3ISY402 – DATABASE SYSTEMS

Lecture 5 –

SQL: Complex DML – Multi-Table Joins

Material from essential text:

T CONNOLLY & C BEGG. Database Systems – A Practical Approach to Design, Implementation and Management, 4th Edition. Addison-Wesley, 2005.

Leena Gulabivala

Lecture - Objectives

- How to retrieve data from database using SELECT statement to create complex queries:
  - Join tables together.
  - Inner and Outer Joins.
  - Union, Intersect and Difference.

Multi-Table Queries

- Can use subqueries provided that the result columns come from the same table.
- *If result columns come from more than one table must use a join.*
- To perform a join, include more than one table in the FROM clause and use comma as separator.
- Typically include WHERE clause to specify join column(s) – known as the *join-condition*.
- To join *two* tables, specify *one* join-condition.
- To join *three* tables, specify *two* join-conditions.
- To join *four* tables, specify *three* join-conditions.

Multi-Table Queries - Alias

- Also possible to use an *alias* for a table named in FROM clause.
- Alias is separated from table name with a space.
- Alias can be used to qualify column names when there is ambiguity.

Example 12 - Simple Join

- List names of all clients who have viewed a property along with any comment supplied.

```
SELECT   c.clientNo, fName, lName,
         propertyNo, comment
FROM     Client c,
         Viewing v
WHERE    c.clientNo = v.clientNo;
```

Join condition

Example 12 - Simple Join

- To obtain correct result, only those rows from both tables that have identical values in the clientNo columns (c.clientNo = v.clientNo) are included in result.
- Equivalent to *equi-join* in relational algebra.

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

Example 13 - Sorting a join

- For each branch, list numbers and names of staff who manage properties, and the properties they manage.

```
SELECT   s.branchNo, s.staffNo, s.fName, s.lName,
         p.propertyNo
FROM     Staff s,
         PropertyForRent p
WHERE    s.staffNo = p.staffNo
ORDER BY s.branchNo, s.staffNo, p.propertyNo;
```

Example 13 - Sorting a join

branchNo	staffNo	fName	lName	propertyNo
B003	SG14	David	Ford	PG16
B003	SG37	Ann	Beech	PG21
B003	SG37	Ann	Beech	PG36
B005	SL41	Julie	Lee	PL94
B007	SA9	Mary	Howe	PA14

Example 14 - Three Table Join

- For each branch, list the staff who manage the properties, including the city in which the branch is located and the properties they manage.
- *Can write query using aliases for table names.*

```
SELECT    b.branchNo, b.city, s.staffNo, s.fName,
          s.lName, p.propertyNo
FROM      Branch b,
          Staff s,
          PropertyForRent p
WHERE     b.branchNo    = s.branchNo
AND       s.staffNo     = p.staffNo
ORDER BY b.branchNo, s.staffNo, p.propertyNo;
```

Example 14 - Three Table Join

- ***OR**, can write query using table names to prefix each column.*

```
SELECT    Branch.branchNo, Branch.city, Staff.staffNo,
          Staff.fName, Staff.lName,
          PropertyForRent.propertyNo
FROM      Branch b,
          Staff s,
          PropertyForRent p
WHERE     Branch.branchNo = Staff.branchNo
AND       Staff.staffNo = PropertyForRent.staffNo
ORDER BY  Branch.branchNo, Staff.staffNo,
          PropertyForRent.propertyNo;
```

Example 14 - Three Table Join

branchNo	city	staffNo	fName	lName	propertyNo
B003	Glasgow	SG14	David	Ford	PG16
B003	Glasgow	SG37	Ann	Beech	PG21
B003	Glasgow	SG37	Ann	Beech	PG36
B005	London	SL41	Julie	Lee	PL94
B007	Aberdeen	SA9	Mary	Howe	PA14

Example 15 - Multiple Grouping Columns

- Find the number of properties handled by each staff member.

```
SELECT    s.branchNo, s.staffNo,
          COUNT(*) AS myCount
FROM      Staff s,
          PropertyForRent p
WHERE     s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```

branchNo	staffNo	myCount
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Computing a Join

- Procedure for generating results of a join are:
  - Form Cartesian product of the tables named in FROM clause.
  - If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.
  - For each remaining row, determine value of each item in SELECT list to produce a single row in result table.
  - If DISTINCT has been specified, eliminate any duplicate rows from the result table.
  - If there is an ORDER BY clause, sort result table as required.

Outer Joins

- *Inner Join is where the rows match.*
- If one row of a joined table is unmatched, row is omitted from result table.
- *Outer join operations retain rows that DO NOT satisfy the join condition.*
- Consider following tables:

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Outer Joins

- The (inner) join of these two tables:

```
SELECT  b.*, p.*
FROM    Branch1 b,
        PropertyForRent1 p
WHERE   b.bCity = p.pCity;
```

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Outer Joins

- Result table has two rows where cities are same.
- There are no rows corresponding to branches in Bristol and Aberdeen.
- To include unmatched rows in result table, use an Outer join.
- Use the **(+)** which has the effect of generating a special NULL value row for the table to which it is applied.
- inner_table.join_column = outer_table.join_column(+)
- The outer join operator **(+)** ensures the insertion of a NULL value for the columns in the outer table that do not have matching rows in the inner table.

Example 16 - Left Outer Join

- List branches and properties that are in the same city along with any unmatched branches.

```
SELECT  b.*, p.*
FROM    Branch1 b,
        PropertyForRent1 p
WHERE   b.bCity = p.pCity(+) ;
```

- Alternative Syntax:

```
SELECT  b.*, p.*
FROM    Branch1 b LEFT JOIN
        PropertyForRent1 p ON b.bCity = p.pCity;
```

Example 16 - Left Outer Join

- Includes those rows of first (left) table unmatched with rows from second (right) table.
- Columns from second table are filled with NULLs.

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

Example 17 - Right Outer Join

- List branches and properties in the same city and any unmatched properties.

```
SELECT  b.*, p.*
FROM    Branch1 b,
        PropertyForRent1 p
WHERE   b.bCity(+) = p.pCity ;
```

- Alternative Syntax:

```
SELECT  b.*, p.*
FROM    Branch1 b RIGHT JOIN
        PropertyForRent1 p ON b.bCity = p.pCity;
```

Example 17 - Right Outer Join

- Right Outer join includes those rows of second (right) table that are unmatched with rows from first (left) table.
- Columns from first table are filled with NULLs.

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

EXISTS and NOT EXISTS

- EXISTS and NOT EXISTS are for use only with subqueries.
- Produce a simple true/false result.
- True if and only if there exists at least one row in result table returned by subquery.
- False if subquery returns an empty result table.
- NOT EXISTS is the opposite of EXISTS.
- As (NOT) EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.
- Common for subqueries following (NOT) EXISTS to be of form: (SELECT * ...)

Example 18 - Query using EXISTS

- Find all staff who work in a London branch.

```
SELECT    staffNo, fName, lName, position
FROM      Staff s
WHERE EXISTS
          (SELECT    *
          FROM       Branch b
          WHERE      s.branchNo = b.branchNo
          AND        city = 'London');
```

staffNo	fName	lName	position
SL21	John	White	Manager
SL41	Julie	Lee	Assistant

Example 18 - Query using EXISTS

- Note, first part of search condition s.branchNo = b.branchNo is necessary to consider correct branch record for each member of staff.
- If omitted, would get all staff records listed out because subquery:

```
SELECT   *
FROM     Branch
WHERE    city='London'
```

- would always be true and query would be:

```
SELECT   staffNo, fName, lName, position
FROM     Staff
WHERE    true;
```

Example 18 - Query using EXISTS

- Could also write this query using the join construct:

```
SELECT  staffNo, fName, lName, position
FROM    Staff s,
        Branch b
WHERE   s.branchNo = b.branchNo
AND     city = 'London';
```

Union, Intersect, and Difference (Except)

- Can use normal set operations of *Union*, *Intersection*, and *Difference* to combine results of two or more queries into a single result table.
- *Union* of two tables, A and B, is table containing all rows in either A or B or both.
- *Intersection* is table containing all rows common to both A and B.
- *Difference* is table containing all rows in A but not in B.
- Two tables must be *union compatible*.

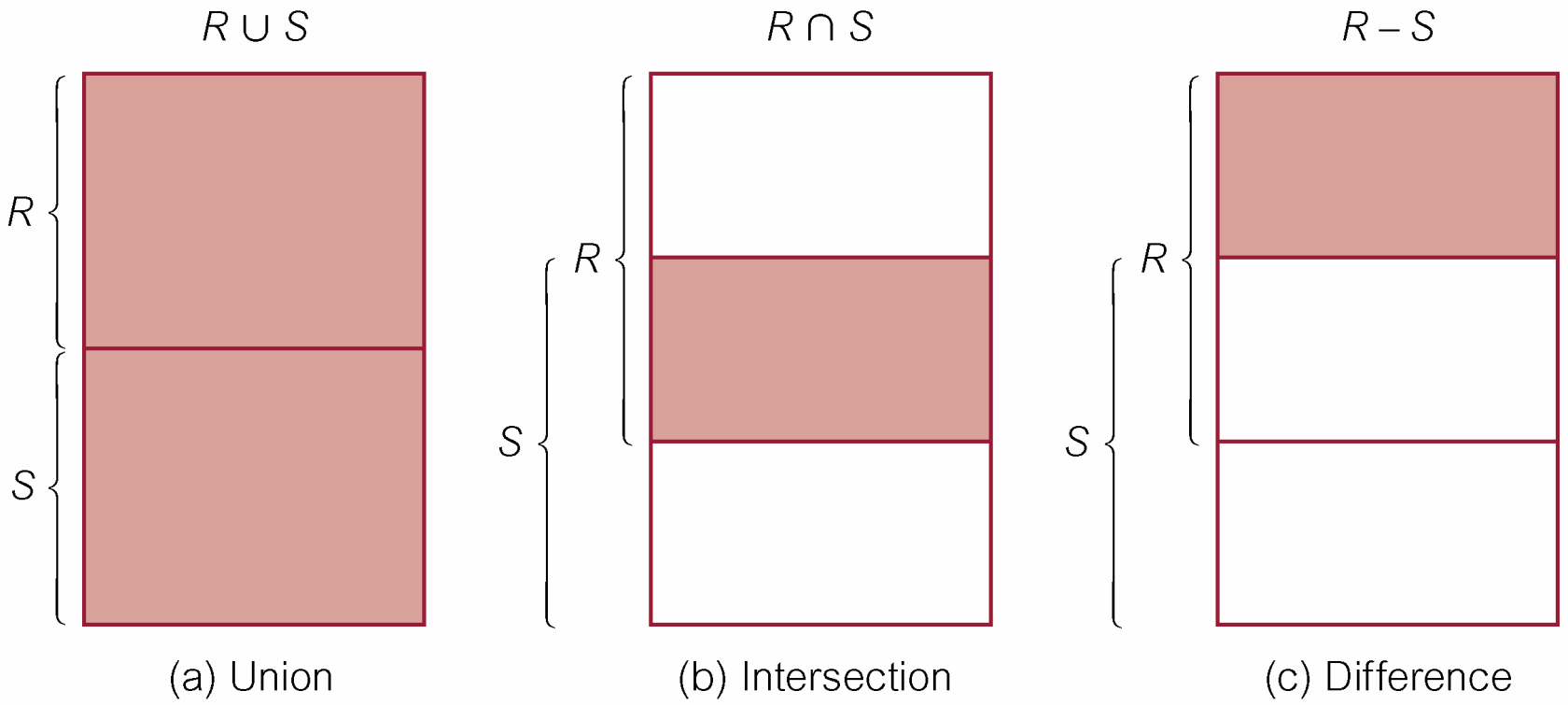
Union, Intersect, and Difference (Except)

- Format of set operator clause in each case is:

  op [ALL] [CORRESPONDING [BY {column1 [, ...]}]]

- If CORRESPONDING BY specified, set operation performed on the named column(s).
- If CORRESPONDING specified but not BY clause, operation performed on common columns.
- If ALL specified, result can include duplicate rows.

Union, Intersect, and Difference (Except)



Example 19 - Use of UNION

- List all cities where there is either a branch office or a property.

```
(SELECT     city
 FROM       Branch
 WHERE      city IS NOT NULL)
 UNION
(SELECT     city
 FROM       PropertyForRent
 WHERE      city IS NOT NULL);
```

Example 19 - Use of UNION

- *Or*

```
(SELECT      *
 FROM        Branch
 WHERE       city IS NOT NULL)
 UNION CORRESPONDING BY city
(SELECT      *
 FROM        PropertyForRent
 WHERE       city IS NOT NULL);
```

city
London
Glasgow
Aberdeen
Bristol

- Produces result tables from both queries and merges both tables together.

Example 20 - Use of INTERSECT

- List all cities where there is both a branch office and a property.

```
(SELECT city FROM Branch)
INTERSECT
(SELECT city FROM PropertyForRent);
```

city
Aberdeen
Glasgow
London

- ***Or:***

```
(SELECT * FROM Branch)
INTERSECT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```

Example 20 - Use of INTERSECT

- Could rewrite this query without INTERSECT operator:

```
SELECT  b.city
FROM    Branch b,
        PropertyForRent p
WHERE   b.city = p.city;
```

- ***Or***

```
SELECT  DISTINCT city
FROM    Branch b
WHERE EXISTS
        (SELECT     *
         FROM       PropertyForRent p
         WHERE      p.city = b.city);
```

Example 21 - Use of EXCEPT

- List of all cities where there is a branch office but no properties.

```
(SELECT city FROM Branch)
EXCEPT
(SELECT city FROM PropertyForRent);
```

- ***Or***

```
(SELECT * FROM Branch)
EXCEPT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```

city
Bristol

Example 21 - Use of EXCEPT

- Could rewrite this query without EXCEPT (using subquery):

```
SELECT DISTINCT city FROM Branch
WHERE city NOT IN
        (SELECT city FROM PropertyForRent);
```

- ***Or***

```
SELECT DISTINCT city FROM Branch b
WHERE NOT EXISTS
        (SELECT *
         FROM PropertyForRent p
         WHERE p.city = b.city);
```

References and Further Reading:

- T CONNOLLY & C BEGG. *Database Systems – A Practical Approach to Design, Implementation and Management,* 4th Edition. Addison-Wesley, 2005.
  - Chapter 5 – SQL Data Manipulation.
- T CONNOLLY & C BEGG. *Database Solutions – A step-by-step guide to building databases,* 2nd Edition. Addison-Wesley, 2004.
  - Chapter on SQL Data Manipulation.