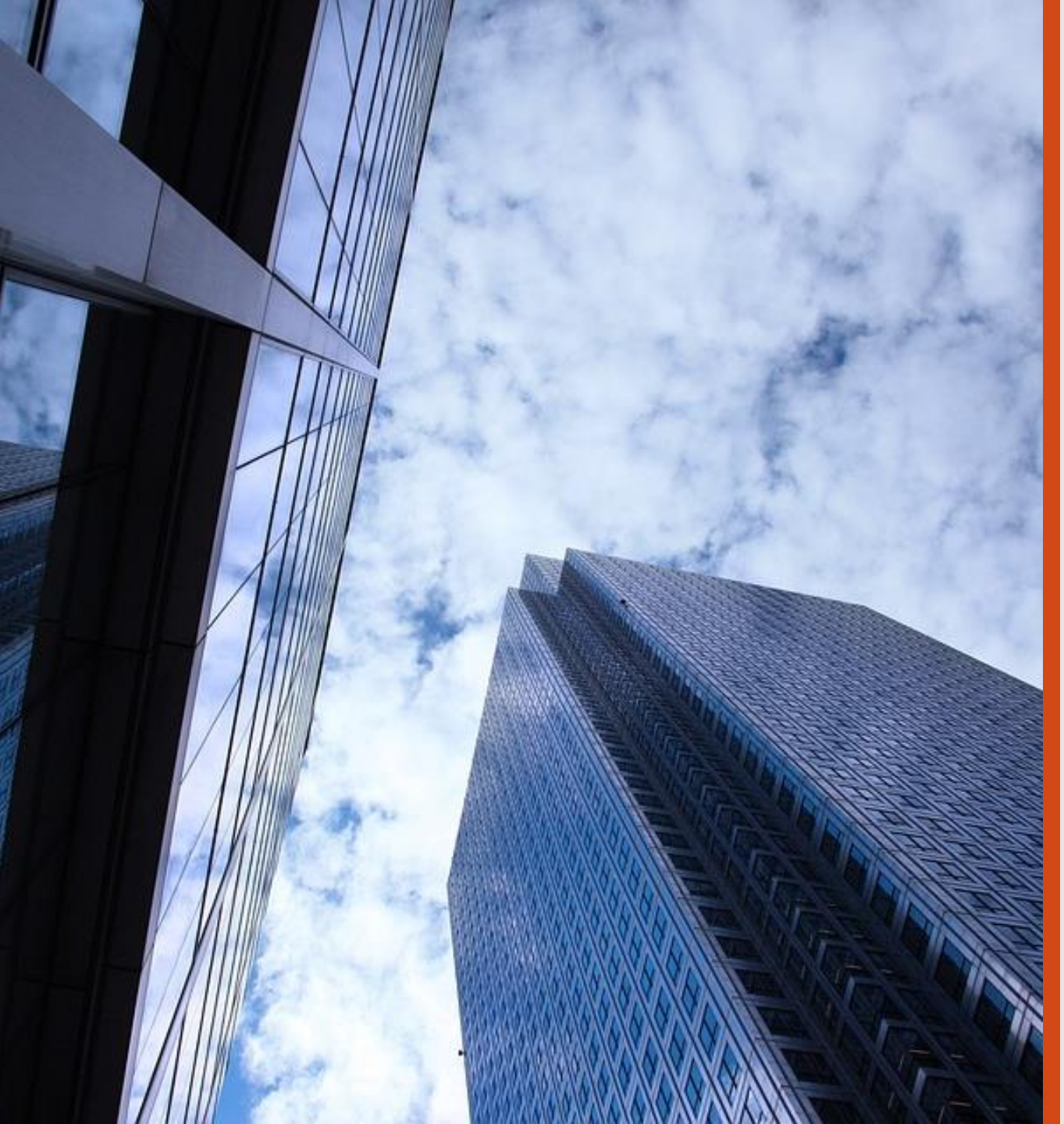


SQL
PART 04

Jayathma Chathurangani

ejc@ucsc.cmb.ac.lk

OUTLINE

DML Continues

- **Sub Queries**
- **ANY and ALL**
- **Multi Table Queries**
- **EXISTS and NOT EXISTS**
- **Combining Result Tables**

1

SUB QUERIES

1.1 INTRODUCTION

- In this section we examine the use of a complete SELECT statement embedded within another SELECT statement.
- The results of this **inner SELECT** statement (or **subselect**) are used in the outer statement to help determine the contents of the final result.
- A sub-select can be used in the WHERE and HAVING clauses of an outer SELECT statement, where it is called a **subquery or nested query**.
- There are three types of subquery:
  1. **A scalar subquery** returns a single column and a single row, that is, **a single value**. In principle, a scalar subquery can be used whenever a single value is needed.
  2. **A row subquery** returns multiple columns, but only **a single row**. A row subquery can be used whenever a row value constructor is needed, typically in predicates.
  3. **A table subquery** returns **one or more columns and multiple rows**. A table subquery can be used whenever a table is needed, for example, as an operand for the IN predicate.

1.1 INTRODUCTION (CONTINUED)

- Subselects may also appear in INSERT, UPDATE, and DELETE statements.
- We can think of the subquery as producing a temporary table with results that can be accessed and used by the outer statement.
- A subquery can be used immediately following a relational operator (=, <, >, <=, > =,< >) in a WHERE clause, or a HAVING clause.
- The subquery itself is always enclosed in parentheses.
- By default, column names in a subquery refer to the table name in the FROM clause of the subquery. It is possible to refer to a table in a FROM clause of an outer query by qualifying the column name.

1.2 USING A SUBQUERY WITH EQUALITY

Query: List the staff who work in the branch at '163 Main St'.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = (SELECT branchNo
                  FROM Branch
                  WHERE street = '163 Main St');
```

- The outer SELECT reduced to:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = 'B003';
```

staffNo	fName	lName	position
SG37	Ann	Beech	Assistant
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

1.3 USING A SUBQUERY WITH AN AGGREGATE FUNCTION

Query: List all staff whose salary is greater than the average salary, and show by how much their salary is greater than the average.

- (When a subquery is one of the two operands involved in a comparison, the subquery must appear on the right-hand side of the comparison.)

```
SELECT staffNo, fName, lName, position,
       salary – (SELECT AVG(salary) FROM Staff) AS salDiff
FROM Staff
WHERE salary > (SELECT AVG(salary) FROM Staff);
```

- The outer SELECT reduced to:

```
SELECT staffNo, fName, lName, position, salary – 17000 AS salDiff
FROM Staff
WHERE salary > 17000;
```

staffNo	fName	lName	position	salDiff
SL21	John	White	Manager	13000.00
SG14	David	Ford	Supervisor	1000.00
SG5	Susan	Brand	Manager	7000.00

1.4 NESTED SUBQUERIES: USE OF IN

- There may be cases where more than one such row found, and so we cannot use the equality condition (=) in the outermost query. Instead, we use the IN keyword.

Query: List the properties that are handled by staff who work in the branch at '163 Main St'.

```
SELECT propertyNo, street, city, postcode, type, rooms, rent
FROM PropertyForRent
WHERE staffNo IN (SELECT staffNo
                  FROM Staff
                  WHERE branchNo = (SELECT branchNo
                                    FROM Branch
                                    WHERE street = '163 Main St'));
```

propertyNo	street	city	postcode	type	rooms	rent
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375
PG21	18 Dale Rd	Glasgow	G12	House	5	600

2

ANY AND ALL

2.1 INTRODUCTION

- The keywords **ANY and ALL** **may be used with subqueries that produce a single column of numbers**.
- If the subquery is preceded by the keyword **ALL**, the condition will be true only if it is satisfied by all values produced by the subquery.
- If the subquery is preceded by the keyword **ANY**, the condition will be true if it is satisfied by any (one or more) values produced by the subquery.
- **If the subquery is empty, the ALL condition returns true, the ANY condition returns false.**
- The ISO standard also allows the qualifier SOME to be used in place of ANY.

2.2 USE OF ANY/SOME

- SQL ANY compares a value of the first table with all values of the second table and returns the row if there is a match with any value.

Query: Find all staff whose salary is larger than the salary of at least one member of staff at branch B003.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > SOME (SELECT salary
                     FROM Staff
                     WHERE branchNo = 'B003');
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

- Although this query can be expressed using a subquery that finds the minimum salary of the staff at branch B003 and then an outer query that finds all staff whose salary is greater than this number, an alternative approach uses the SOME/ANY keyword.
- The inner query produces the set {12000, 18000, 24000} and the outer query selects those staff whose salaries are greater than any of the values in this set (that is, greater than the minimum value, 12000).

2.3 USE OF ALL

- SQL ALL compares a value of the first table with all values of the second table and returns the row if there is a match with all values.

Query: Find all staff whose salary is larger than the salary of every member of staff at branch B003.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > ALL (SELECT salary
                    FROM Staff
                    WHERE branchNo = 'B003');
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00

- This example is very similar to the previous example.
- Again, we could use a subquery to find the maximum salary of staff at branch B003 and then use an outer query to find all staff whose salary is greater than this number.
- The inner query produces the set {12000, 18000, 24000} and the outer query selects those staff whose salaries are greater than all the values in this set (that is, greater than the maximum value, 24000).

3

MULTI TABLE QUERIES

3.1 INTRODUCTION

- The SQL join operation combines information from two tables by forming pairs of related rows from the two tables.
- The row pairs that make up the joined table are those where the matching columns in each of the two tables have the same value.
- If we need to obtain information from more than one table, the choice is between using a subquery and using a join.
- If the final result table is to contain columns from different tables, then we must use a join.

3.1 INTRODUCTION (CONTINUED)

- The most common multi-table queries involve two tables that have a one-to-many (1:*) (or a parent/child) relationship.
- Each viewing (child) has an associated client (parent), and each client (parent) can have many associated viewings (children).
- **The pairs of rows that generate the query results are parent/child row combinations.**
- **The table containing the primary key is the parent table and the table containing the foreign key is the child table.**
- To use the parent/child relationship in an SQL query, we specify a search condition that compares the primary key and the foreign key.

3.2 SIMPLE JOIN

- Note that it is necessary to qualify the client number, clientNo, in the SELECT list: clientNo could come from either table, and we have to indicate which one.

Query: List the names of all clients who have viewed a property, along with any comments supplied.

- To obtain the required rows, we include those rows from both tables that have identical values in the clientNo columns, using the search condition (c.clientNo = v.clientNo). We call these two columns the matching columns for the two tables.
- We compared the primary key in the Client table, c.clientNo, with the foreign key in the Viewing table, v.clientNo.

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

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

3.2 SIMPLE JOIN (CONTINUED)

- The SQL standard provides the following alternative ways to specify this join:

FROM Client c **JOIN** Viewing v **ON** c.clientNo = v.clientNo

FROM Client **JOIN** Viewing **USING** clientNo

FROM Client **NATURAL JOIN** Viewing

- In each case, the FROM clause replaces the original FROM and WHERE clauses.
- However, the first alternative produces a table with two identical clientNo columns; the remaining two produce a table with a single clientNo column.

3.3 SORTING A JOIN

Query: For each branch office, list the staff numbers and names of staff who manage properties and the properties that they manage.

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

- In this example, we need to join the Staff and PropertyForRent tables, based on the primary key/foreign key attribute (staffNo).
- To make the results more readable, we have ordered the output using the branch number as the major sort key and the staff number and property number as the minor keys.

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

3.4 THREE-TABLE JOIN

Query: For each branch, list the staff numbers and names of staff who manage properties, including the city in which the branch is located and the properties that the staff manage.

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

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

3.4 THREE-TABLE JOIN (CONTINUED)

- The result table requires columns from three tables: Branch (branchNo and city), Staff (staffNo, fName and lName), and PropertyForRent (propertyNo), so a join must be used.
- The Branch and Staff details are joined using the condition (b.branchNo = s.branchNo) to link each branch to the staff who work there.
- The Staff and PropertyForRent details are joined using the condition (s.staffNo = p.staffNo) to link staff to the properties they manage.
- Note again that the SQL standard provides alternative formulations for the FROM and WHERE clauses, for example:

```
FROM (Branch b JOIN Staff s USING branchNo) AS bs
JOIN PropertyForRent p USING staffNo
```

3.5 MULTIPLE GROUPING COLUMNS

Query: Find the number of properties handled by each staff member, along with the branch number of the member of staff.

```
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

- To list the required numbers, we first need to find out which staff actually manage properties.
- This can be found by joining the Staff and PropertyForRent tables on the staffNo column, using the FROM/WHERE clauses.
- Next, we need to form groups consisting of the branch number and staff number, using the GROUP BY clause.
- Finally, we sort the output using the ORDER BY clause.

3.6 COMPUTING A JOIN

Cartesian Product

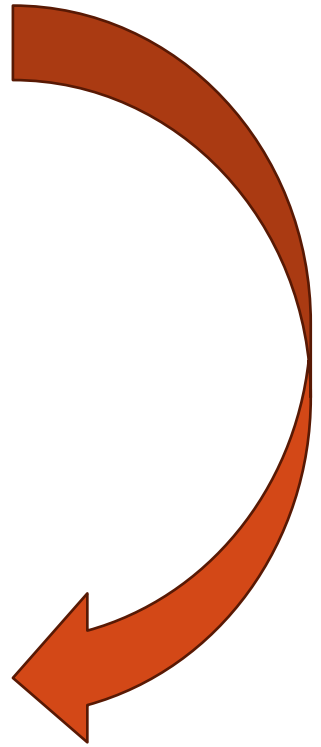
- A join is a subset of a more general combination of two tables known as the **Cartesian product**
- The Cartesian product of two tables is another table consisting of all possible pairs of rows from the two tables.
- The columns of the product table are all the columns of the first table followed by all the columns of the second table.
- **If we specify a two-table query without a WHERE clause, SQL produces the Cartesian product of the two tables as the query result.**
- In fact, the ISO standard provides a special form of the SELECT statement for the Cartesian product

SELECT [DISTINCT | ALL] {* | columnList}
FROM TableName1 **CROSS JOIN** TableName2

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00

OID	DATE	CUSTOMER_ID	AMOUNT
100	2009-10-08 00:00:00	3	1500.00
101	2009-11-20 00:00:00	2	1560.00

```
SELECT ID, NAME, AMOUNT, DATE
FROM CUSTOMERS
CROSS JOIN ORDERS;
```



ID	NAME	AMOUNT	DATE
2	Khilan	1500.00	2009-10-08 00:00:00
1	Ramesh	1560	2009-11-20 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
1	Ramesh	1500.00	2009-10-08 00:00:00

3.6 COMPUTING A JOIN (CONTINUED)

Cartesian Product

- Conceptually, the procedure for generating the results of a SELECT with a join is as follows:

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

SELECT [DISTINCT | ALL] {* | columnList}
FROM TableName1 **CROSS JOIN** TableName2

3.7 OUTER JOIN

- The join operation combines data from two tables by forming pairs of related rows where the matching columns in each table have the same value.
- **If one row of a table is unmatched, the row is omitted from the result table. This has been the case for the joins we examined earlier (Except Cross join).**
- **The Outer join retains rows that do not satisfy the join condition.**
- There are three types of Outer join: **Left**, **Right**, and **Full** Outer joins.
- The differences between them involve which unrelated data they keep – it can be from the first table, from the second, or from both of them.
- The cells without data to fill will have a value of NULL.

3.7 OUTER JOIN (CONTINUED)

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

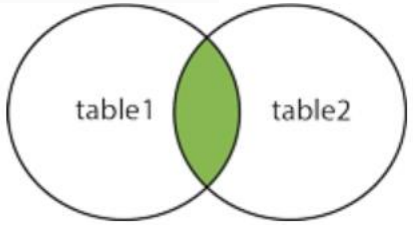
- **The (Inner) join of these two tables:**

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

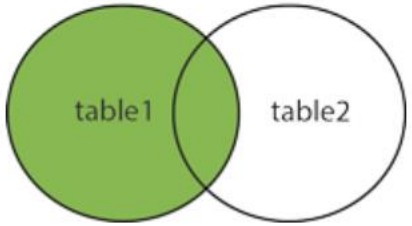
branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

- The **INNER JOIN** keyword selects records that have matching values in both tables.

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



3.7.1 LEFT (OUTER) JOIN



- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).
- **The result is 0 records from the right side, if there is no match.**

Query: List all branch offices and any properties that are in the same city.

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

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

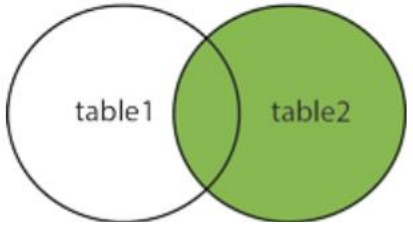
PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

- Left Outer join includes not only those rows that have the same city, but also those rows of the first (left) table that are unmatched with rows from the second (right) table.
- The columns from the second table are filled with NULLs.

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

3.7.2 RIGHT (OUTER) JOIN



- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1).
- The result is 0 records from the left side, if there is no match.

Query: List all properties and any branch offices that are in the same city.

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

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

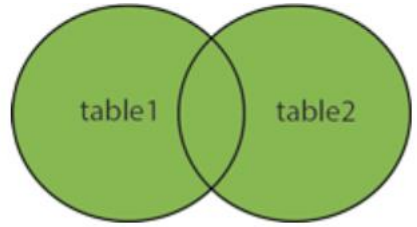
PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

- In this example, the Right Outer join includes not only those rows that have the same city, but also those rows of the second (right) table that are unmatched with rows from the first (left) table.
- The columns from the first table are filled with NULLs.

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

3.7.3 FULL (OUTER) JOIN



- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Query: List the branch offices and properties that are in the same city along with any unmatched branches or properties.

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

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

- In this case, the Full Outer join includes not only those rows that have the same city, but also those rows that are unmatched in both tables. The unmatched columns are filled with NULLs.

3.8 SELF JOIN

- A self join is a regular join, but the table is joined with itself.
- In the example below list the employees with their manager's name.

Id	FullName	Salary	ManagerId
1	John Smith	10000	3
2	Jane Anderson	12000	3
3	Tom Lanon	15000	4
4	Anne Connor	20000	
5	Jeremy York	9000	1

3.8 SELF JOIN

- A self join is a regular join, but the table is joined with
- In the example below list the employees with their ma

```
SELECT
      employee.Id,
      employee.FullName,
      employee.ManagerId,
      manager.FullName AS ManagerName
FROM Employees employee
JOIN Employees manager
ON employee.ManagerId = manager.Id
```

Id	FullName	Salary	ManagerId
1	John Smith	10000	3
2	Jane Anderson	12000	3
3	Tom Lanon	15000	4
4	Anne Connor	20000	
5	Jeremy York	9000	1

Id	FullName	ManagerId	ManagerName
1	John Smith	3	Tom Lanon
2	Jane Anderson	3	Tom Lanon
3	Tom Lanon	4	Anne Connor
5	Jeremy York	1	John Smith

4

EXISTS AND NOT EXISTS

4.1 EXISTS AND NOT EXISTS

- **The keywords EXISTS and NOT EXISTS are designed for use only with subqueries.**
- **They produce a simple true/false result.**
- EXISTS is true if and only if there exists at least one row in the result table returned by the subquery; it is false if the subquery returns an empty result table.
- NOT EXISTS is the opposite of EXISTS.
- Because EXISTS and NOT EXISTS check only for the existence or nonexistence of rows in the subquery result table, the subquery can contain any number of columns.

4.2 EXISTS EXAMPLE

Query: Find all staff who work in a London branch office. OR Find all staff such that there exists a Branch row containing his/her branch number, branchNo, and the branch city equal to London

```
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

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

- If it exists, the subquery evaluates to true.
- Note that the first part of the search condition s.branchNo = b.branchNo is necessary to ensure that we consider the correct branch row for each member of staff.

4.2 EXISTS EXAMPLE (CONTINUED)

- If we omitted this part of the query, we would get all staff rows listed out because the subquery (SELECT * FROM Branch WHERE city = 'London') would always be true and the query would be reduced to:

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

- which is equivalent to:

SELECT staffNo, fName, lName, position **FROM** Staff;

We could also have written this query using the join construct:

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

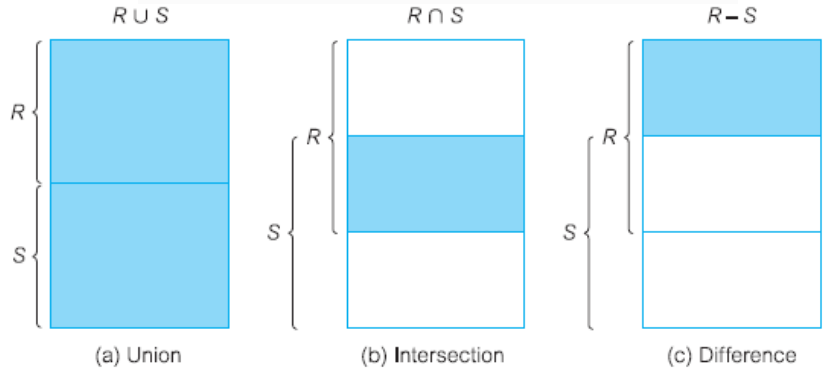
5

COMBINING RESULT TABLES

5.1 INTRODUCTION

- In SQL, we can use the normal set operations of Union, Intersection, and Difference to **combine the results** of two or more queries into a single result table:
  a) The **Union** of two tables, A and B, is a table containing all rows that are in either the first table A or the second table B or both.
  b) The **Intersection** of two tables, A and B, is a table containing all rows that are common to both tables A and B.
  c) The **Difference** of two tables, A and B, is a table containing all rows that are in table A but are not in table B.

- The most important one being that the two tables have to be **union-compatible**; that is, they have the same structure.
- This implies that the two tables must contain the same number of columns, and that their corresponding columns have the same data types and lengths.



5.1 INTRODUCTION (CONTINUED)

- The three set operators in the ISO standard are called UNION, INTERSECT, and EXCEPT.
- The format of the set operator clause in each case is:

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

- If **CORRESPONDING BY** is specified, then the set operation is performed on the named column(s);
- If **CORRESPONDING** is specified but not the **BY** clause, the set operation is performed on the columns that are common to both tables.
- If **ALL** is specified, the result can include duplicate rows.
- Some dialects of SQL do not support INTERSECT and EXCEPT; others use MINUS in place of EXCEPT.

5.2 UNION

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Query: Construct a list of 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);
```

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

- This query is executed by producing a result table from the first query and a result table from the second query, and then merging both tables into a single result table consisting of **all the rows from both result tables with the duplicate rows removed.**

5.3.1 UNION (UNION ALL)

- **UNION**: This removes duplicate rows without using DISTINCT in the SELECT statements.
- **UNION ALL**: This does not remove duplicate rows, and they will remain in the final result. This will perform faster than UNION because it doesn't have to remove duplicates.
- Consider the below example

Employee Table (tblemp)

Emp_id	Emp_name	street	city	Emp_contact	Salary	Dept_id
101	jone	althan	Surat	1111111	20000	10001
102	cartin	udhna	Surat	2222222	15000	20001
103	krish	ajava	Vadodara	3333333	30000	20001
104	dhiru	ramnagar	Vadodara	8888888	36000	30001
105	om	althan	Surat	7777777	22000	30001
106	adi	vesu	Navsari	2323232	35000	10001
107	annant	shivnagar	Navsari	5555555	34000	10002
108	yogi	althan	Surat	8989898	25000	10002
109	muskan	vesu	Vadodara	9999999	18000	10001
110	rudra	kashi	hazira	1212121	31000	20001

Department Table (tbldept)

Dept_id	Dept_name	Dept_location
10001	Account	Surat
20001	Sales	Hazira
30001	Finance	Vadodara
10002	Marketing	Surat

5.3.1 UNION (UNION ALL) (CONTINUED)

```
SELECT city
FROM tbldept
UNION
SELECT city
FROM tblemp
```

city
Hazira
Navsari
Surat
Vadodara

```
SELECT city
FROM tbldept
UNION ALL
SELECT city
FROM tblemp
```

city
Surat
Vadodara
Surat
Hazira
Surat
Surat
Vadodara
Vadodara
Surat
Navsari
Navsari
Surat
Vadodara
Hazira

5.4 INTERSECT

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Query: Construct a list of all cities where there is both a branch office and a property.

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

or

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

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

- This query is executed by producing a result table from the first query and a result table from the second query, and then creating a single result table consisting of those rows that are common to both result tables.

city
Aberdeen
Glasgow
London

- We could rewrite this query *without the INTERSECT* operator, for example:

```
SELECT DISTINCT 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 b.city = p.city);
```

5.5 EXCEPT

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Query: Construct a list of all cities where there is a branch office but no properties.

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

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

or

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

- This query is executed by producing a result table from the first query and a result table from the second query, and then creating a single result table consisting of those rows that appear in the first result table but not in the second one.

- We could rewrite this query *without the EXCEPT* operator, for example:

```
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 b.city = p.city);
```

city
Bristol

WRAP UP
THANK YOU!