



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

Joins and its types

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

www.uwtsd.ac.uk

Learning Objectives

- Joins
- Implementation in MySQL

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

3 rows

3 rows

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)
- The Cartesian product will have **$3 \times 3 = 9$ rows**

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

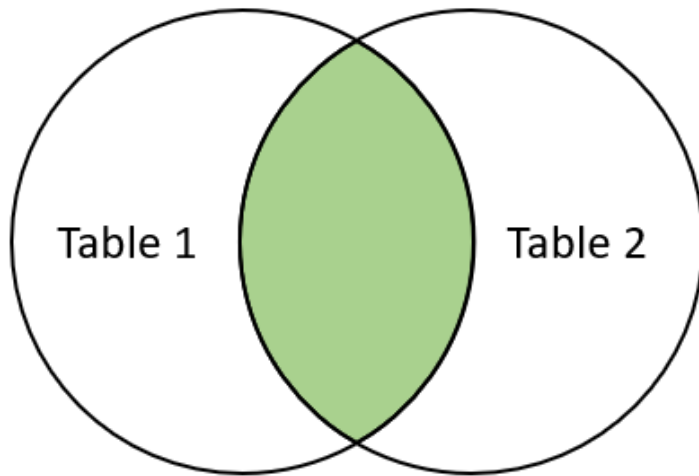
- Cartesian product creates messy results.
- Join is a **derivative of Cartesian product**, equivalent to performing a **Selection** operation, using the join predicate as the selection formula, **over the Cartesian product** of the two operand relations.
- Most difficult operations to implement efficiently 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:

- **INNER JOIN(Natural join):**

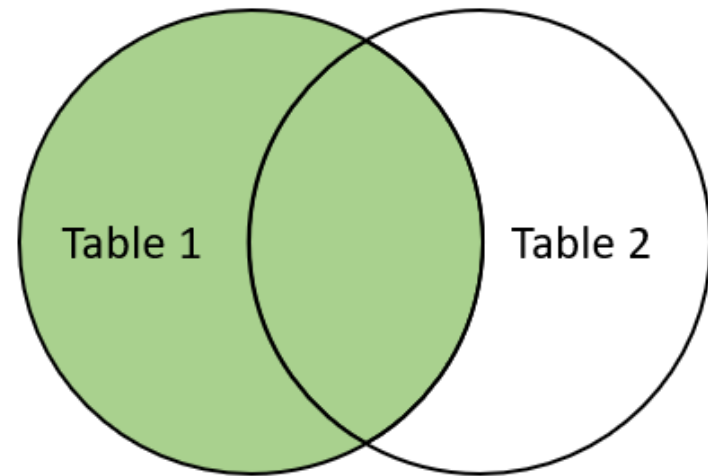
Returns records that have matching values in both tables



INNER JOIN

- **LEFT JOIN:**

Returns all records from the left table, and the matched records from the right table



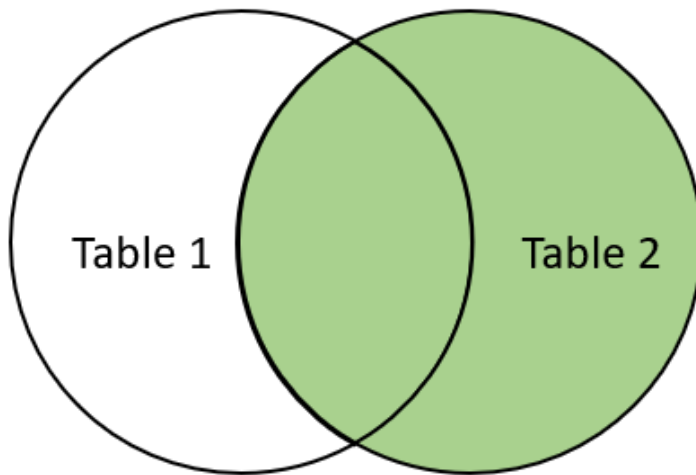
LEFT JOIN

Types of Joins

There are various forms of Join operation, each with subtle differences, some more useful than others:

- **RIGHT JOIN:**

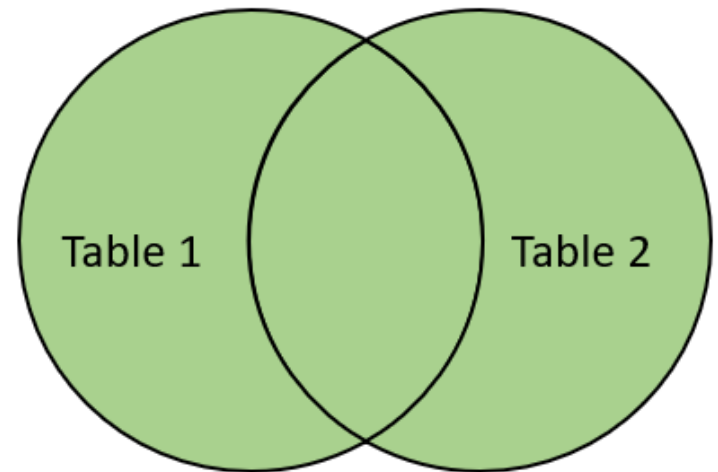
Returns all records from the right table, and the matched records from the left table



RIGHT JOIN

- **CROSS JOIN:**

Returns all records from both tables



CROSS JOIN

inner join(Natural 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:

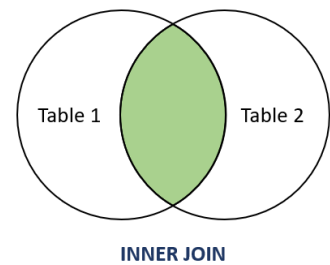
LoanID	MemberID	BookTitle	LoanDate
1	1	"Book A"	2023-10-01
2	2	"Book B"	2023-09-20
3	1	"Book C"	2023-09-28
4	3	"Book D"	2023-08-25
5	4	"Book E"	2023-10-10

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

Natural Join

SELECT * FROM members

JOIN loans ON members.MemberID = loans.MemberID;



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

MemberID	FirstName	LastName	LoanID	MemberID	BookTitle	LoanDate
1	John	Smith	1	1	Book A	2023-10-01
1	John	Smith	3	1	Book C	2023-09-28
2	Jane	Doe	2	2	Book B	2023-09-20
3	Mark	Brown	4	3	Book D	2023-08-25
4	Lisa	Green	5	4	Book E	2023-10-10

Natural Join – MySQL

StudentID	FirstName	LastName	Department
1	John	Doe	Computer Science
2	Alice	Smith	Mathematics
3	Bob	Johnson	Physics

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

Natural Join

-- Create the Students table

```
CREATE TABLE Students (  
    StudentID SERIAL 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');
```

Natural Join

-- Create the Courses table

```
CREATE TABLE Courses (  
    CourseID SERIAL 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');
```

Natural Join – MySQL

Try to perform join with SQL Query.

Natural Join – MySQL

Natural join with SQL Query.

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

Natural join with SQL Query.

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

StudentID	FirstName	LastName	Department	Courseid	CourseName
1	John	Doe	Computer Science	1	Database Management
1	John	Doe	Computer Science	3	Data Structures
2	Alice	Smith	Mathematics	2	Calculus
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:

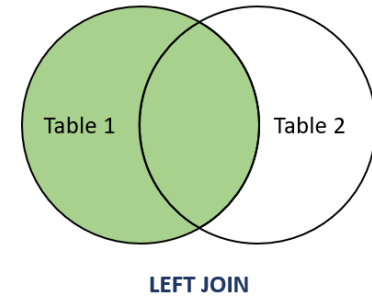
StudentID	FirstName	LastName
1	John	Doe
2	Alice	Smith
3	Bob	Johnson
4	David	White

Courses Table:

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

Left Outer Join

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

```
INSERT INTO Courses (StudentID, CourseName) VALUES  
(NULL, 'Artificial Intelligence'), -- Extra course without StudentID  
(NULL, 'Machine Learning'); -- Extra course without StudentID
```

Right Outer Join

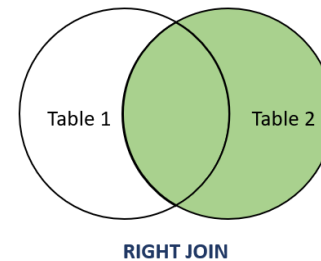
A Right Join returns all records from the Courses table (right table) and the matched records from the Students table (left table).

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

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



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.

```
SELECT
    Students.StudentID,
    Students.FirstName,
    Students.LastName,
    Courses.CourseName
FROM Students
FULL OUTER 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

Lab Exercise

- Continue working with the Tutorial 5 queries

Key points about HAVING:

HAVING is mainly used with GROUP BY.

It filters groups after aggregation.

Example: “Show me doctors with more than 10 appointments.”

```
SELECT DoctorID, COUNT (*) AS  
NumAppointments  
FROM Appointments  
GROUP BY DoctorID  
HAVING COUNT (*) > 10;
```

Here, GROUP BY creates groups per DoctorID, and HAVING filters the groups.

Do you always need GROUP BY for HAVING?

- Technically, no.

```
SELECT COUNT (*) AS  
TotalAppointments  
FROM Appointments  
HAVING COUNT (*) > 50;
```

- This counts **all appointments** and only returns a result if the count is greater than 50.

3.9 Retrieve Patients with Multiple Appointments:

```
SELECT PatientID,  
COUNT(AppointmentID) AS  
AppointmentCount  
FROM Appointment  
GROUP BY PatientID  
HAVING COUNT(AppointmentID) > 1;
```

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