Database Systems and Web (15B11CI312)

Contents to be covered

JOIN

1. Cartesian Products
2. Inner Joins (Equijoins)
3. Self Joins
4. Outer Joins (Left, Right and Full)

JOIN

- An SQL **join** clause combines records from two or more tables in a database.
- It creates a set that can be saved as a table or used as it is.
- A JOIN is a means for combining fields from two tables by using values common to each *(in most of the cases by using foreign key)*.
- A programmer writes a JOIN statement to identify the records for joining. If the evaluated predicate is true, the combined record is then produced in the expected format, a record set or a temporary table.

Types of JOINS in a table

(SQL) Joins can be classified into the following categories :

I. Cartesian Products
II. Inner Joins (Equijoins)
III. Self Joins
IV. Outer Joins (Left, Right and Full)

Cartesian Products (Cross Join)

- The Cartesian-product operation, denoted by a cross (×), allows us to combine information from any two relations.
- We write the Cartesian product of relations r1 and r2 as **r1 × r2.**
- Join without a Join Condition
- It is the base of all the other types of join.
- In other words, it will produce rows which combine each row from the first table with each row from the second table.

Source: Fundamentals of database systems / Ramez Elmasri, Shamkant B. Navathe.—7th ed, Pearson Publications

Example:

```
SELECT * FROM
  Employee CROSS JOIN Project;
```

Employee (20 rows)

SSN	Name
123	John
124	Mary
125	Mark
126	Jane
⋮	

Project (8 rows)

SSN	Code
123	DBS
124	PRG
124	DBS
126	PRG
⋮	

8 rows selected

SSN	Name	SSN	Code	
123	John	123	DBS	
124	Mary	123	DBS	
125	Mark	123	DBS	
126	Jane	123	DBS	
123	John	124	PRG	
124	Mary	124	PRG	
125	Mark	124	PRG	
126	Jane	124	PRG	
123	John	124	DBS	
124	Mary	124	DBS	
⋮				

Cartesian product:
20 x 8 = 160 rows

INNER JOIN

- An '***inner join'*** is the most common join operation used in applications and can be regarded as the *default join-type*.
- Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate.
- The result of the join can be defined as the outcome of first taking the Cartesian product (or Cross join) of all records in the tables (combining every record in table A with every record in table B) and then returning all records which satisfy the join predicate.

Inner join Cont.

- The two tables must be *joined* by at *least one common field*. That is, the *join field* is a member of both tables.
- **Syntax:**

  Select * from A, B where A.x = B.y

  The column names (x and y in this example) are often, but not necessarily, the same.
- Example:

SELECT EmployeeName, DeptName FROM **Employee *INNER JOIN* Department** ON Employee.DeptID = Department.DeptID;

Source: Database System Concepts / Silberschatz−Korth−Sudarshan

Notations

1. The *"explicit join notation"* uses the **JOIN** keyword, optionally preceded by the **INNER** keyword, to specify the table to join, and the **ON** keyword to specify the predicates for the join:

```
SELECT * FROM employee INNER JOIN department ON
        employee.DepartmentID = department.DepartmentID;
```

2. The "implicit join notation" simply lists the tables for joining without using JOIN keyword.

```
SELECT * FROM employee, department WHERE
    employee.DepartmentID = department.DepartmentID;
```

Join Clauses

1. Using (A1,A2,A3…An) where A1,A2,….An are attributes
2. ON(Predicate)

Creating Joins with the Using Clause

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code
123	DBS
124	PRG
124	DBS
126	PRG

```
SELECT * FROM

  Student INNER JOIN
 Enrolment USING (ID)
```

ID	Name	ID	Code	
123	John	123	DBS	
124	Mary	124	PRG	
124	Mary	124	DBS	
126	Jane	126	PRG	

Creating Joins with the `ON` Clause

- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

Example with on clause:

Buyer

Name	Budget
Smith	100,000
Jones	150,000
Green	80,000

```
SELECT * FROM Buyer as B,
 Property as P

 ON P.Price <= B.Budget
```

Property

Address	Price
15 High St	85,000
12 Queen St	125,000
87 Oak Row	175,000

Name	Budget	Address	Price
Smith	100,000	15 High St	85,000
Jones	150,000	15 High St	85,000
Jones	150,000	12 Queen St	125,000

USING	ON

SELECT * FROM
A INNER JOIN B
USING(col1, col2,...)
is the same as
SELECT * FROM A, B
WHERE A.col1 = B.col1
AND A.col2 = B.col2
AND ...
	SELECT * FROM
	A INNER JOIN B
	ON <condition>
	is the same as
	SELECT * FROM A, B
	WHERE <condition>

Equijoins

- An **Equi-join** is a specific type of comparator-based join, that uses only **equality** comparisons in the join-predicate.
- Using other comparison operators (such as <) disqualifies a join as an Equi-join.
- The order of the tables listed in the FROM clause should have no significance.

Natural join

A natural join is *a type of Equi-join* that only work if the **column** you are joining by has **same name** in both tables.

The resulting joined table contains only one column for each pair of equally named columns.

NATURAL JOIN

EMPLOYEE

SSN	Name
123	John
124	Mary
125	Mark
126	Jane

PROJECT

SSN	Code
123	DBS
124	PRG
124	DBS
126	PRG

SELECT * FROM

Employee **NATURAL JOIN** Project;

SSN	NAME	CODE
123	John	DBS
124	Mary	PRG
124	Mary	DBS
126	Jane	PRG

The join condition for the natural join is basically an **equijoin of all columns with the same name**.

Self Join

- A Self Join is a join of a table to itse
- Put the table in the FROM clause twice.
- Use aliases to distinguish columns in the WHERE clause.
- Syntax:

```
SELECT a.column_name, b.column_name
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field;
```

A query to find all pairings of two employees in the same country is desired.

```
SELECT e1.EmployeeID, e1.LastName, e2.EmployeeID, e2.LastName, e2.Country
FROM   Employee As e1, Employee As e2 WHERE e1.Country = e2.Country;
```

Can also write like this: without writing As

```
Employee  e1, Employee  e2
```

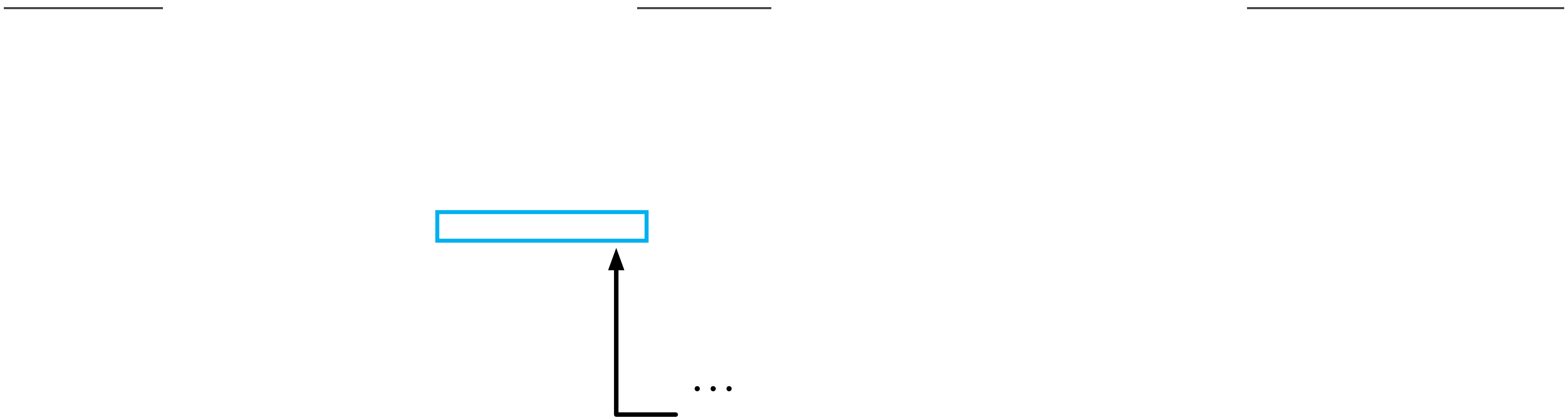
Outer Join

- An Inner Join excludes rows from either table that don't have a matching row in the other table.
- *An Outer Join allows us to return unmatched rows.*
- Outer Joins come in three varieties :
  - **LEFT** (MATCHED ROWS IN RIGHT TABLE AND ALL ROWS IN LEFT TABLE)
  - **RIGHT** (MATCHED ROWS IN LEFT TABLE AND ALL ROWS IN RIGHT TABLE)
  - **FULL** (ALL ROWS IN ALL TABLES IT DOESN'T MATTERS EVEN MATCH IS THERE OR NOT)

Outer Joins

DEPARTMENTS

EMPLOYEES



There are no employees in department 190.

Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row does not appear in the query result. For example, in the equijoin condition of `EMPLOYEES` and `DEPARTMENTS` tables, department ID 190 does not appear because there are no employees with that department ID recorded in the `EMPLOYEES` table. Instead of seeing 20 employees in the result set, you see 19 records.

To return the department record that does not have any employees, you can use an outer join.

LEFT OUTER JOIN

A left outer join will give all rows in A, plus any common rows in B

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

…

This query retrieves all rows in the `EMPLOYEES` table, which is the left table even if there is no match in the `DEPARTMENTS` table.

RIGHT OUTER JOIN

```
SELECT e.last name, e.department id, d.department name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

…

This query retrieves all rows in the `DEPARTMENTS` table, which is the right table even if there is no match in the `EMPLOYEES` table.

FULL OUTER JOIN

The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side