

SQL JOINS, Triggers & Stored Procedures

*Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales*

Learning Objectives

- SQL JOINS – Combining data from multiple tables
- Triggers – Automatic actions on table events
- Stored Procedures – Reusable SQL blocks
- Environment: MySQL in Uniform Server

Setup – Create the Database

```
CREATE DATABASE SchoolDB;  
USE SchoolDB;
```

Setup – Create tables

```
CREATE TABLE Students (  
  student_id INT PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(100)  
);
```

Setup – Create tables

```
CREATE TABLE Courses (  
    course_id INT PRIMARY KEY AUTO_INCREMENT,  
    course_name VARCHAR(100)  
);
```

Setup – Create tables

```
CREATE TABLE Enrollments (  
  enrollment_id INT PRIMARY KEY AUTO_INCREMENT,  
  student_id INT,  
  course_id INT,  
  FOREIGN KEY (student_id) REFERENCES Students(student_id),  
  FOREIGN KEY (course_id) REFERENCES Courses(course_id)  
);
```

Insert Sample Data

```
INSERT INTO Students (name) VALUES  
( 'Alice'), ( 'Bob'), ( 'Charlie');
```

```
INSERT INTO Courses (course_name) VALUES  
( 'Math'), ( 'Science'), ( 'English');
```

```
INSERT INTO Enrollments (student_id, course_id) VALUES  
(1, 1), (2, 2), (3, 1);
```

INNER JOIN

An INNER JOIN returns records **only where there is a match** in both tables.

student_id	name
1	Alice
2	Bob
3	Charlie

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

```
SELECT Students.name, Courses.course_name  
FROM Enrollments  
JOIN Students ON Enrollments.student_id =  
Students.student_id  
JOIN Courses ON Enrollments.course_id =  
Courses.course_id;
```

INNER JOIN

An INNER JOIN returns records **only where there is a match** in both tables.

student_id	name
1	Alice
2	Bob
3	Charlie

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

```
SELECT Students.name, Courses.course_name
FROM Enrollments
JOIN Students ON Enrollments.student_id =
Students.student_id
JOIN Courses ON Enrollments.course_id =
Courses.course_id;
```

tells the database
which columns to
return in the query
result.

INNER JOIN

An INNER JOIN returns records **only where there is a match** in both tables.

student_id	name
1	Alice
2	Bob
3	Charlie

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

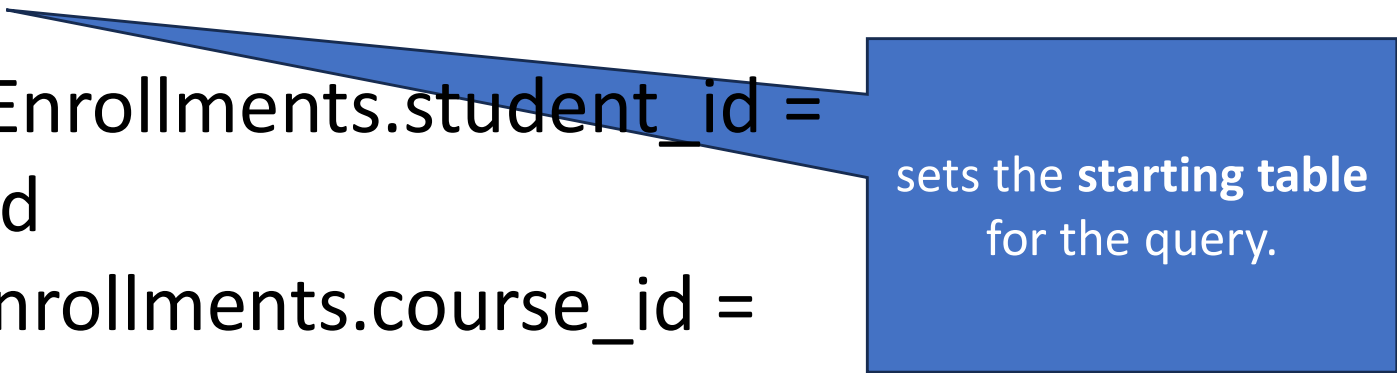
course_id	course_name
1	Math
2	Science
3	English

SELECT Students.name, Courses.course_name

FROM Enrollments

JOIN Students ON Enrollments.student_id =
Students.student_id

JOIN Courses ON Enrollments.course_id =
Courses.course_id;



sets the **starting table**
for the query.

INNER JOIN

An INNER JOIN returns records **only where there is a match** in both tables.

student_id	name
1	Alice
2	Bob
3	Charlie

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

```
SELECT Students.name, Courses.course_name
FROM Enrollments
JOIN Students ON Enrollments.student_id =
Students.student_id
JOIN Courses ON Enrollments.course_id =
Courses.course_id;
```

INNER JOIN
between
Enrollments and
Students

INNER JOIN

An INNER JOIN returns records **only where there is a match** in both tables.

student_id	name
1	Alice
2	Bob
3	Charlie

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

```
SELECT Students.name, Courses.course_name
FROM Enrollments
JOIN Students ON Enrollments.student_id =
Students.student_id
JOIN Courses ON Enrollments.course_id =
Courses.course_id;
```

INNER JOIN
between
Enrollments and
Courses

INNER JOIN

An INNER JOIN returns records **only where there is a match** in both tables.

student_id	name
1	Alice
2	Bob
3	Charlie

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

```
SELECT Students.name, Courses.course_name
FROM Enrollments
JOIN Students ON Enrollments.student_id = Students.student_id
JOIN Courses ON Enrollments.course_id = Courses.course_id;
```

Returns rows **only if** the student and course both have matching enrollment.

name	course_name
Alice	Math
Bob	Science
Charlie	Math

LEFT JOIN

A LEFT JOIN returns all the records from the left table and the matched records from the right table. If there is no match found in the right table, the result will still include the row from the left table but with NULL values for the columns of the right table.

INSERT INTO Students (name) VALUES ('David');

student_id	name
1	Alice
2	Bob
3	Charlie
4	David

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

LEFT JOIN

INSERT INTO Students (name) VALUES ('David');

student_id	name
1	Alice
2	Bob
3	Charlie
4	David

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

SELECT Students.name, Courses.course_name

FROM Students

LEFT JOIN Enrollments ON Students.student_id = Enrollments.student_id

LEFT JOIN Courses ON Enrollments.course_id = Courses.course_id;

name	course_name
Alice	Math
Bob	Science
Charlie	Math
David	NULL

LEFT JOIN

INSERT INTO Students (name) VALUES ('David');

student_id	name
1	Alice
2	Bob
3	Charlie
4	David

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

SELECT Students.name, Courses.course_name

FROM Students

LEFT JOIN Enrollments ON Students.student_id = Enrollments.student_id

LEFT JOIN Courses ON Enrollments.course_id = Courses.course_id;

name	course_name
Alice	Math
Bob	Science
Charlie	Math
David	NULL

STORED PROCEDURE

student_id	name
1	Alice
2	Bob
3	Charlie
4	David

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

DELIMITER //

```
CREATE PROCEDURE ViewAllStudents()  
BEGIN  
    SELECT * FROM Students;  
END;  
//
```

DELIMITER ;

CALL ViewAllStudents();

STORED PROCEDURE

student_id	name
1	Alice
2	Bob
3	Charlie
4	David

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

DELIMITER //

```
CREATE PROCEDURE ViewStudentNames()  
BEGIN  
    SELECT name FROM Students;  
END;  
//
```

DELIMITER ;

CALL ViewStudentNames();

STORED PROCEDURE

student_id	name
1	Alice
2	Bob
3	Charlie
4	David

enrollment_id	student_id	course_id
1	1	1
2	2	2
3	3	1

course_id	course_name
1	Math
2	Science
3	English

```
DELIMITER //  
CREATE PROCEDURE ViewStudentByID(IN stu_id INT)  
BEGIN  
    SELECT * FROM Students WHERE student_id = stu_id;  
END;  
//  
DELIMITER ;  
CALL ViewStudentByID(1);
```

```
SHOW PROCEDURE STATUS WHERE Db = 'YourDatabaseName';
```

```
SHOW CREATE PROCEDURE procedure_name;
```

STORED PROCEDURE – to insert records

```
CREATE DATABASE IF NOT EXISTS SimpleDB;  
USE SimpleDB;
```

```
CREATE TABLE Items (  
  item_id INT PRIMARY KEY AUTO_INCREMENT,  
  item_name VARCHAR(100)  
);
```

```
INSERT INTO Items (item_name) VALUES  
('Item A'), ('Item B'), ('Item C'), ('Item D'), ('Item E');
```

```
SELECT * FROM Items;
```

STORED PROCEDURE – to insert records

```
DELIMITER //  
  
CREATE PROCEDURE AddMoreItems()  
BEGIN  
    INSERT INTO Items (item_name) VALUES  
    ('Item F'), ('Item G'), ('Item H'), ('Item I'), ('Item J');  
END;  
//  
  
DELIMITER ;
```

```
CALL AddMoreItems();
```

```
SELECT * FROM Items;
```

STORED PROCEDURE – to view records

```
DELIMITER //
```

```
CREATE PROCEDURE ViewAllItems()
```

```
BEGIN
```

```
    SELECT * FROM Items;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

```
CALL ViewAllItems();
```

Transaction

A transaction groups multiple SQL statements so they either all succeed (commit) or all fail (rollback).

Useful to keep data consistent, especially when multiple inserts or updates happen together.

Transaction

```
DELIMITER //
CREATE PROCEDURE AddMoreItems()
BEGIN
    DECLARE exit handler FOR SQLEXCEPTION
    BEGIN
        -- If any error occurs, rollback transaction
        ROLLBACK;
    END;

    START TRANSACTION;
    INSERT INTO Items (item_name) VALUES
    ('Item F'), ('Item G'), ('Item H'), ('Item I'), ('Item J');
    COMMIT;
END;
//
DELIMITER ;
```

Transaction

```
DELIMITER //
CREATE PROCEDURE AddMoreItems()
BEGIN
    DECLARE exit handler FOR SQLEXCEPTION
    BEGIN
        -- If any error occurs, rollback transaction
        ROLLBACK;
    END;

    START TRANSACTION;
    INSERT INTO Items (item_name) VALUES
    ('Item F'), ('Item G'), ('Item H'), ('Item I'), ('Item J');
    COMMIT;
END;
//
DELIMITER ;
```

- **START TRANSACTION;** begins a new transaction.
- **INSERT** statements run inside that transaction.
- If all succeed, **COMMIT;** saves the changes permanently.
- If any error occurs, the exit handler triggers and runs **ROLLBACK;** undoing all changes in this transaction.

Transaction - Testing

CALL AddMoreItems();

The 5 items will be inserted and committed.

If you want to test rollback, you could insert a deliberate error (e.g., duplicate primary key or invalid data) inside the procedure and see that no changes are committed.

Transaction - Testing

```
DELIMITER //
CREATE PROCEDURE AddMoreItemsWithError()
BEGIN
    DECLARE exit handler FOR SQLEXCEPTION
    BEGIN
        -- Rollback transaction if any error occurs
        ROLLBACK;
        SELECT 'Error occurred, transaction rolled back.' AS Message;
    END;
    START TRANSACTION;
    -- Valid inserts
    INSERT INTO Items (item_name) VALUES
    ('Item F'), ('Item G'), ('Item H'), ('Item I'), ('Item J');
    -- Deliberate error: inserting NULL into NOT NULL column (assuming item_name NOT
    NULL)
    INSERT INTO Items (item_name) VALUES (NULL);
    COMMIT;
END;
//
DELIMITER ;
```

Cartesian Product

- result of combining all rows from one table with all rows from another table
- cross join (X)

Customers Table:

CustomerID	CustomerName
1	Alice
2	Bob
3	Charlie

Orders Table:

OrderID	OrderDate
101	2023-01-10
102	2023-01-15
103	2023-01-20

Cartesian Product

- result of combining all rows from one table with all rows from another table
- cross join (X)

Customers Table:

CustomerID	CustomerName	OrderID	OrderDate
1	Alice	101	2023-01-10
		102	2023-01-15
		103	2023-01-20
2	Bob	101	2023-01-10
		102	2023-01-15
		103	2023-01-20
3	Charlie		

Cartesian Product

- result of combining all rows from one table with all rows from another table
- cross join (X)

CustomerID	CustomerName	OrderID	OrderDate
1	Alice	101	2023-01-10
1	Alice	102	2023-01-15
1	Alice	103	2023-01-20
2	Bob	101	2023-01-10
2	Bob	102	2023-01-15
2	Bob	103	2023-01-20
3	Charlie	101	2023-01-10
3	Charlie	102	2023-01-15
3	Charlie	103	2023-01-20

Cartesian Product

Venues Table:

VenueID	VenueName	Capacity
1	Grand Ballroom	200
2	Garden Pavilion	100
3	Lakeside Lodge	150

Catering Options Table:

CateringID	Caterer	Cuisine
101	Gourmet Delight	Italian
102	Taste of India	Indian
103	Savory Bites	Mediterranean

Cartesian Product

Result (Cartesian Product):

VenueID	VenueName	Capacity	CateringID	Caterer	Cuisine
1	Grand Ballroom	200	101	Gourmet Delight	Italian
1	Grand Ballroom	200	102	Taste of India	Indian
1	Grand Ballroom	200	103	Savory Bites	Mediterranean
2	Garden Pavilion	100	101	Gourmet Delight	Italian
2	Garden Pavilion	100	102	Taste of India	Indian
2	Garden Pavilion	100	103	Savory Bites	Mediterranean
3	Lakeside Lodge	150	101	Gourmet Delight	Italian
3	Lakeside Lodge	150	102	Taste of India	Indian
3	Lakeside Lodge	150	103	Savory Bites	Mediterranean

Cartesian Product - MySQL

Employees Table:

EmployeeID	EmployeeName
1	Alice
2	Bob
3	Charlie

Projects Table:

ProjectID	ProjectName
101	Sales Forecast
102	Marketing Analysis

What is the resultant table after cartesian product?

Cartesian Product - MySQL

Result (Cartesian Product):

EmployeeID	EmployeeName	ProjectID	ProjectName
1	Alice	101	Sales Forecast
1	Alice	102	Marketing Analysis
2	Bob	101	Sales Forecast
2	Bob	102	Marketing Analysis
3	Charlie	101	Sales Forecast
3	Charlie	102	Marketing Analysis

What is the statement to get cartesian product?

Cartesian Product - MySQL

Result (Cartesian Product):

EmployeeID	EmployeeName	ProjectID	ProjectName
1	Alice	101	Sales Forecast
1	Alice	102	Marketing Analysis
2	Bob	101	Sales Forecast
2	Bob	102	Marketing Analysis
3	Charlie	101	Sales Forecast
3	Charlie	102	Marketing Analysis

```
SELECT * FROM Employees, Projects;
```

Joins

- Cartesian product creates mess results.
- Need to ensure that they are done using specific conditions.
- **Join** operation does it for us.
- Join is a **derivative of Cartesian product**, equivalent to performing a **Selection** operation, using the join predicate as the selection formula, **over the Cartesian product** of the two operand relations.
- Most difficult operations to implement efficiently in an RDBMS.
- - one of the reasons why relational systems have intrinsic performance problems.

Types of Joins

- There are various forms of Join operation, each with subtle differences, some more useful than others:
- Theta join
- Equi-join (a particular type of Theta join)
- Natural join
- Outer join
- Semi-join

- Our Area of Focus

Natural Join(inner join)

- The Natural join combines the two relations by automatically joining them on all common attributes.

Members Table:

MemberID	FirstName	LastName
1	John	Smith
2	Jane	Doe
3	Mark	Brown
4	Lisa	Green

Loans Table:

LoanID	MemberID	BookTitle	LoanDate
1	1	"Book A"	2023-10-01
2	2	"Book B"	2023-09-20
3	1	"Book C"	2023-09-28
4	3	"Book D"	2023-08-25
5	4	"Book E"	2023-10-10

Natural Join

Members Table:

MemberID	FirstName	LastName
1	John	Smith
2	Jane	Doe
3	Mark	Brown
4	Lisa	Green

Loans Table:

LoanID	MemberID	BookTitle	LoanDate
1	1	"Book A"	2023-10-01
2	2	"Book B"	2023-09-20
3	1	"Book C"	2023-09-28
4	3	"Book D"	2023-08-25
5	4	"Book E"	2023-10-10

```
SELECT * FROM members
JOIN loans ON members.MemberID = loans.MemberID;
```

Natural Join

```
SELECT * FROM members
```

```
JOIN loans ON members.MemberID = loans.MemberID;
```

MemberID	First Name	Last Name	Loan ID	Book Title	Loan Date
1	John	Smith	1	"Book A"	2023-10-01
1	John	Smith	3	"Book C"	2023-09-28
2	Jane	Doe	2	"Book B"	2023-09-20
3	Mark	Brown	4	"Book D"	2023-08-25
4	Lisa	Green	5	"Book E"	2023-10-10

Members Table:

MemberID	FirstName	LastName
1	John	Smith
2	Jane	Doe
3	Mark	Brown
4	Lisa	Green

Loans Table:

LoanID	MemberID	BookTitle	LoanDate
1	1	"Book A"	2023-10-01
2	2	"Book B"	2023-09-20
3	1	"Book C"	2023-09-28
4	3	"Book D"	2023-08-25
5	4	"Book E"	2023-10-10

studentid	firstname	lastname	department
-----------	-----------	----------	------------

studentid	firstname	lastname	department
-----------	-----------	----------	------------

1	John	Doe	Computer Science
---	------	-----	------------------

2	Alice	Smith	Mathematics
---	-------	-------	-------------

3	Bob	Johnson	Physics
---	-----	---------	---------

courseid	studentid	coursename
----------	-----------	------------

courseid	studentid	coursename
----------	-----------	------------

1	1	Database Management
---	---	---------------------

2	2	Calculus
---	---	----------

3	1	Data Structures
---	---	-----------------

4	3	Quantum Mechanics
---	---	-------------------

-- Create the Students table

```
CREATE TABLE Students (  
    StudentID INT AUTO_INCREMENT PRIMARY KEY,  
    FirstName VARCHAR(255),  
    LastName VARCHAR(255),  
    Department VARCHAR(255)  
);
```

-- Insert sample data into the Students table

```
INSERT INTO Students (FirstName, LastName,  
Department) VALUES  
    ('John', 'Doe', 'Computer Science'),  
    ('Alice', 'Smith', 'Mathematics'),  
    ('Bob', 'Johnson', 'Physics');
```

-- Create the Courses table

```
CREATE TABLE Courses (  
    CourseID INT AUTO_INCREMENT PRIMARY KEY,  
    StudentID INT,  
    CourseName VARCHAR(255),  
    FOREIGN KEY (StudentID) REFERENCES  
Students(StudentID)  
);
```

-- Insert sample data into the Courses table

```
INSERT INTO Courses (StudentID, CourseName) VALUES  
    (1, 'Database Management'),  
    (2, 'Calculus'),  
    (1, 'Data Structures'),  
    (3, 'Quantum Mechanics');
```

Try to perform join with SQL Query.

Natural join with SQL Query.

```
SELECT *  
FROM Students  
JOIN Courses ON Students.StudentID =  
Courses.StudentID;
```

or

```
SELECT Students.StudentID, FirstName, LastName,  
Department, Courseid, CourseName  
FROM Students  
JOIN Courses ON Students.StudentID =  
Courses.StudentID;
```

Natural join with SQL Query.

studentid	firstname	lastname	department	courseid	coursename
1	John	Doe	Computer Science	1	Database Management
2	Alice	Smith	Mathematics	2	Calculus
1	John	Doe	Computer Science	3	Data Structures
3	Bob	Johnson	Physics	4	Quantum Mechanics

Left Outer Join

A LEFT OUTER JOIN returns all records from the left table (in this case, Students) and the matching records from the right table (Courses). If there is no match, the result will contain NULL values for columns from the right table (Courses).

Students Table:

StudentID	FirstName	LastName
1	John	Doe
2	Alice	Smith
3	Bob	Johnson
4	David	White

Courses Table:

CourseID	StudentID	CourseName
1	1	Database Management
2	2	Calculus
3	1	Data Structures
4	3	Quantum Mechanics

Left Outer Join

A LEFT OUTER JOIN returns all records from the left table (in this case, Students) and the matching records from the right table (Courses). If there is no match, the result will contain NULL values for columns from the right table (Courses).

Dataset 1: Students Table

StudentID	FirstName	LastName
1	John	Smith
2	Jane	Doe
3	Alice	Brown
4	Bob	Green

Dataset 2: Courses Table

CourseID	StudentID	CourseName
101	1	Math
102	1	Physics
103	2	Chemistry

Left Outer Join

Left Join Output:

StudentID	FirstName	LastName	CourseName
1	John	Doe	Database Management
1	John	Doe	Data Structures
2	Alice	Smith	Calculus
3	Bob	Johnson	Quantum Mechanics
4	David	White	NULL

```
SELECT Students.StudentID, Students.FirstName,  
Students.LastName, Courses.CourseName  
FROM Students  
LEFT JOIN Courses ON Students.StudentID = Courses.StudentID;
```

Right Outer Join

A Right Join returns all records from the Courses table (right table) and the matched records from the Students table (left table).

If there's no match, the result from the left table will be NULL.

Students Table:

StudentID	FirstName	LastName
1	John	Doe
2	Alice	Smith
3	Bob	Johnson
4	David	White

Right Outer Join

A Right Join returns all records from the Courses table (right table) and the matched records from the Students table (left table).

If there's no match, the result from the left table will be NULL.

Courses Table:

CourseID	StudentID	CourseName
1	1	Database Management
2	2	Calculus
3	1	Data Structures
4	3	Quantum Mechanics
5	NULL	Artificial Intelligence
6	NULL	Machine Learning

Right Outer Join

A Right Join returns all records from the Courses table (right table) and the matched records from the Students table (left table).

If there's no match, the result from the left table will be NULL.

Dataset 1: Students Table

StudentID	FirstName	LastName
1	John	Smith
2	Jane	Doe
3	Alice	Brown
4	Bob	Green

Dataset 2: Courses Table

CourseID	StudentID	CourseName
101	1	Math
102	1	Physics
103	2	Chemistry
104	NULL	Biology

Right Outer Join

Right Join Output:

StudentID	FirstName	LastName	CourseName
1	John	Doe	Database Management
1	John	Doe	Data Structures
2	Alice	Smith	Calculus
3	Bob	Johnson	Quantum Mechanics
NULL	NULL	NULL	Artificial Intelligence
NULL	NULL	NULL	Machine Learning

```
SELECT Students.StudentID, Students.FirstName,  
Students.LastName, Courses.CourseName  
FROM Students  
RIGHT JOIN Courses ON Students.StudentID =  
Courses.StudentID;
```

Full Outer Join

A Full Outer Join returns all records when there is a match in either the left or right table. If there's no match, NULL values are returned from the table without a match.

Dataset 1: Students Table

StudentID	FirstName	LastName
1	John	Smith
2	Jane	Doe
3	Alice	Brown
4	Bob	Green

Dataset 2: Courses Table

CourseID	StudentID	CourseName
101	1	Math
102	1	Physics
103	2	Chemistry
104	NULL	Biology

Full Outer Join

A Full Outer Join returns all records when there is a match in either the left or right table. If there's no match, NULL values are returned from the table without a match.

```
SELECT Students.StudentID, Students.FirstName,  
Students.LastName, Courses.CourseNameFROM  
StudentsLEFT JOIN Courses ON Students.StudentID =  
Courses.StudentID
```

UNION

```
SELECT Students.StudentID, Students.FirstName,  
Students.LastName, Courses.CourseNameFROM  
StudentsRIGHT JOIN Courses ON Students.StudentID =  
Courses.StudentID;
```

Full Outer Join

Full Join Output:

StudentID	FirstName	LastName	CourseName
1	John	Doe	Database Management
1	John	Doe	Data Structures
2	Alice	Smith	Calculus
3	Bob	Johnson	Quantum Mechanics
4	David	White	NULL
5	Eve	Davis	NULL
NULL	NULL	NULL	Artificial Intelligence
NULL	NULL	NULL	Machine Learning

Outer Join – Types - Example

R		S	
Emp	Dept	Dept	Head
Smith	Sales	Production	Mori
Black	Production	Purchasing	Brown
White	Production		

R		S	
Emp	Dept	Dept	Head
Smith	Sales	Production	Mori
Black	Production	Purchasing	Brown
White	Production		

R		S	
Emp	Dept	Dept	Head
Smith	Sales	Production	Mori
Black	Production	Purchasing	Brown
White	Production		

R ⋈ S (left outer join)		
Emp	Dept	Head
Smith	Sales	Null
Black	Production	Mori
White	Production	Mori

R ⋈ S (right outer join)		
Emp	Dept	Head
Black	Production	Mori
White	Production	Mori
Null	Purchasing	Brown

(full outer join)		
Emp	Dept	Head
Smith	Sales	Null
Black	Production	Mori
White	Production	Mori
Null	Purchasing	Brown

Left Outer Join – Types - MySQL

Enrollments Table:

EnrollmentID	StudentID	CourseName
1	1	Mathematics
2	2	Physics
3	1	Chemistry
4	3	Biology

Students Table:

StudentID	FirstName	LastName
1	John	Doe
2	Alice	Smith
3	Bob	Johnson

Left Outer Join – Types - MySQL

Enrollments Table:

EnrollmentID	StudentID	CourseName
1	1	Mathematics
2	2	Physics
3	1	Chemistry
4	3	Biology

Students Table:

StudentID	FirstName	LastName
1	John	Doe
2	Alice	Smith
3	Bob	Johnson

```
SELECT Students.FirstName, Students.LastName, Enrollments.CourseName
FROM
Students LEFT JOIN Enrollments ON Students.StudentID = Enrollments.StudentID;
```

Right Outer Join – Types - MySQL

Enrollments Table:

EnrollmentID	StudentID	CourseName
1	1	Mathematics
2	2	Physics
3	1	Chemistry
4	3	Biology
5	4	History

Students Table:

StudentID	FirstName	LastName
1	John	Doe
2	Alice	Smith
3	Bob	Johnson

```
SELECT Students.FirstName, Students.LastName, Enrollments.CourseName
FROM
Students RIGHT JOIN Enrollments ON Students.StudentID = Enrollments.StudentID;
```

Full Outer Join – Types - MySQL

Enrollments Table:

EnrollmentID	StudentID	CourseName
1	1	Mathematics
2	2	Physics
3	1	Chemistry
4	3	Biology
5	4	History

Students Table:

StudentID	FirstName	LastName
1	John	Doe
2	Alice	Smith
3	Bob	Johnson

Full Outer Join – Types - MySQL

Enrollments Table:

EnrollmentID	StudentID	CourseName
1	1	Mathematics
2	2	Physics
3	1	Chemistry
4	3	Biology
5	4	History

Students Table:

StudentID	FirstName	LastName
1	John	Doe
2	Alice	Smith
3	Bob	Johnson

```
SELECT Students.FirstName, Students.LastName,  
Enrollments.CourseName  
FROM Students  
LEFT JOIN Enrollments ON Students.StudentID =  
Enrollments.StudentID
```

UNION

```
SELECT Students.FirstName, Students.LastName,  
Enrollments.CourseName  
FROM Students  
RIGHT JOIN Enrollments ON Students.StudentID =  
Enrollments.StudentID;
```

Stored Procedure

- a set of SQL statements stored in the database that can be executed as a single unit.
- can perform operations like inserting, updating, and deleting data.

Stored Procedure

```
CREATE TABLE Students (  
    StudentID INT AUTO_INCREMENT  
PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    Email VARCHAR(100)  
);
```

Stored Procedure

```
DELIMITER //
```

```
CREATE PROCEDURE AddStudent(  
    IN p_FirstName VARCHAR(50),  
    IN p_LastName VARCHAR(50),  
    IN p_Email VARCHAR(100)  
)  
BEGIN  
    INSERT INTO Students (FirstName, LastName,  
Email)  
    VALUES (p_FirstName, p_LastName, p_Email);  
END //
```

```
DELIMITER ;
```

Stored Procedure

```
CALL AddStudent ( 'John', 'Doe',  
'john.doe@example.com' );
```

```
SELECT * FROM Students;
```

Trigger

A **trigger** is a function that is automatically executed in response to certain events on a particular table, like insert, update, or delete.

Trigger

```
CREATE TABLE Students (  
    StudentID INT  
    AUTO_INCREMENT PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    Email VARCHAR(100),  
    LastUpdated TIMESTAMP  
    DEFAULT CURRENT_TIMESTAMP ON  
    UPDATE CURRENT_TIMESTAMP  
);
```

Trigger

```
DELIMITER //
```

```
CREATE TRIGGER UpdateLastUpdated  
BEFORE UPDATE ON Students  
FOR EACH ROW  
BEGIN  
    SET NEW.LastUpdated =  
CURRENT_TIMESTAMP;  
END //
```

```
DELIMITER ;
```

Trigger

Insert Sample Data

```
INSERT INTO Students (FirstName,  
LastName, Email) VALUES  
( 'John', 'Doe', 'john.doe@example.com' ),  
( 'Jane', 'Smith',  
'jane.smith@example.com' );
```

Update a Student Record

```
UPDATE Students  
SET Email =  
'john.doe@newdomain.com'  
WHERE StudentID = 1;
```

Trigger

Verify the Update

```
SELECT * FROM Students;
```

Transaction

A **transaction** is a group of operations that are executed as a single unit.

You can either **commit** these operations to the database or **rollback** if something goes wrong.

Transaction – Example: Bank Transfer

```
CREATE TABLE Accounts (  
    AccountID INT  
    AUTO_INCREMENT PRIMARY KEY,  
    AccountHolder  
    VARCHAR(100),  
    Balance DECIMAL(10, 2)  
);
```

Transaction – Example: Bank Transfer

```
INSERT INTO Accounts  
  (AccountHolder, Balance)  
VALUES  
  ('Alice', 500.00),  
  ('Bob', 300.00);
```

Transaction – Example: Bank Transfer

Transaction Example: Money Transfer

- Deducting £100 from Alice's balance.
- Adding £100 to Bob's balance.

Transaction - Example

Start a Transaction

```
START TRANSACTION;
```

Transaction - Example

Deduct from Alice's Account

```
UPDATE Accounts  
SET Balance = Balance - 100  
WHERE AccountHolder = 'Alice';
```

Transaction - Example

Add to Bob's Account

```
UPDATE Accounts  
SET Balance = Balance + 100  
WHERE AccountHolder = 'Bob';
```

Transaction - Example

Commit the Transaction

COMMIT;

Rollback

ROLLBACK;

Transaction - Example

```
START TRANSACTION;
```

```
UPDATE Accounts
```

```
SET Balance = Balance - 100
```

```
WHERE AccountHolder = 'Alice';
```

```
UPDATE Accounts
```

```
SET Balance = Balance + 100
```

```
WHERE AccountHolder = 'Bob';
```

```
-- Check for sufficient funds
```

```
IF (SELECT Balance FROM Accounts WHERE AccountHolder = 'Alice') < 0 THEN
```

```
    ROLLBACK;
```

```
ELSE
```

```
    COMMIT;
```

```
END IF;
```

Indexes

An **index** improves the speed of data retrieval by creating a pointer to the data in a table.

Indexes

create an index on the LastName field in the Members table to speed up searches.

```
CREATE INDEX idx_lastname ON  
Members(LastName);
```

Indexes

create an index on the LastName field in the Members table to speed up searches.

```
SELECT * FROM Members WHERE LastName =  
'Green';
```

Indexes

Viewing the Indexes

```
SHOW INDEX FROM Members;
```

Dropping an Index

```
DROP INDEX idx_lastname ON Customers;
```

Lab Exercise

- Continue working with the Tutorial 5 queries

Coming up...

ACID Properties

Further Reading :

Chapter 6 SQL: Data Manipulation

Chapter 7 SQL: Data Definition

Database Systems: A Practical Approach to Design, Implementation, and Management Sixth edition by • Thomas Connolly • Carolyn Begg