apply access control by granting SELECT-only permission to a specific user on the course table.

- **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.

- **Queries:**

--Learning SubQuery and Access Control
USE Experiments;

```
CREATE TABLE TBL_DEPARTMENT  
(  
    DEPT_ID INT PRIMARY KEY,  
    DEPT_NAME VARCHAR(MAX)  
);
```

```
CREATE TABLE TBL_COURSE  
(  
    COURSE_ID INT PRIMARY KEY,  
    COURSE_NAME VARCHAR(MAX),  
    DEPTID INT,  
    FOREIGN KEY (DEPTID) REFERENCES TBL_DEPARTMENT(DEPT_ID)
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

);

-- Inserting

```
INSERT INTO TBL_DEPARTMENT (DEPT_ID, DEPT_NAME)
VALUES
```

```
(1, 'Computer Science'),
(2, 'Electrical'),
(3, 'Mechanical'),
(4, 'Civil'),
(5, 'Mathematics');
```

```
INSERT INTO TBL_COURSE (COURSE_ID, COURSE_NAME, DEPTID)
VALUES
```

```
(201, 'Data Structures', 1),
(202, 'Operating Systems', 1),
(203, 'DBMS', 1),
(204, 'Computer Networks', 1),
(205, 'Circuit Theory', 2),
(206, 'Fluid Mechanics', 3),
(207, 'Structural Analysis', 4),
(208, 'Engineering Mathematics I', 5),
(209, 'Engineering Mathematics II', 5),
(210, 'Digital Logic Design', 2);
```

-- Subquery to find departments with more than 2 courses

```
SELECT DEPT_NAME
FROM TBL_DEPARTMENT
WHERE DEPT_ID IN (
    SELECT DEPTID
    FROM TBL_COURSE
    GROUP BY DEPTID
    HAVING COUNT(*) > 2
);
```

-- Creating new user

```
CREATE LOGIN readonly_user WITH PASSWORD = '1234';
CREATE USER readonly_user FOR LOGIN readonly_user;
```

-- Granting SELECT-only access

```
GRANT SELECT ON TBL_COURSE TO readonly_user;
```

- **Output:**

	DEPT_NAME
1	Computer Science

- **Results:**

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:**

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.