More SQL: Complex Queries

More Complex SQL Retrieval Queries

- Additional features allow users to specify more complex retrievals from database:
  - Nested queries
  - JOINS
  - Aggregate functions
  - Grouping

Nested Queries

- **Nested/Sub queries**
  - A query within a query
  - Useful when condition has to be applied against an unknown value
  - E.g. Get the names of those students who have more *cgpa* than that of *maximum* of BCS students
- Comparison operator **`IN`**
  - Complete select-from-where blocks within WHERE clause of another query called **Outer query**
  - Compares value *v* with a set (or multiset) of values *V*
  - Evaluates to `TRUE` if *v* is one of the elements in *V*

Nested Queries Example (1)

- **Example:** Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT Fname, Lname, Address
FROM Employee
WHERE Dno IN (SELECT Dnumber
FROM Department
WHERE Dname='Research')
```

- Sub query executed one time before the outer query
- May return single attribute and single tuple or multiple
- Output is temporary dataset used by outer query

Nested Queries Example (2)

- **Retrieve project no of ‘smith’ where he worked as manager or worker**

```
SELECT DISTINCT Pnumber
FROM PROJECT
WHERE Pnumber IN
                ( SELECT Pnumber
                FROM PROJECT, DEPARTMENT, EMPLOYEE
                WHERE Dnum=Dnumber AND mgrssn=Ssn AND
                Lname='Smith' )
                OR Pnumber IN
                ( SELECT Pno
                FROM WORKS_ON, EMPLOYEE
                WHERE Essn=Ssn AND Lname='Smith' );
```

	Pnumber
1	1
2	2

Nested Queries (Tuple of values)

- Can use Tuples of values in comparisons
  - Place them within parentheses
- **Example:** Retrieve Essn of all employee who work in same (Project,Hour) combination on some project that employee whose ssn= ‘12345678

```
SELECT DISTINCT Essn
FROM WORKS_ON
WHERE (Pno, Hours) IN ( SELECT Pno, Hours
FROM WORKS_ON
WHERE Essn='123456789');
```

Nested Queries (Comparison Operator)

- Other comparison operators instead of IN can be:
  - ANY or SOME operator returns TRUE if the value *v* is equal to *some value* in the set *V* and is equivalent to IN.
  - Other operators that can be combined with ANY (or SOME) include >, >=, <, <=, and <>
  - The keyword ALL can also be combined with each of these operators.
- **Example :** Returns the names of employees whose salary is greater than the salary of all the employees in department 5:

```
SELECT Lname, Fname
FROM EMPLOYEE
 WHERE Salary > ALL ( SELECT Salary
 FROM EMPLOYEE WHERE Dno=5 );
```

Accessing Multiple Tables

- Cartesian Product
- Inner join
- Natural Join
- Outer Joins
- Self Join
- Semi Join

Cartesian Product

- No specific command, Simply use:
  - SELECT command without specifying WHERE clause
- Resultant table :
  - Cardinality (no. of rows): m x n rows
  - Degree (no. of Attributes): m+ n Attributes
- Example:

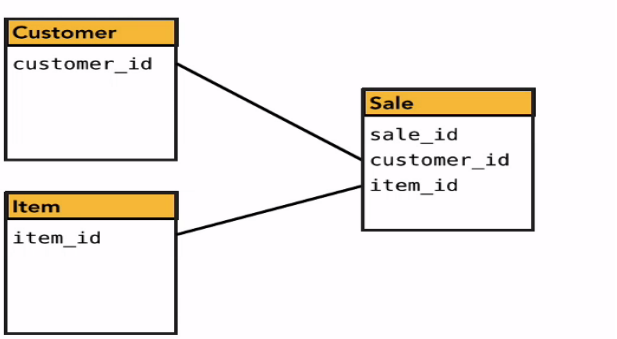
  SELECT * FROM Employee, Department

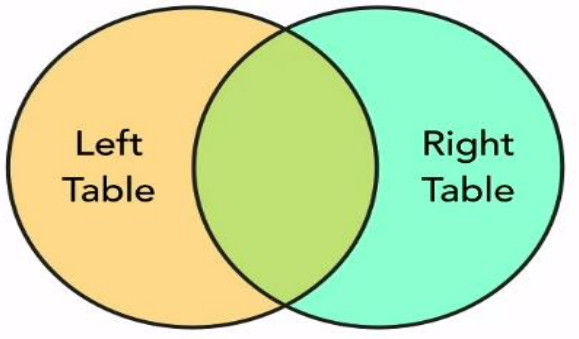
  SELECT * from program, course

Table Relationship

- Some table contain information related to other tables
  - Using Join statement SQL is powerful tool to extract related data from multiple tables
  - The intersection of shape where record overlap or condition is match
  - Id field is often use where condition to be match

Left.Id= Right.Id



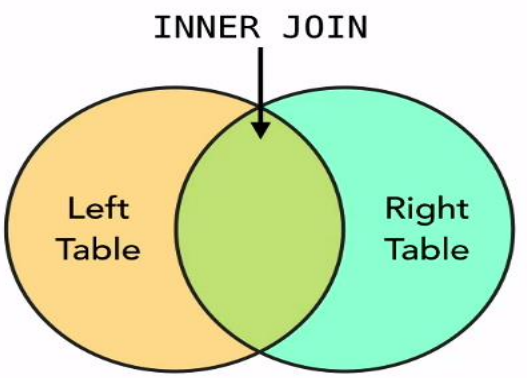


INNER JOIN

- The result of inner join includes row from both table where join condition is met (equi Join)
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation

SELECT Fname, Lname, Address
FROM (EMPLOYEE **JOIN**
DEPARTMENT **ON** Dno=Dnumber)
WHERE Dname='Research';

SELECT Fname, Lname, Address
FROM (EMPLOYEE INNER **JOIN**
DEPARTMENT **ON** Dno=Dnumber)
WHERE Dname='Research';



NATURAL JOIN

- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Implicit EQUIJOIN condition for each pair of attributes with same name from R and S

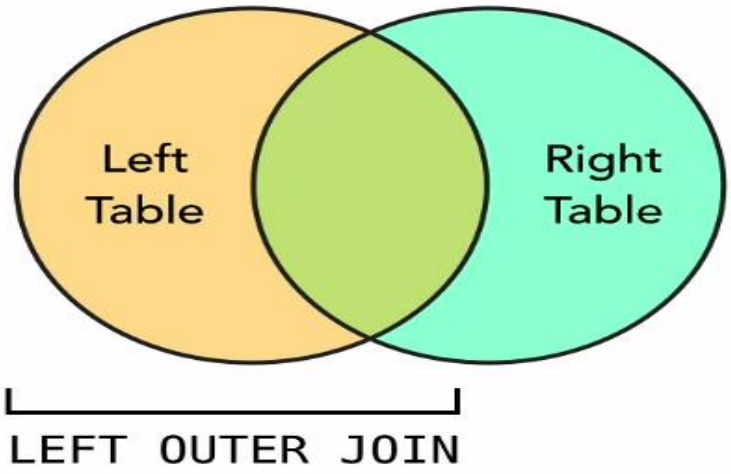
SELECT Fname, Lname, Address
FROM (EMPLOYEE **NATURAL JOIN** DEPARTMENT)
WHERE Dname='Research';

LEFT OUTER JOIN

- Includes rows where condition is met + All the rows from left where condition is not met

Syntax:

SELECT*
FROM *table_name*
LEFT OUTER JOIN *Table_name*
ON *(Predicate)*



Left Outer Join Example

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

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
John	NULL	NULL	NULL
Steinberg	33	Engineering	33

Employee Table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table

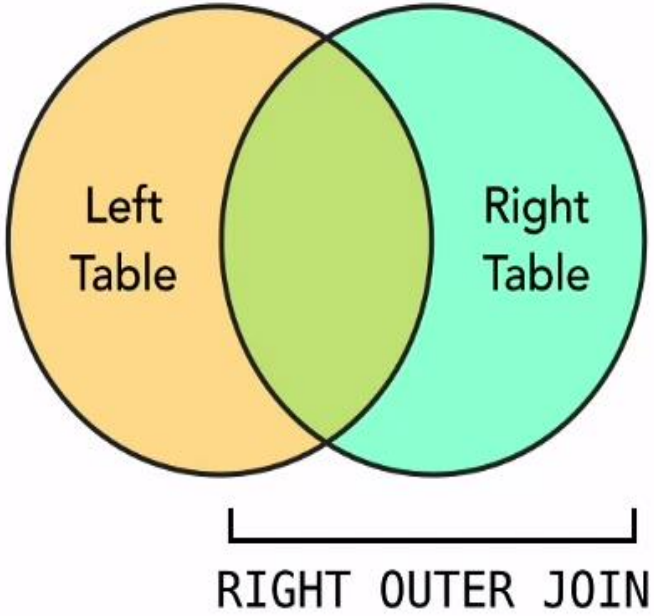
DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

RIGHT OUTER JOIN

- Includes rows where condition is met + All the rows from right table where condition is not met

Syntax:

SELECT*
FROM *table_name*
RIGHT OUTER JOIN *Table_name*
ON *(Predicate)*



Right Outer Join Example

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

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	35

Employee Table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table

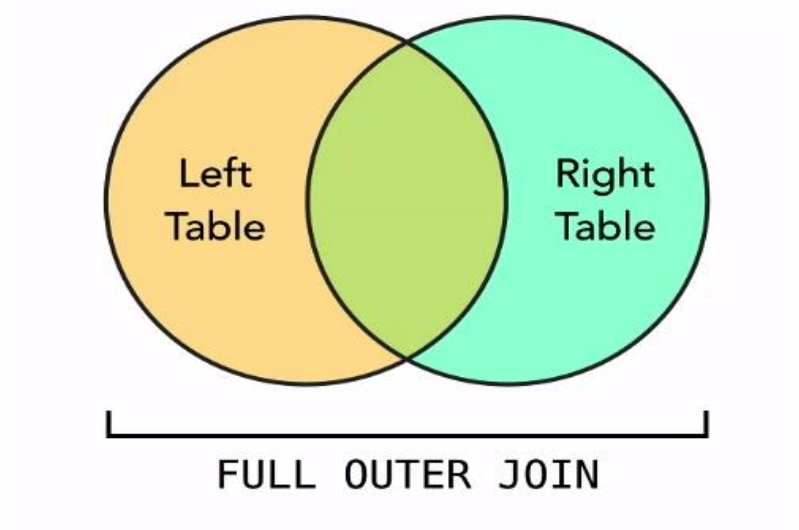
DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

FULL OUTER JOIN

- All rows from both table + rows where condition is met

Syntax:

SELECT*
FROM *table_name*
FULL OUTER JOIN *Table_name*
ON *(Predicate)*



Full Outer Join Example

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

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
John	NULL	NULL	NULL
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

Employee Table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Aggregate Functions in SQL

- Aggregate data is information derived from more than one row at a time
  - Used to summarize information
  - Operate on a set of rows and return a single value, like, AVG, SUM, MAX, MIN, STDEV, COUNT

- **Grouping** is used to create subgroups of tuples before summarization.
- Functions can be used in the **`SELECT`** clause or in a **`HAVING`** clause

Aggregate Functions: Examples

- **Query:** Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

  SELECT **SUM** (Salary), **MAX** (Salary), **MIN** (Salary), **AVG** (Salary)
  FROM EMPLOYEE;

	SUMSalary	MAXSalary	MINSalary	AVGSalary
1	2041000.00	96000.00	25000.00	48595.238095

- **Query:** **SELECT COUNT** (*) AS TotalEmp
  FROM EMPLOYEE;

	TotalEmp
1	42

- **Query:** **SELECT COUNT** (**DISTINCT** Salary) As DISTINCTSalary
  FROM EMPLOYEE;

	DISTINCTSalary
1	32

GROUP BY

- **`GROUP BY`** clause
  - Specifies grouping attributes
  - **Partition** relation into subsets of tuples based on **grouping attribute(s)**
  - Aggregate functions (like SUM) return the aggregate of all column values every time they are called, and without the GROUP BY function it is impossible to find the sum for each individual group of column values.
  - Apply function to each such group independently

Syntax: SELECT column1, column2, ... column_n,
aggregate_function (expression)
FROM tables WHERE predicates
GROUP BY column1, column2, ... column_n;

GROUP BY: Example

- If NULLs exist in grouping attribute
  - Separate group created for all tuples with a NULL value in grouping attribute
- Query: For each department, retrieve the *department number*, the *number of employees* in the department, and their *average salary*.

SELECT Dno, **COUNT** (*) AS TotalEmp, **AVG** (Salary) AS AVGSalary
FROM EMPLOYEE
GROUP BY Dno;

	Dno	TotalEmp	AVGSalary
1	NULL	5	28600.000000
2	1	1	55000.000000
3	4	3	39000.000000
4	5	1	40000.000000
5	6	8	60000.000000
6	7	10	63450.000000
7	8	14	40821.428571

HAVING Clause

- HAVING Clause
  - Restrict groups by satisfying having condition will be selected
  - Provides a condition on aggregate data
  - WHERE clause provide condition on non aggregate data(rows)
- Example: For each project *on which more than two employees work*, retrieve the *project number*, the *project name*, and the number of employees who work on the project.

```
SELECT Pnumber, Pname, COUNT (*) AS T
FROM PROJECT, WORKS_ON
WHERE Pnumber=Pno
GROUP BY Pnumber, Pname
HAVING COUNT (*) > 2;
```

	Pnumber	Pname	TotalEmp
1	10	Computerization	3
2	62	DatabaseSystems	8
3	91	InkjetPrinters	8
4	92	LaserPrinters	3
5	63	Middleware	4
6	30	Newbenefits	3
7	61	OperatingSystems	9
8	2	ProductY	3
9	20	Reorganization	3

Discussion and Summary of SQL Queries

SELECT <attribute and function list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>];