CSE 204 - INTRO TO DATABASE SYSTEMS
SQL

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OUTLINE

- Purpose and importance of SQL.
- How to retrieve data from database using SELECT and:
  - Use compound WHERE conditions.
  - Sort query results using ORDER BY.
  - Use aggregate functions.
  - Group data using GROUP BY and HAVING.
  - Use subqueries.
  - Join tables together.
  - Perform set operations (UNION, INTERSECT, EXCEPT).
- How to update database using INSERT, UPDATE, and DELETE.

OBJECTIVES OF SQL

- Ideally, database language should allow user to:
  - create the database and relation structures;
  - perform insertion, modification, deletion of data from relations;
  - perform simple and complex queries.
- Must perform these tasks with minimal user effort and command structure/syntax must be easy to learn.
- It must be portable.

OBJECTIVES OF SQL

- SQL is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.
- Until SQL:1999, SQL did not contain flow of control commands. These had to be implemented using a programming or job-control language, or interactively by the decisions of user.

OBJECTIVES OF SQL

- SQL is relatively easy to learn:
  - it is non-procedural - you specify what information you require, rather than how to get it;
  - it is essentially free-format.
  - Consists of standard English words:

```
1.    CREATE TABLE Staff(staffNo VARCHAR(5),
lName VARCHAR(15),
salary DECIMAL(7,2));
2.    INSERT INTO Staff VALUES (‘SG16’, ‘Brown’, 8300);
3.    SELECT staffNo, lName, salary
   FROM Staff
   WHERE salary > 10000;
```

OBJECTIVES OF SQL

- Can be used by range of users including DBAs, management, application developers, and other types of end users.
- An ISO standard now exists for SQL, making it both the formal and de facto standard language for relational databases.

HISTORY OF SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called ‘Structured English Query Language’ (SEQUEL).
- A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.
- Still pronounced ‘see-quel’, though official pronunciation is ‘S-Q-L’.
- IBM subsequently produced a prototype DBMS called System R, based on SEQUEL/2.
- Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.

HISTORY OF SQL

- In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL.
- In 1987, ANSI and ISO published an initial standard for SQL.
- In 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.
- In 1999, SQL:1999 was released with support for object-oriented data management.
- In late 2003, SQL:2003 was released.
- In summer 2008, SQL:2008 was released.
- In late 2011, SQL:2011 was released.

IMPORTANCE OF SQL

- SQL has become part of application architectures such as IBM’s Systems Application Architecture.
- It is strategic choice of many large and influential organizations (e.g. X/OPEN).
- SQL is Federal Information Processing Standard (FIPS) to which conformance is required for all sales of databases to American Government.
- SQL is used in other standards and even influences development of other standards as a definitional tool. Examples include:
  - ISO’s Information Resource Directory System (IRDS) Standard
  - Remote Data Access (RDA) Standard.

WRITING SQL COMMANDS

- SQL statement consists of reserved words and user-defined words.

❑ Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.

❑ User-defined words are made up by user and represent names of various database objects such as relations, columns, views.

WRITING SQL COMMANDS

- Most components of an SQL statement are case insensitive, except for literal character data.
- More readable with indentation and lineation:
  - Each clause should begin on a new line.
  - Start of a clause should line up with start of other clauses.
  - If clause has several parts, should each appear on a separate line and be indented under start of clause.

WRITING SQL COMMANDS

- Use extended form of BNF notation:
  - ❑ Upper-case letters represent reserved words.
  - ❑ Lower-case letters represent user-defined words.
  - ❑ | indicates a choice among alternatives.
  - ❑ Curly braces indicate a required element.
  - ❑ Square brackets indicate an optional element.
  - ❑ … indicates optional repetition (0 or more).

LITERALS

- Literals are constants used in SQL statements.
- All non-numeric literals must be enclosed in single quotes (e.g. ‘London’).
- All numeric literals must not be enclosed in quotes (e.g. 650.00).

SELECT STATEMENT

```
SELECT [DISTINCT | ALL]
     {* | [columnExpression [AS newName]] [,...] }
FROM       TableName [alias] [, ...]
[WHERE     condition]
[GROUP BY columnList]  [HAVING     condition]
[ORDER BY columnList]
```

SELECT STATEMENT

- SELECT Specifies which columns are to appear in output.
- FROM Specifies table(s) to be used.
- WHERE Filters rows.
- GROUP BY Forms groups of rows with same column value.
- HAVING Filters groups subject to some condition.
- ORDER BY Specifies the order of the output.
- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.

EXAMPLE – ALL COLUMNS, ALL ROWS

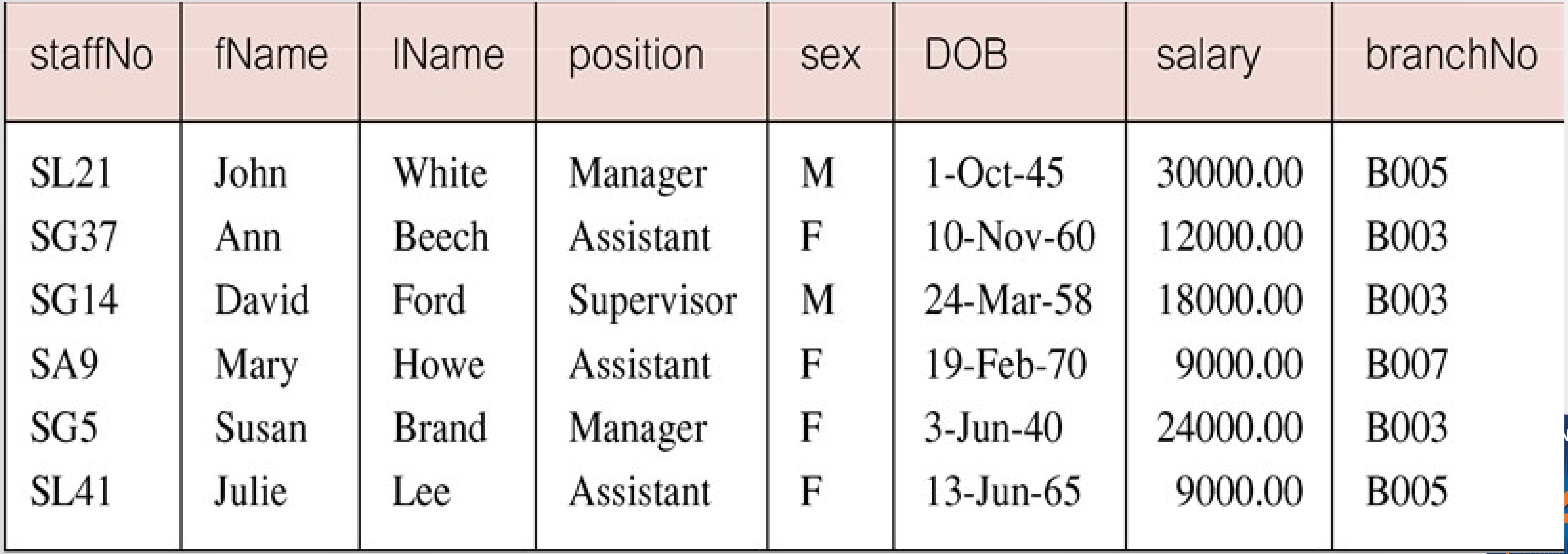
- List full details of all staff.

```
SELECT staffNo, fName, lName, address,
            position, sex, DOB, salary, branchNo
      FROM Staff;
```

- Can use * as an abbreviation for 'all columns':

```
      SELECT *
      FROM Staff;
```

EXAMPLE – ALL COLUMNS, ALL ROWS



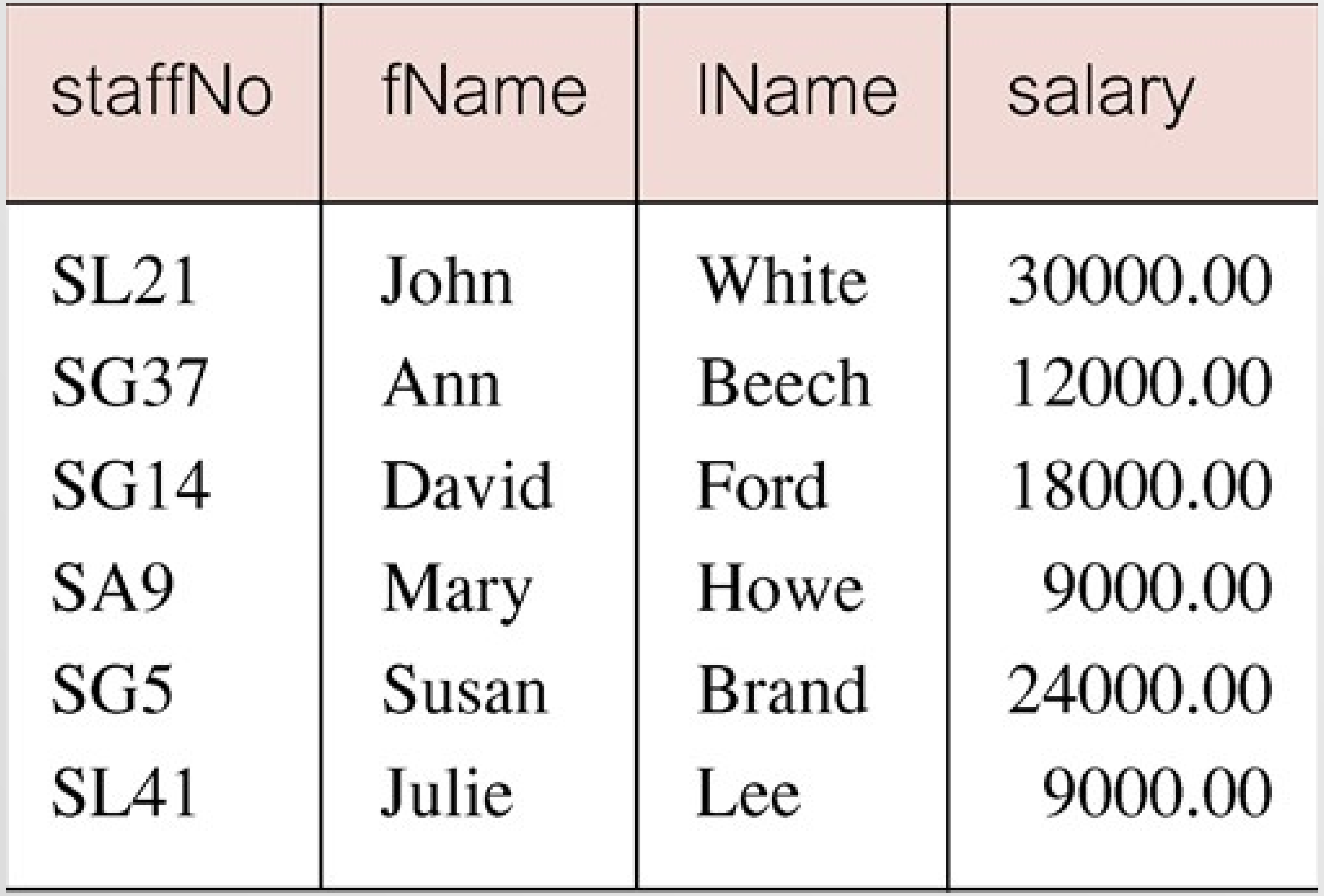
staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

EXAMPLE 2 – SPECIFIC COLUMNS, ALL ROWS

- Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

```
SELECT staffNo, fName, lName, salary

FROM Staff;
```

EXAMPLE 2 – SPECIFIC COLUMNS, ALL ROWS

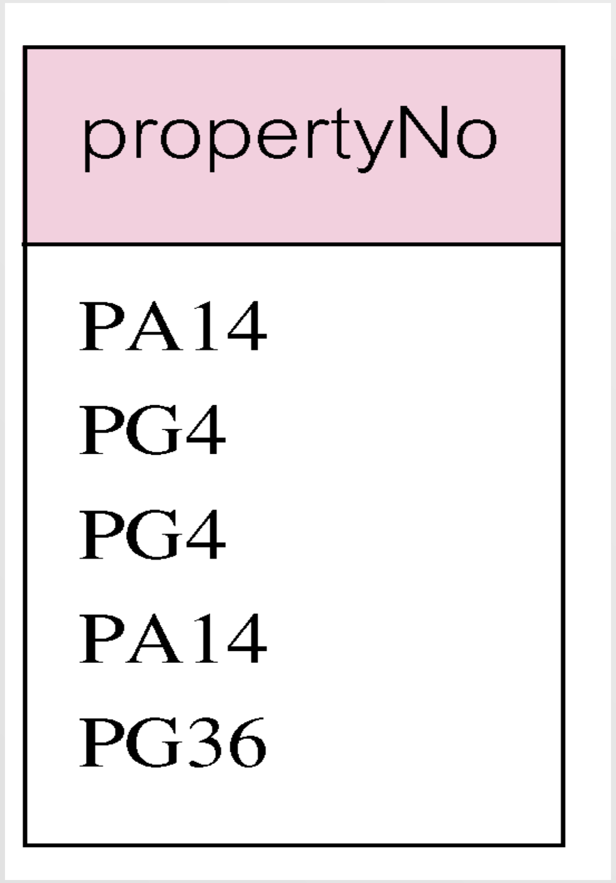


staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

EXAMPLE 3 – USE OF DISTINCT

- List the property numbers of all properties that have been viewed.

```
SELECT propertyNo

FROM Viewing;
```

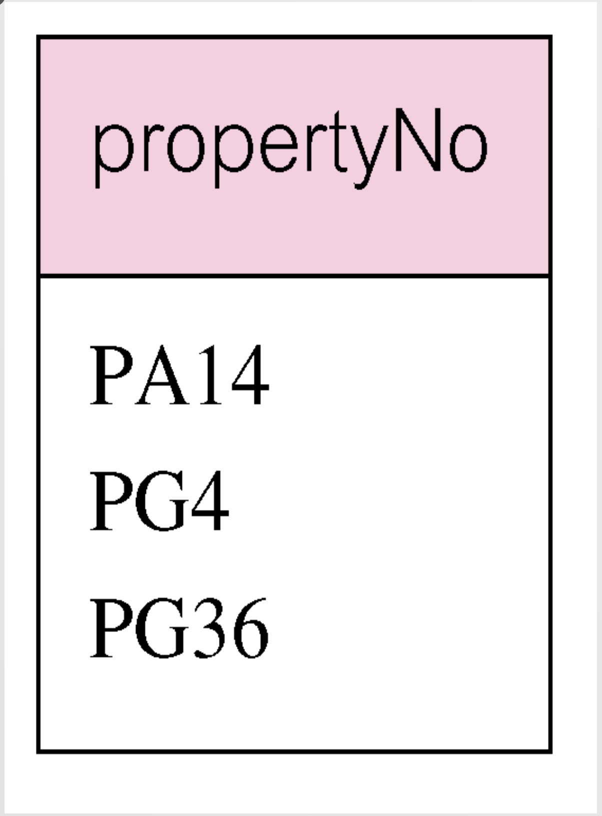


propertyNo
PA14
PG4
PG4
PA14
PG36

EXAMPLE 3 – USE OF DISTINCT

- Use DISTINCT to eliminate duplicates:

```
SELECT DISTINCT propertyNo
FROM Viewing;
```

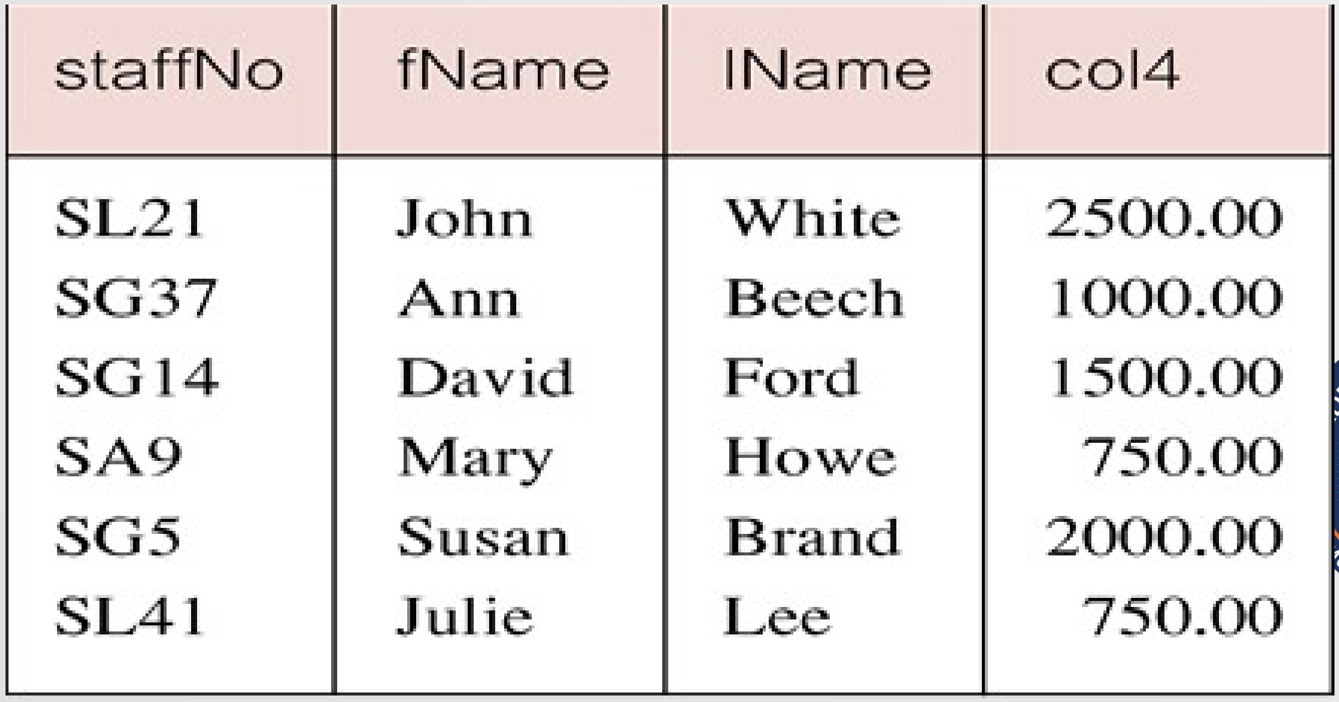


propertyNo
PA14
PG4
PG36

EXAMPLE 4 – CALCULATED FIELDS

- Produce list of monthly salaries for all staff, showing staff number, first/last name, and salary.

```
SELECT staffNo, fName, lName, salary/12

FROM Staff;
```



staffNo	fName	lName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

EXAMPLE 4 – CALCULATED FIELDS

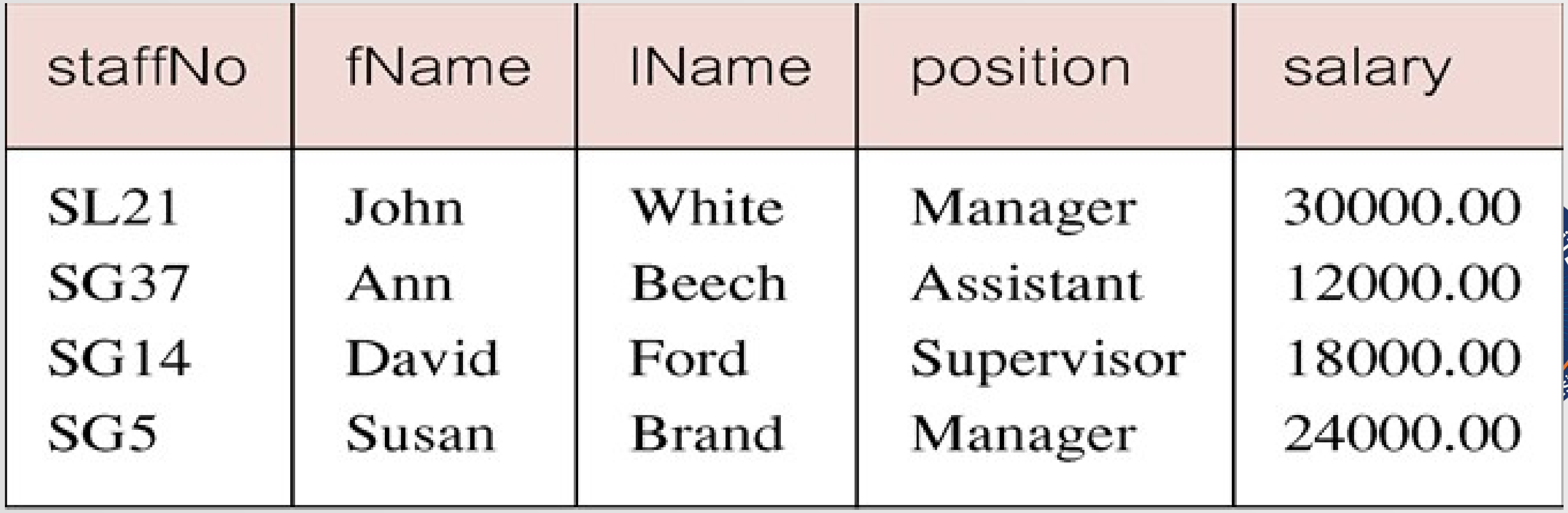
- To name column, use AS clause:

```
     SELECT staffNo, fName, lName, salary/12 AS 
monthlySalary

     FROM Staff;
```

EXAMPLE 5 – COMPARISON SEARCH

- List all staff with a salary greater than 10,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000;
```

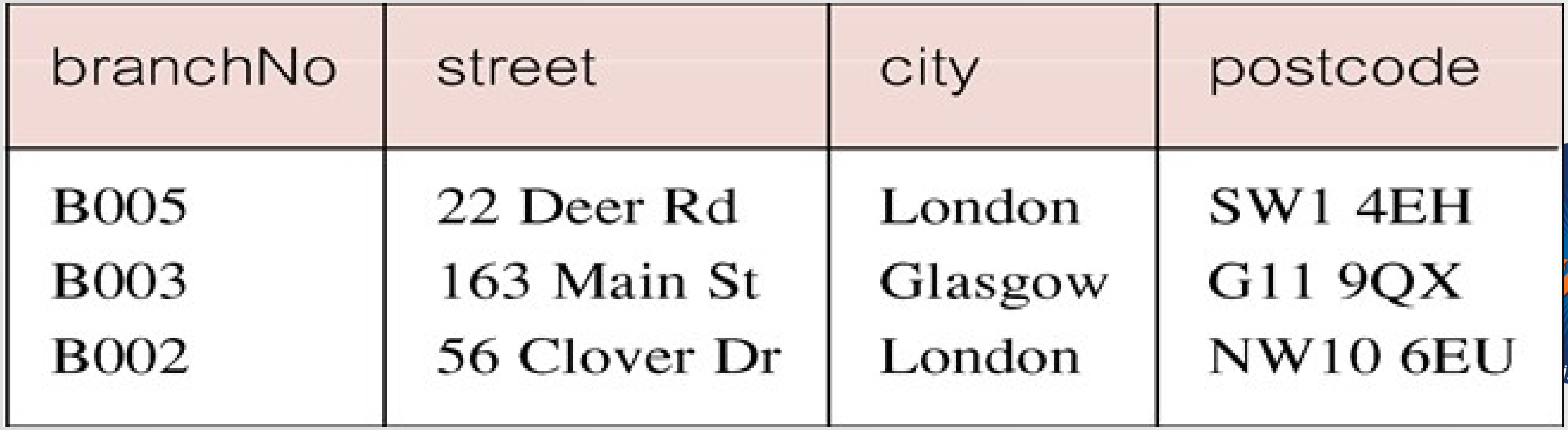


staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

EXAMPLE 6 – COMPOUND COMPARISON SEARCH

- List addresses of all branch offices in London or Glasgow.

```
SELECT *
FROM Branch
WHERE city = ‘London’ OR city = ‘Glasgow’;
```



branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

EXAMPLE 7 – RANGE SEARCH

- List all staff with a salary between 20,000 and 30,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary BETWEEN 20000 AND 30000;
```

- BETWEEN test includes the endpoints of range.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

EXAMPLE 7 – RANGE SEARCH

- Also a negated version NOT BETWEEN.
- BETWEEN does not add much to SQL's expressive power. Could also write:

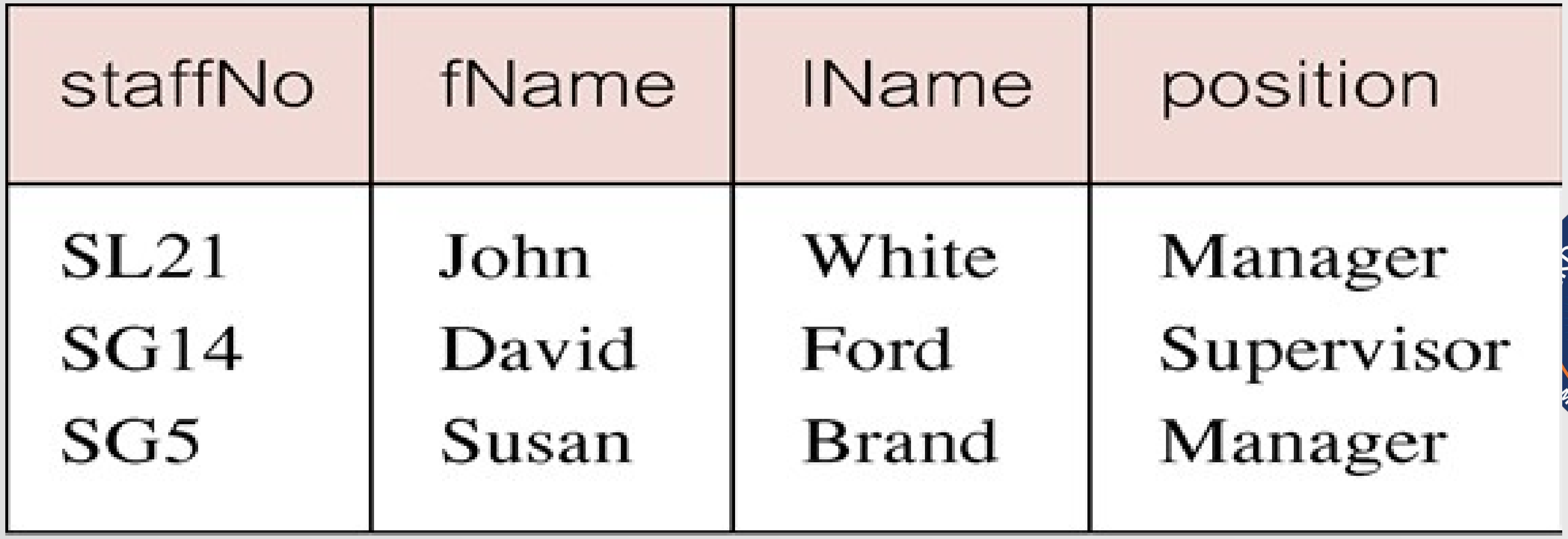
```
SELECT staffNo, fName, lName, position, salary

FROM Staff

WHERE salary>=20000 AND salary <= 30000;
```

- Useful, though, for a range of values.

EXAMPLE 8 – SET MEMBERSHIP

- List all managers and supervisors.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN (‘Manager’, ‘Supervisor’);
```



staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

EXAMPLE 8 – SET MEMBERSHIP

- There is a negated version (NOT IN).
- IN does not add much to SQL’s expressive power. Could have expressed this as:

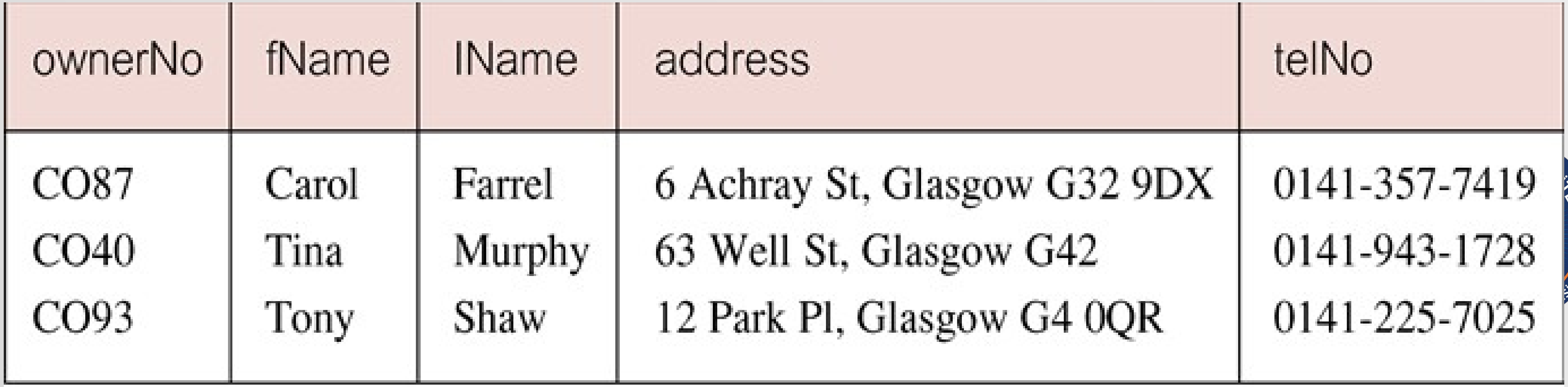
```
SELECT staffNo, fName, lName, position
    FROM Staff
WHERE position=‘Manager’ OR
        position=‘Supervisor’;
```

- IN is more efficient when set contains many values.

EXAMPLE 9 – PATTERN MATCHING

- Find all owners with the string ‘Glasgow’ in their address.

```
SELECT ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE ‘%Glasgow%’;
```



ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

EXAMPLE 9 – PATTERN MATCHING

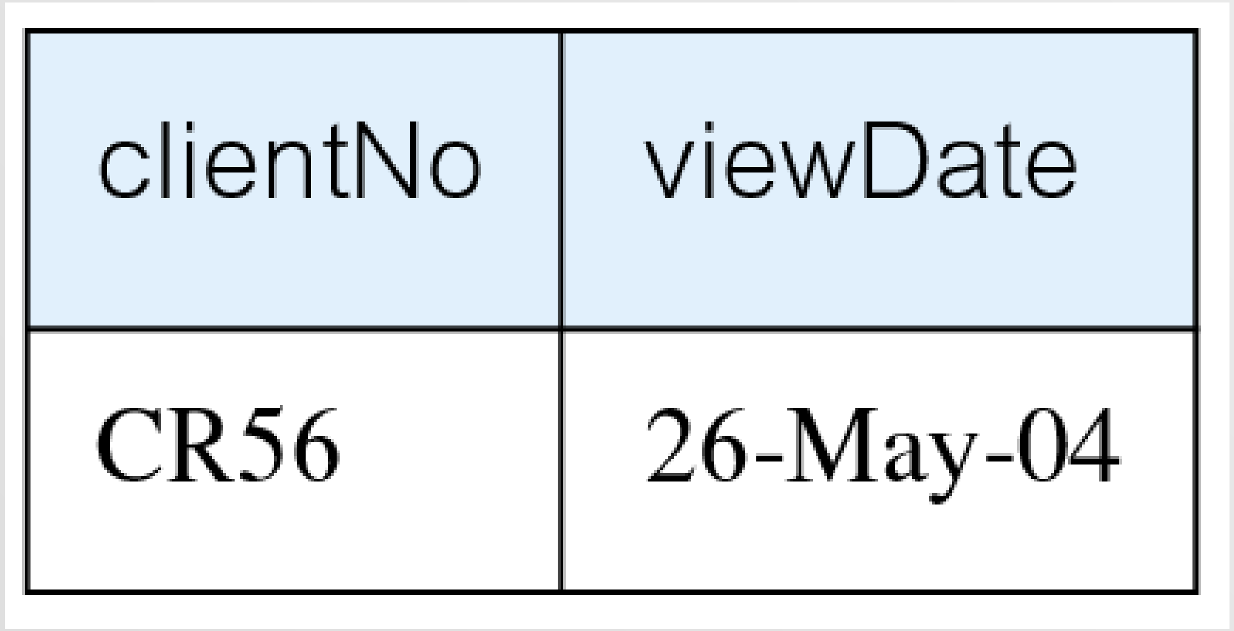
- SQL has two special pattern matching symbols:
  - %: sequence of zero or more characters;
  - _ (underscore): any single character.
- LIKE ‘%Glasgow%’ means a sequence of characters of any length containing ‘Glasgow’.

EXAMPLE 10 – NULL SEARCH CONDITION

- List details of all viewings on property PG4 where a comment has not been supplied.
- There are 2 viewings for property PG4, one with and one without a comment.
- Have to test for null explicitly using special keyword IS NULL:

```
SELECT clientNo, viewDate

FROM Viewing

WHERE propertyNo = ‘PG4’ AND

                     comment IS NULL;
```

EXAMPLE 10 – NULL SEARCH CONDITION



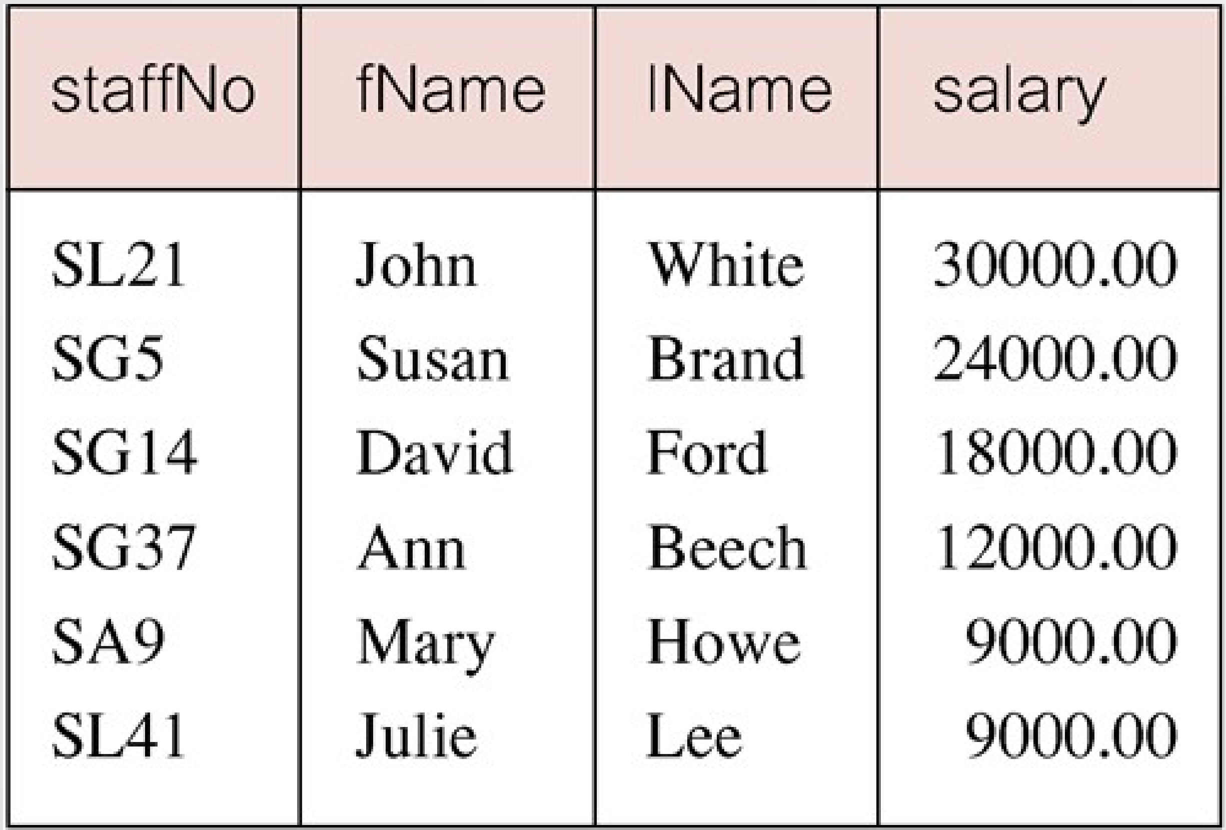
clientNo	viewDate
CR56	26-May-04

- Negated version (IS NOT NULL) can test for non-null values.

EXAMPLE 11 – ORDERING (SINGLE COLUMN)

- List salaries for all staff, arranged in descending order of salary.

```
SELECT staffNo, fName, lName, salary

FROM Staff

ORDER BY salary DESC;
```

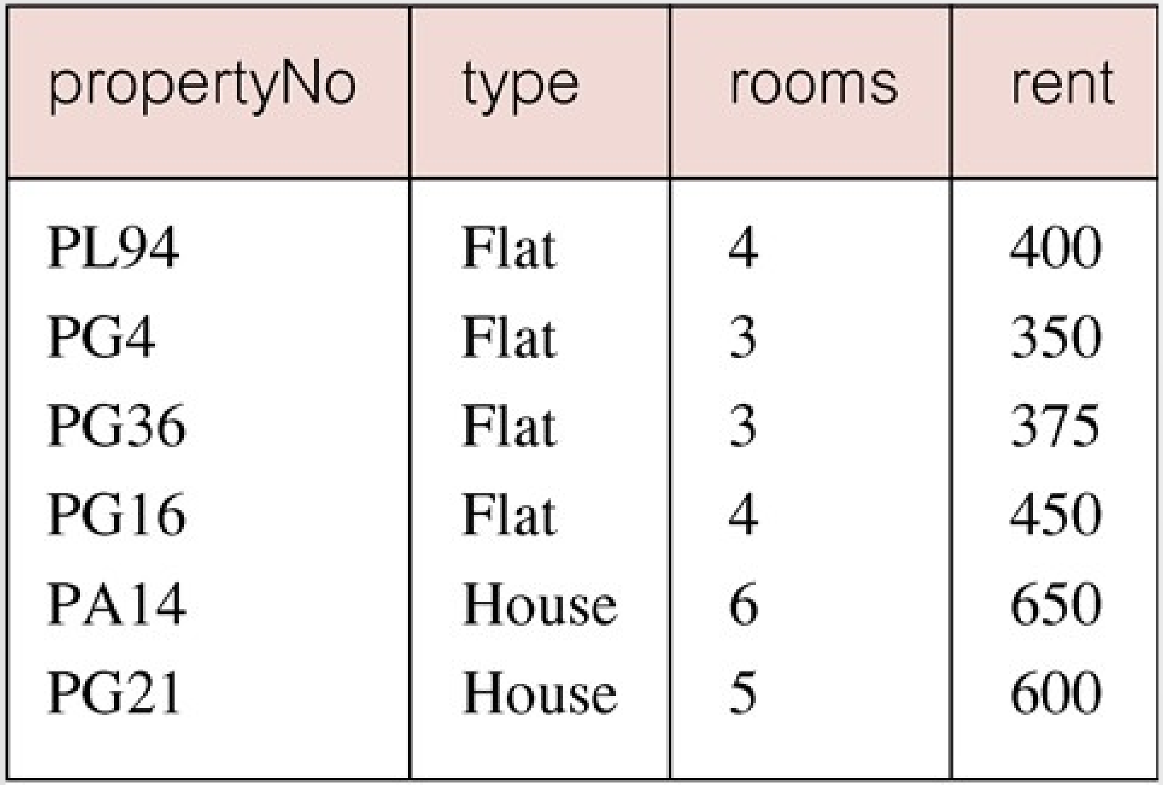


staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

EXAMPLE 12 – ORDERING (MULTIPLE COLUMN)

- Produce abbreviated list of properties in order of property type.

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type;
```

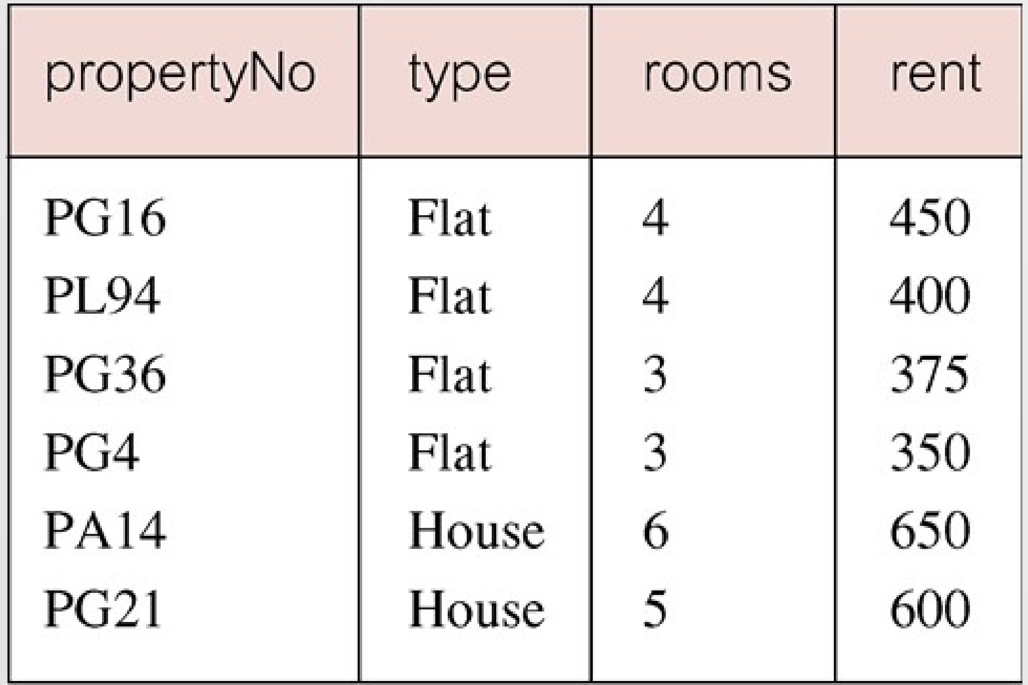


propertyNo	type	rooms	rent
PL94	Flat	4	400
PG4	Flat	3	350
PG36	Flat	3	375
PG16	Flat	4	450
PA14	House	6	650
PG21	House	5	600

EXAMPLE 12 – ORDERING (MULTIPLE COLUMN)

- Four flats in this list - as no minor sort key specified, system arranges these rows in any order it chooses.
- To arrange in order of rent, specify minor order:

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type, rent DESC;
```



propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

SELECT STATEMENT AGGREGATES

- ISO standard defines five aggregate functions:
  - COUNT returns number of values in specified column.
  - SUM returns sum of values in specified column.
  - AVG returns average of values in specified column.
  - MIN returns smallest value in specified column.
  - MAX returns largest value in specified column.

SELECT STATEMENT AGGREGATES

- Each operates on a single column of a table and returns a single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.
- COUNT(*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use DISTINCT before column name to eliminate duplicates.
- DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.

SELECT STATEMENT AGGREGATES

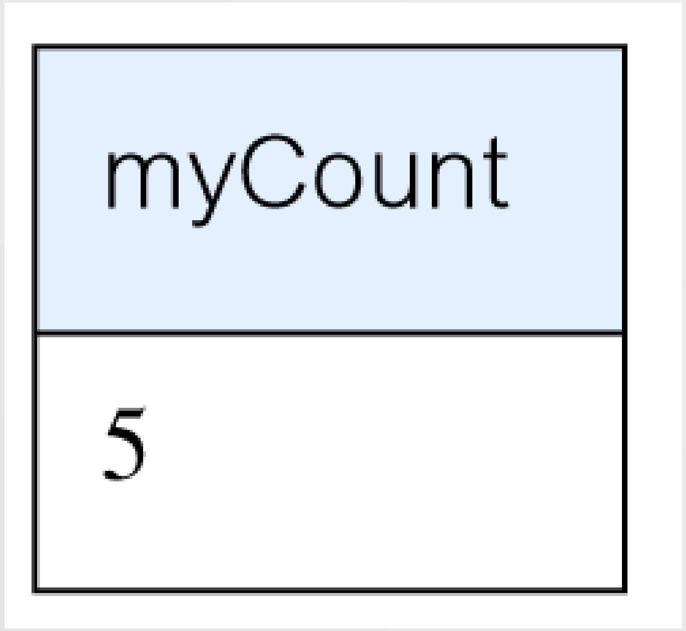
- Aggregate functions can be used only in SELECT list and in HAVING clause.
- If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference a column out with an aggregate function. For example, the following is illegal:

```
SELECT staffNo, COUNT(salary)

FROM Staff;
```

EXAMPLE 13 – USE OF COUNT(*)

- How many properties cost more than 350 per month to rent?

```
SELECT COUNT(*) AS myCount
        FROM PropertyForRent
        WHERE rent > 350;
```

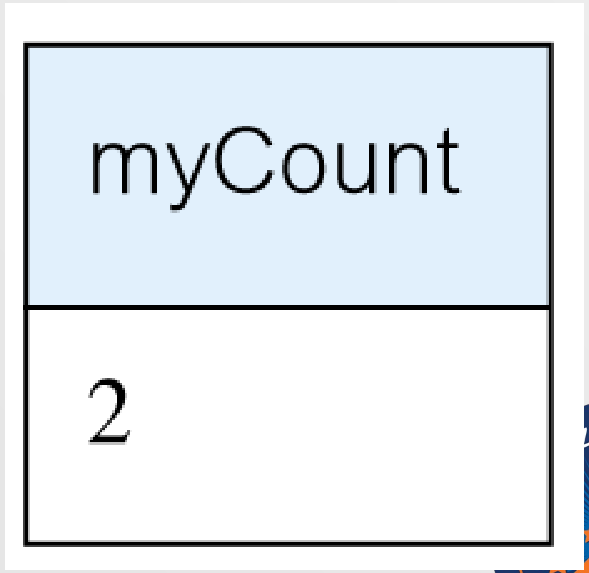


myCount
5

EXAMPLE 14 – USE OF COUNT(DISTINCT)

- How many different properties viewed in May '13?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN '1-May-13'
          AND '31-May-13';
```

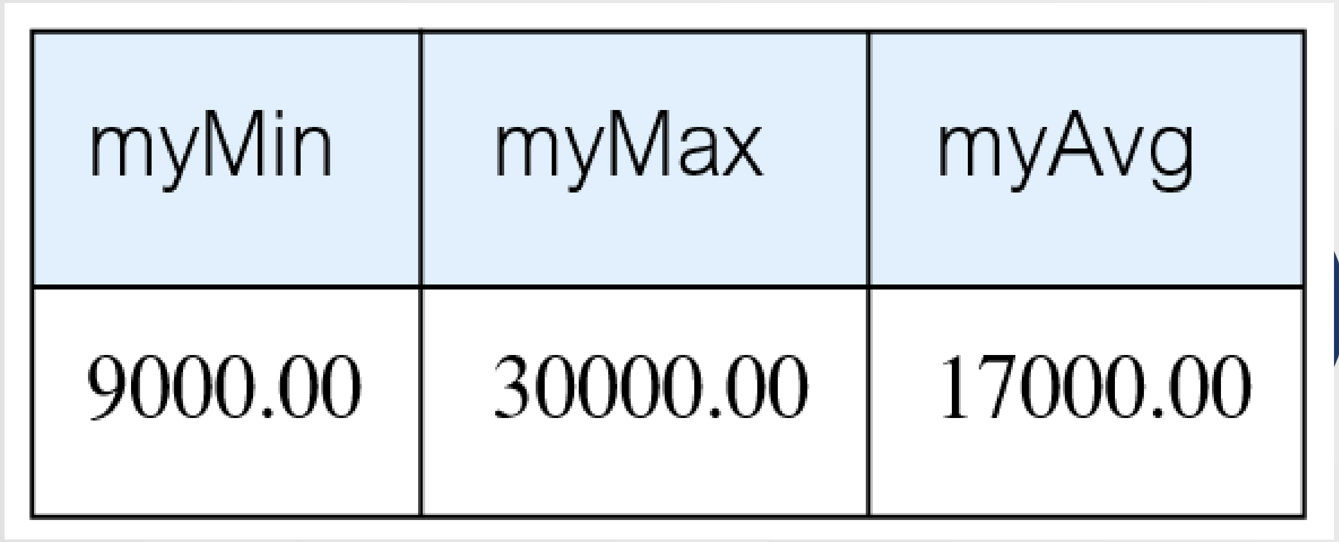


myCount
2

EXAMPLE 16 – USE OF MIN, MAX, AVG

- Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS myMin,
       MAX(salary) AS myMax,
       AVG(salary) AS myAvg
FROM Staff;
```



myMin	myMax	myAvg
9000.00	30000.00	17000.00

SELECT STATEMENT GROUPING

- Use GROUP BY clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be single-valued per group, and SELECT clause may only contain:
  - column names
  - aggregate functions
  - constants
  - expression involving combinations of the above.

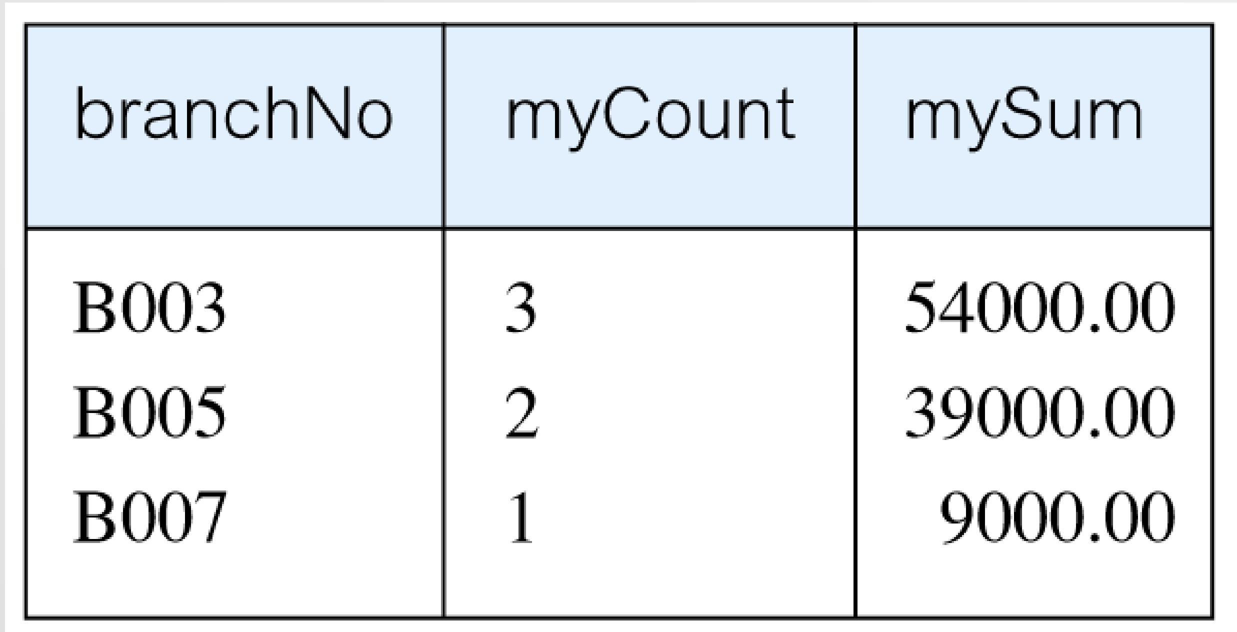
SELECT STATEMENT GROUPING

- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.

EXAMPLE 17 – USE OF GROUP BY

- Find number of staff in each branch and their total salaries.

```
SELECT branchNo,
                COUNT(staffNo) AS myCount,
       SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```



branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

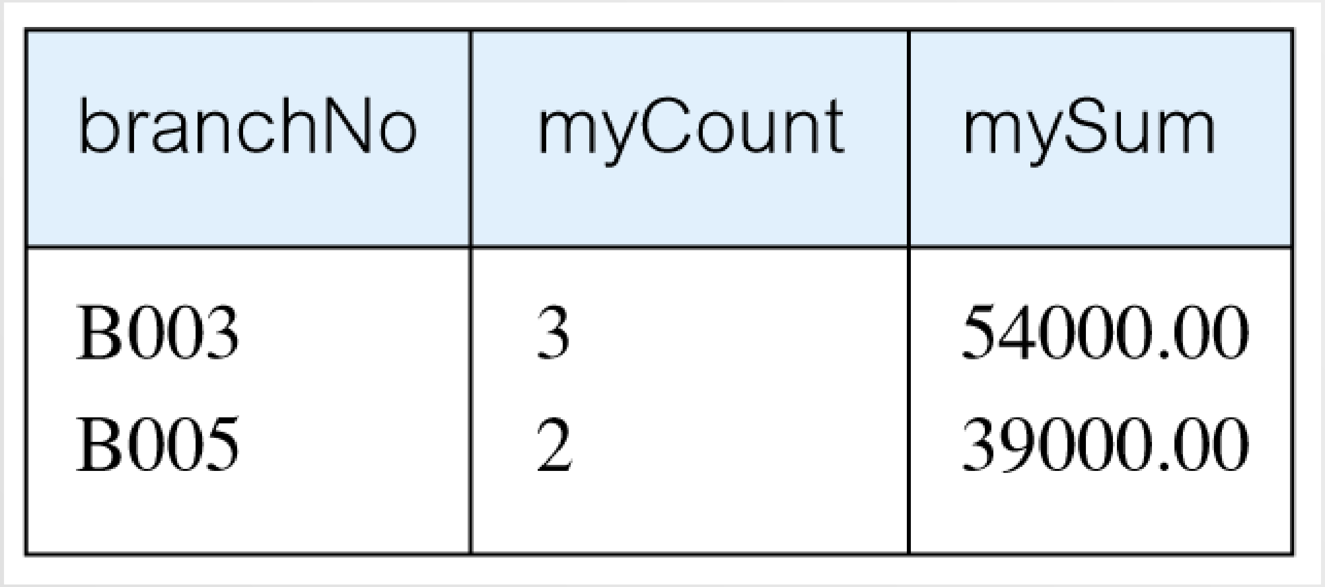
RESTRICTED GROUPINGS – HAVING CLAUSE

- HAVING clause is designed for use with GROUP BY to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.

EXAMPLE 18 – USE OF HAVING

- For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

```
SELECT branchNo,
      COUNT(staffNo) AS myCount,
 SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```



branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

SUBQUERIES

- Some SQL statements can have a SELECT embedded within them.
- A subselect can be used in WHERE and HAVING clauses of an outer SELECT, where it is called a subquery or nested query.
- Subselects may also appear in INSERT, UPDATE, and DELETE statements.

EXAMPLE 19 – SUBQUERY WITH EQUALITY

- List staff who work in branch at '163 Main St'.

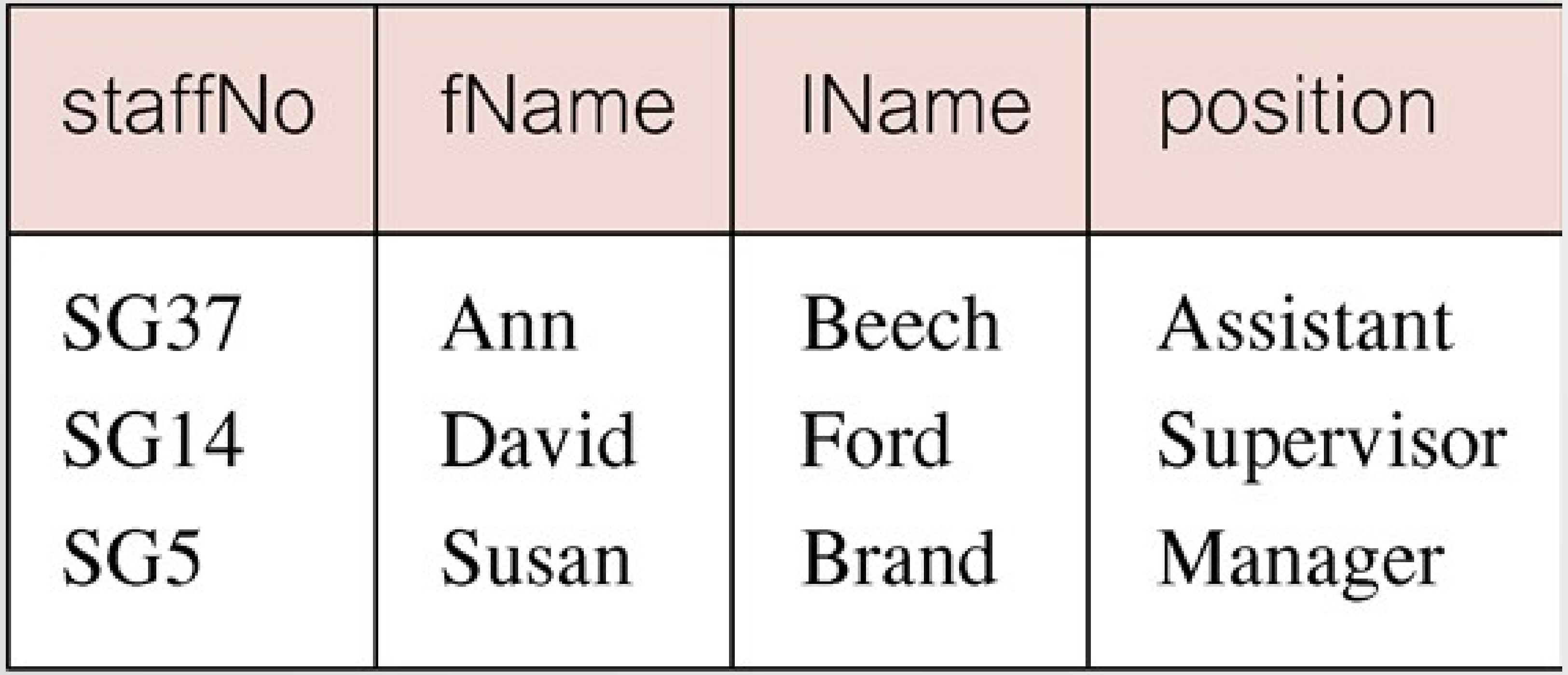
```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo =
          (SELECT branchNo
           FROM Branch
           WHERE street = '163 Main St');
```

EXAMPLE 19 – SUBQUERY WITH EQUALITY

- Inner SELECT finds branch number for branch at ‘163 Main St’ (‘B003’).
- Outer SELECT then retrieves details of all staff who work at this branch.
- Outer SELECT then becomes:

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

EXAMPLE 19 – SUBQUERY WITH EQUALITY



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

EXAMPLE 20 – SUBQUERY WITH AGGREGATE

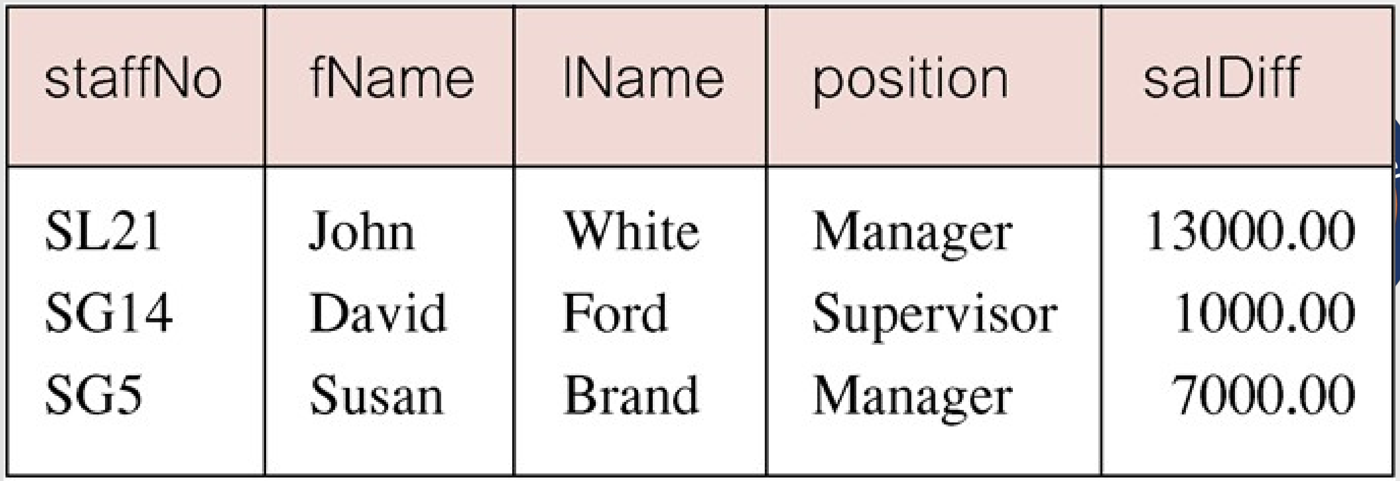
- List all staff whose salary is greater than the average salary, and show by how much.

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

EXAMPLE 20 – SUBQUERY WITH AGGREGATE

- Cannot write ‘WHERE salary > AVG(salary)’
- Instead, use subquery to find average salary (17000), and then use outer SELECT to find those staff with salary greater than this:

```
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

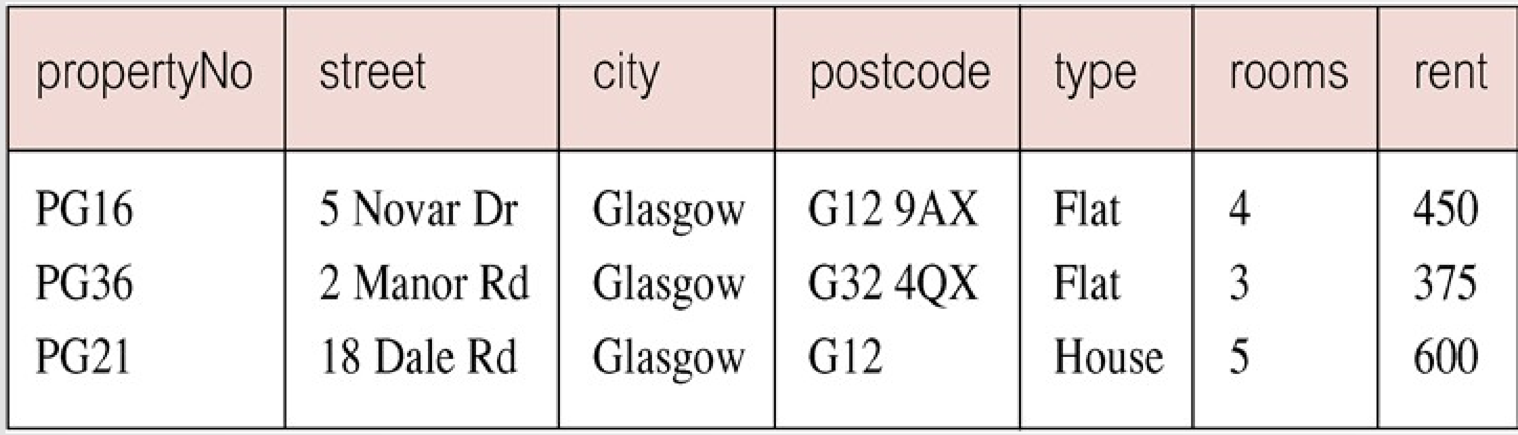
SUBQUERY RULES

- ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).
- Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.
- By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an alias.
- When subquery is an operand in a comparison, subquery must appear on right-hand side.
- A subquery may not be used as an operand in an expression.

EXAMPLE 21 – NESTED SUBQUERY: USE OF IN

- List properties handled by staff 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

ANY AND ALL

- ANY and ALL may be used with subqueries that produce a single column of numbers.
- With ALL, condition will only be true if it is satisfied by all values produced by subquery.
- With ANY, condition will be true if it is satisfied by any values produced by subquery.
- If subquery is empty, ALL returns true, ANY returns false.
- SOME may be used in place of ANY.

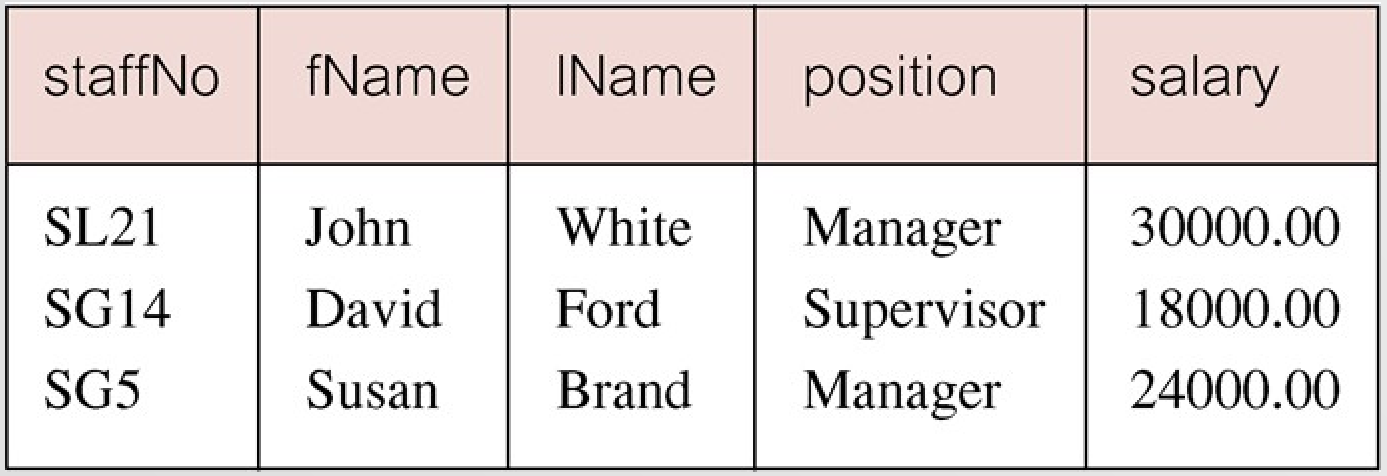
EXAMPLE 22 – USE OF ANY/SOME

- Find staff whose salary is larger than 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’);
```

EXAMPLE 22 – USE OF ANY/SOME

- Inner query produces set {12000, 18000, 24000} and outer query selects those staff whose salaries are greater than any of the values in this set.

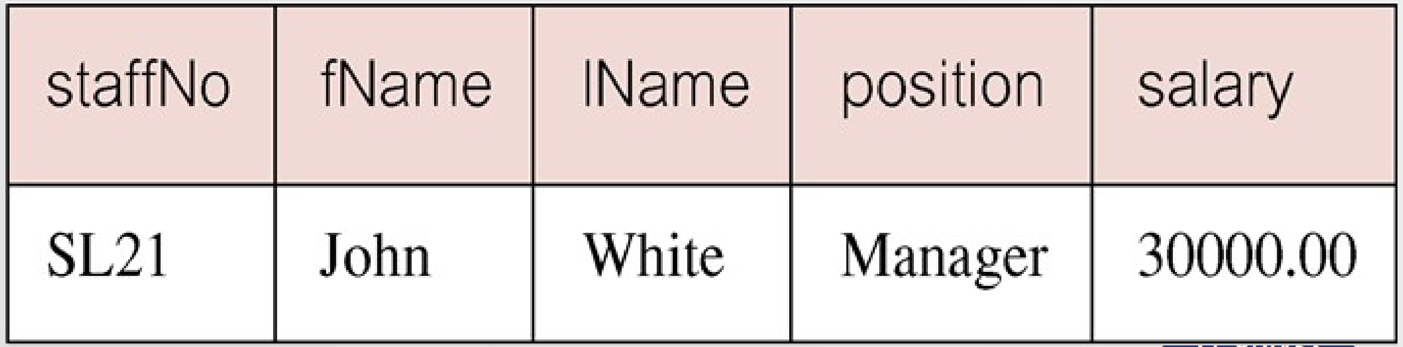


staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

EXAMPLE 23 – USE OF ALL

- Find staff whose salary is larger than 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

MULTI-TABLE QUERIES

- Can use subqueries provided result columns come from same table.
- If result columns come from more than one table must use a join.
- To perform join, include more than one table in FROM clause.
- Use comma as separator and typically include WHERE clause to specify join column(s).

MULTI-TABLE QUERIES

- 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.

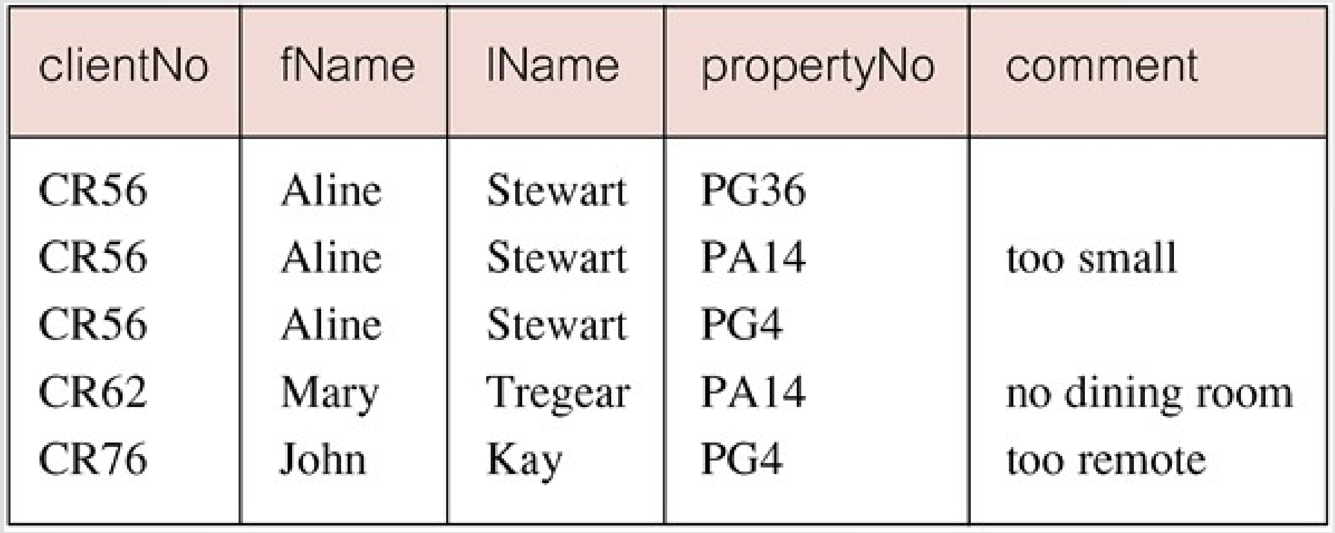
EXAMPLE 24 – SIMPLE JOIN

- 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;
```

EXAMPLE 24 – SIMPLE JOIN

- 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

ALTERNATIVE JOIN CONSTRUCTS

- SQL provides alternative ways to specify joins:

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

FROM Client JOIN Viewing USING clientNo

FROM Client NATURAL JOIN Viewing
```

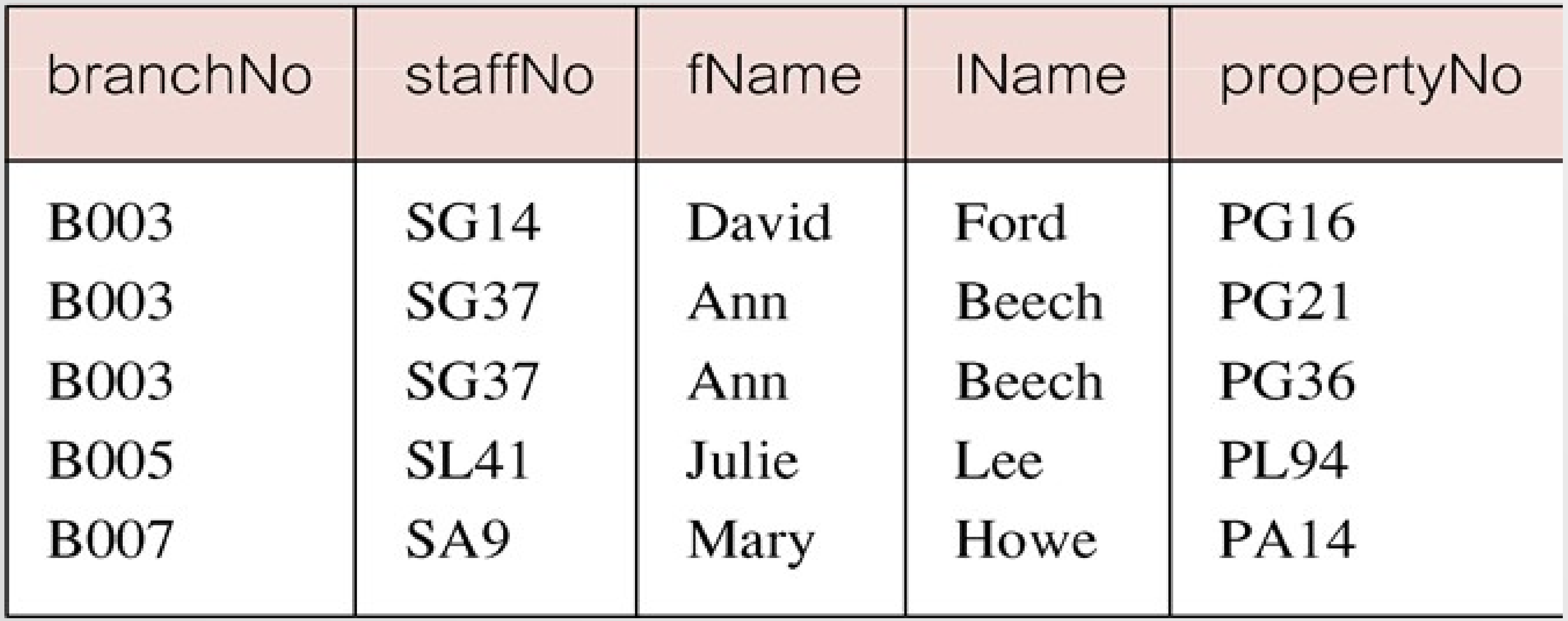
- In each case, FROM replaces original FROM and WHERE. However, first produces table with two identical clientNo columns.

EXAMPLE 25 – SORTING A JOIN

- For each branch, list numbers and names of staff who manage properties, and properties 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;
```

EXAMPLE 25 – SORTING A JOIN



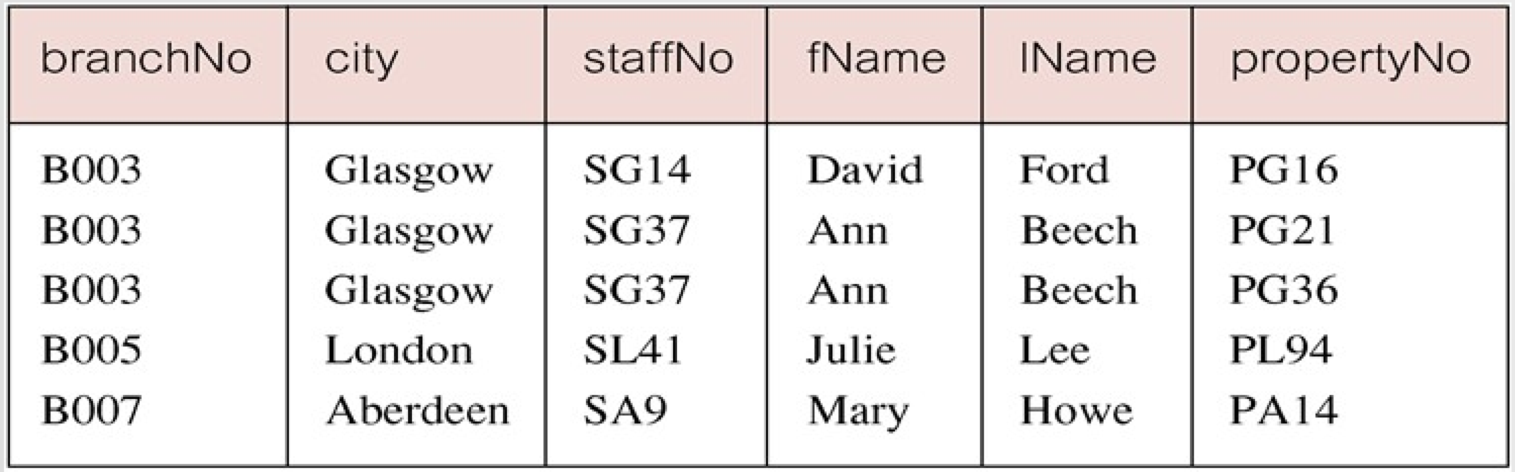
branchNo	staffNo	fName	lName	propertyNo
B003	SG14	David	Ford	PG16
B003	SG37	Ann	Beech	PG21
B003	SG37	Ann	Beech	PG36
B005	SL41	Julie	Lee	PL94
B007	SA9	Mary	Howe	PA14

EXAMPLE 26 – THREE TABLE JOIN

- For each branch, list staff who manage properties, including city in which branch is located and properties they 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;
```

EXAMPLE 26 – 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

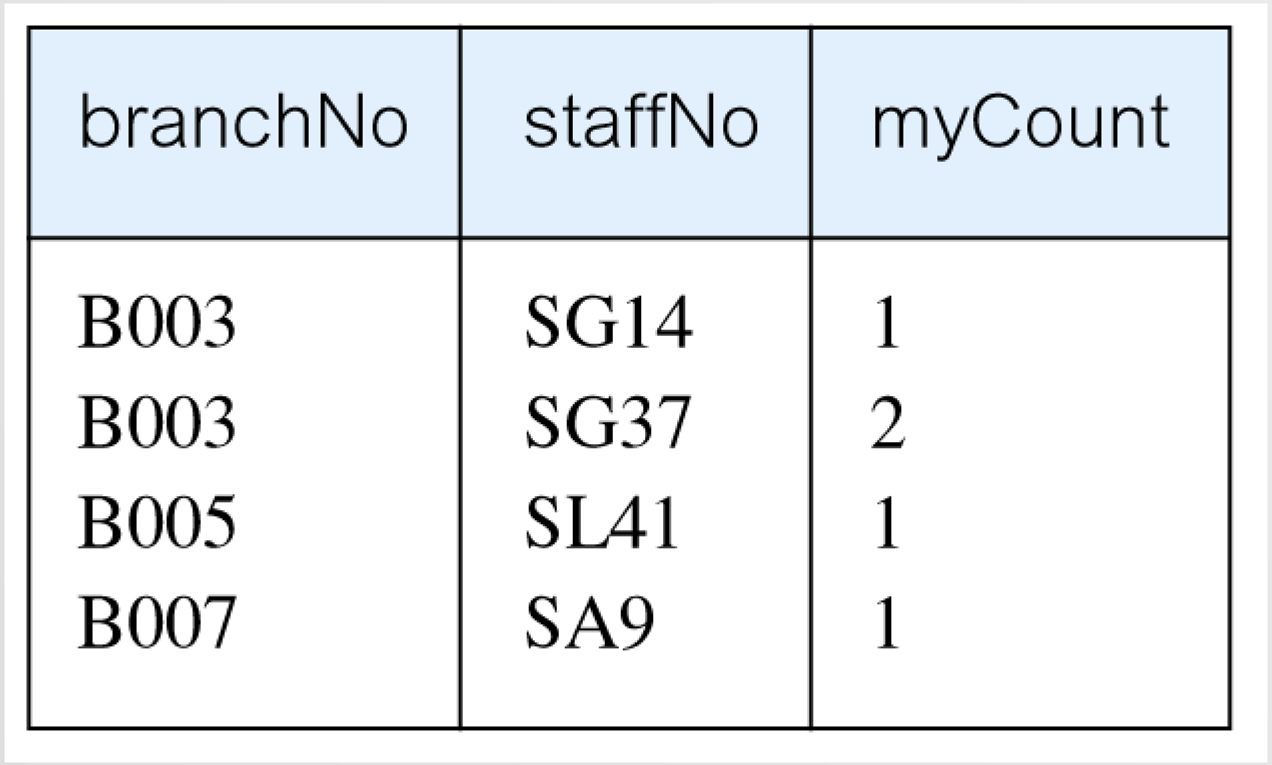
- Alternative formulation for FROM and WHERE:
- FROM (Branch b JOIN Staff s USING branchNo) AS
- bs JOIN PropertyForRent p USING staffNo

EXAMPLE 27 – MULTIPLE GROUPING COLUMNS

- Find 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;
```

EXAMPLE 27 – MULTIPLE GROUPING COLUMNS



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

COMPUTING A JOIN

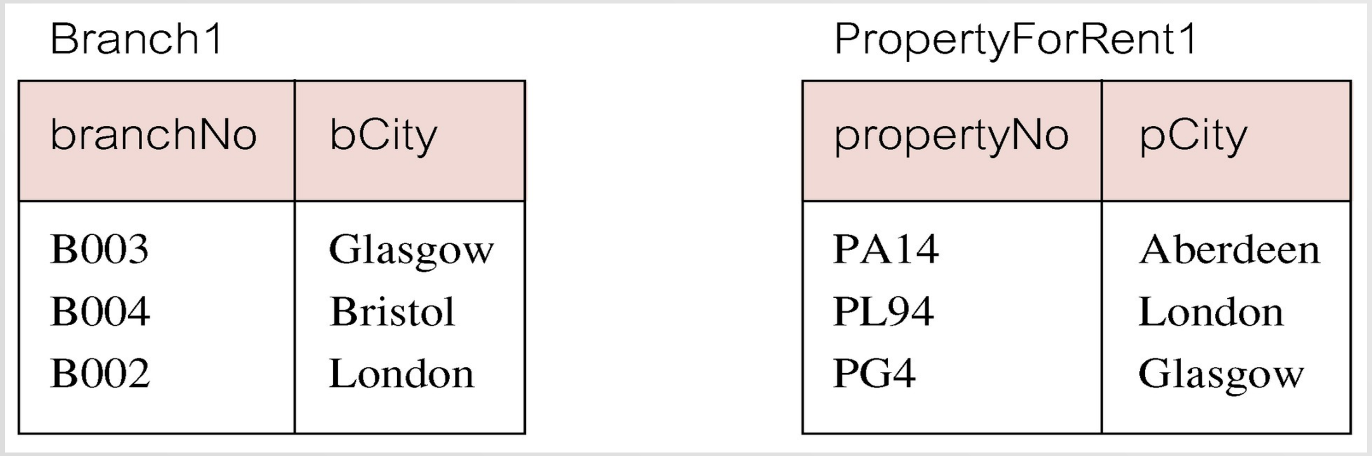
- Procedure for generating results of a join are:

1. Form Cartesian product of the tables named in 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 value of each item in SELECT list to produce a single row in result table.
4. If DISTINCT has been specified, eliminate any duplicate rows from the result table.
5. If there is an ORDER BY clause, sort result table as required.
6. SQL provides special format of SELECT for Cartesian product:

```
SELECT    [DISTINCT | ALL]    {* | columnList}
FROM Table1 CROSS JOIN Table2
```

OUTER JOIN

- 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

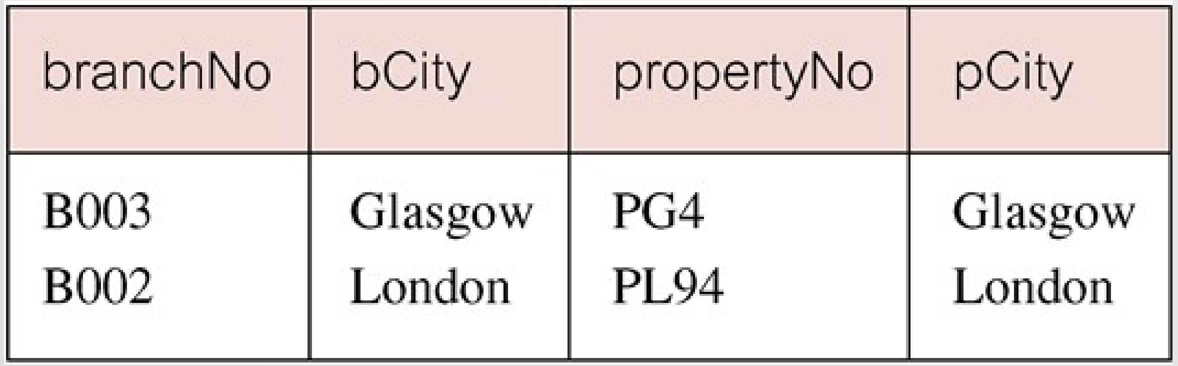
PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

OUTER JOIN

- 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 JOIN

- 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.

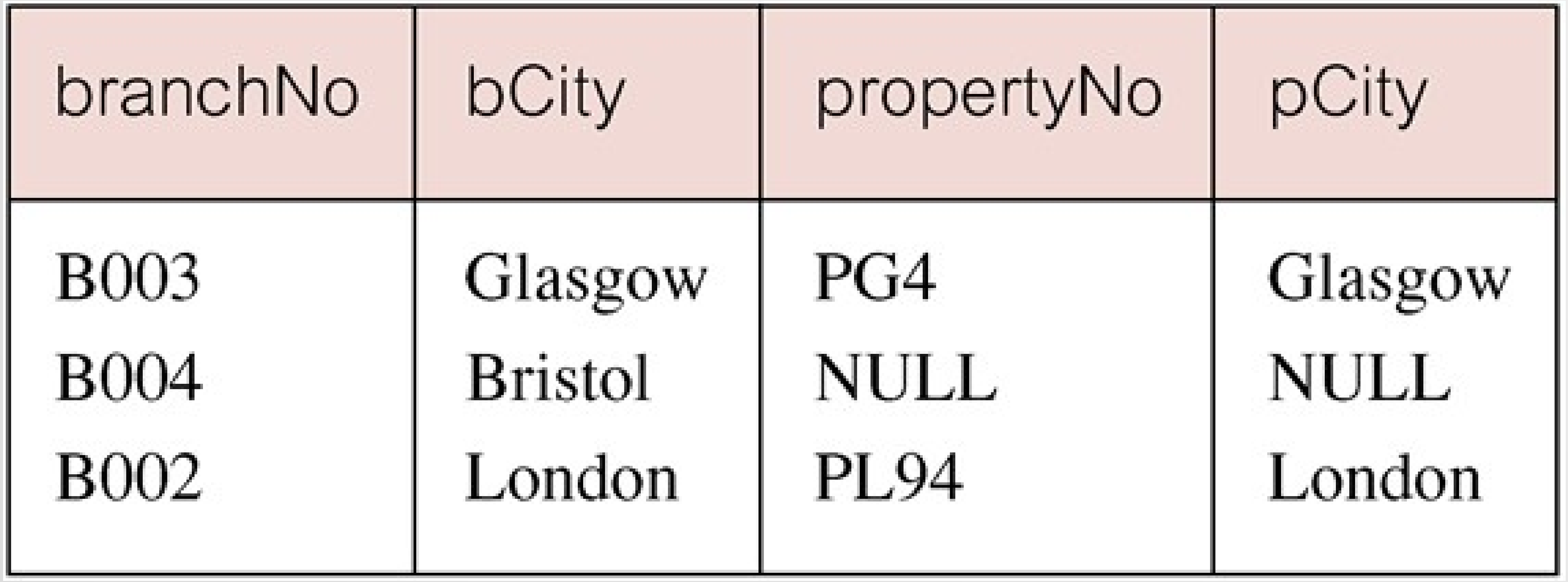
EXAMPLE 28 – LEFT OUTER JOIN

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

```
SELECT b.*, p.*

FROM Branch1 b LEFT JOIN

     PropertyForRent1 p ON b.bCity = p.pCity;
```

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

EXAMPLE 28 – LEFT OUTER JOIN



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

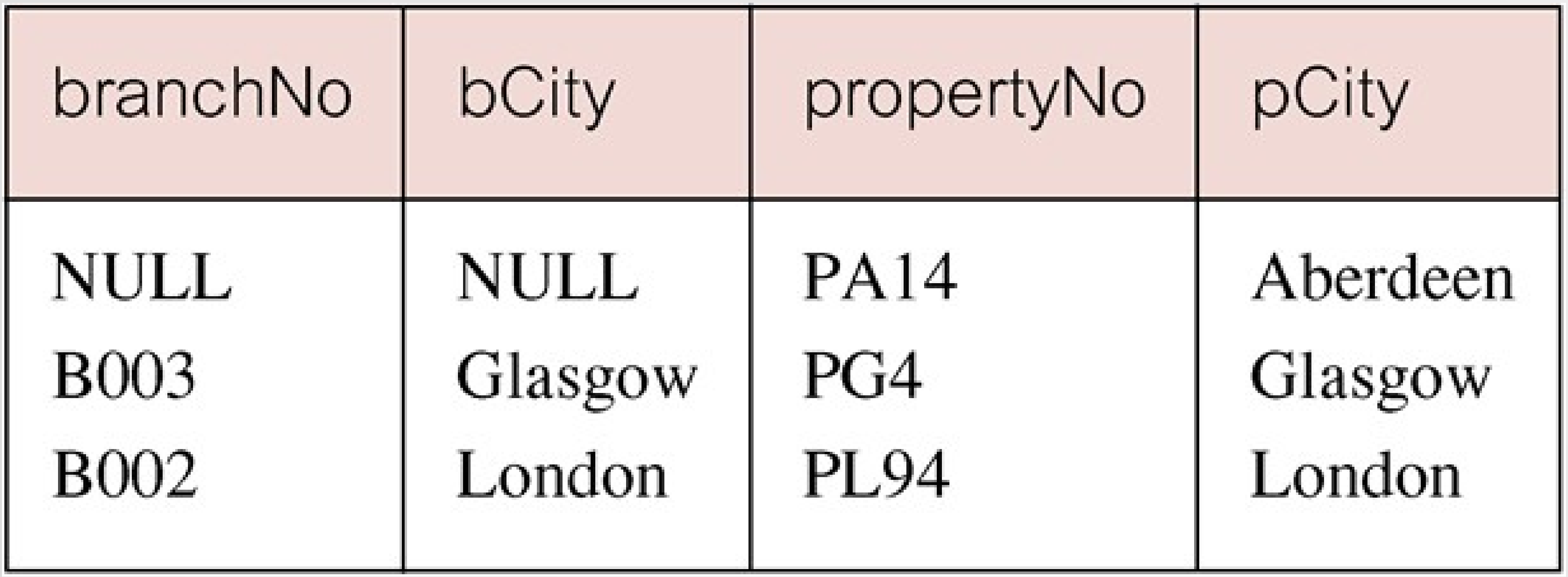
EXAMPLE 29 – RIGHT OUTER JOIN

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

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

- 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.

EXAMPLE 29 – RIGHT OUTER JOIN



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

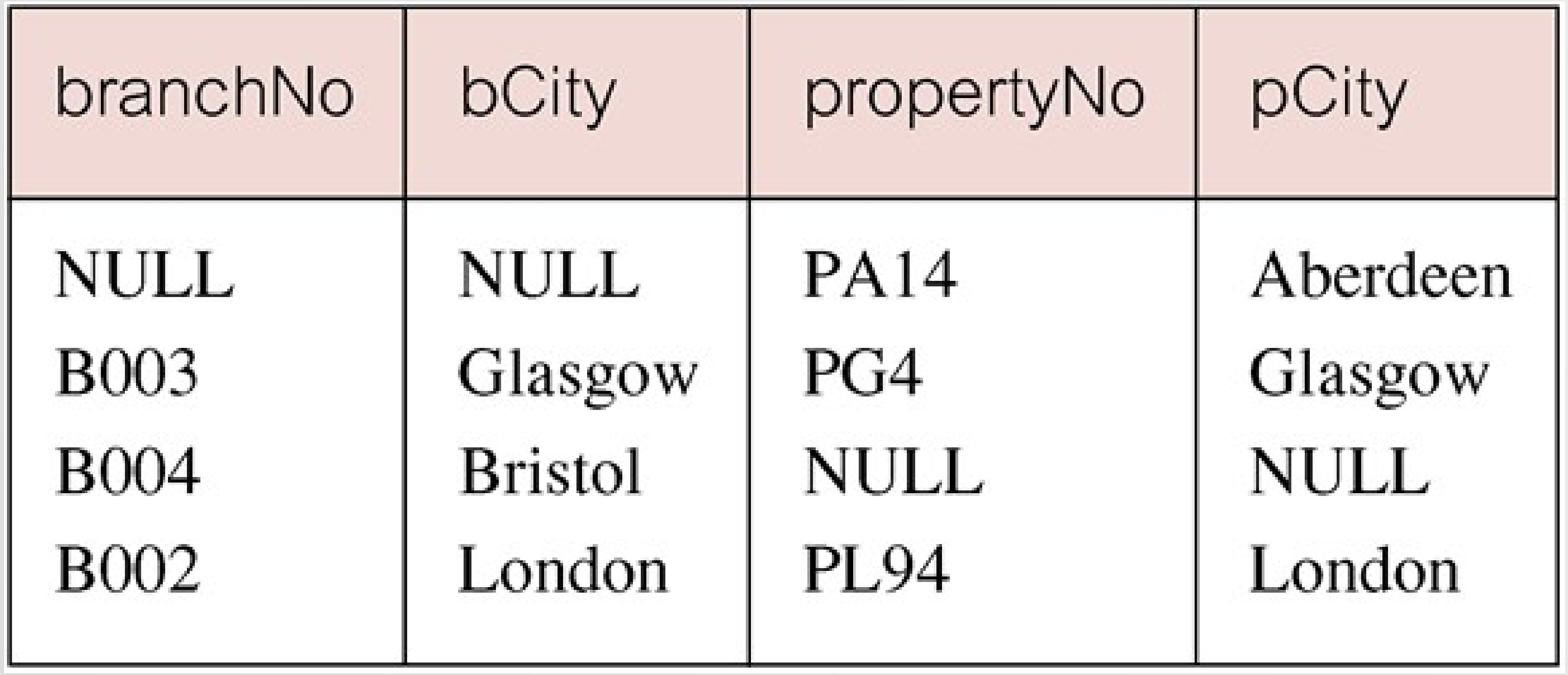
EXAMPLE 30 – FULL OUTER JOIN

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

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

- Includes rows that are unmatched in both tables.
- Unmatched columns are filled with NULLs.

EXAMPLE 30 – FULL OUTER JOIN



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

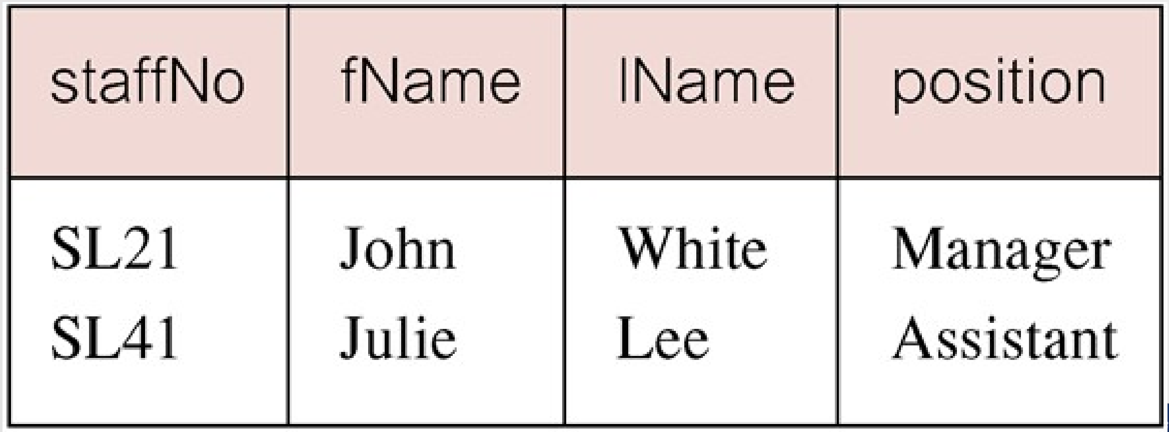
EXISTS AND NOT EXISTS

- 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 31 – 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 31 – QUERY USING EXISTS

- Note, 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 31 – QUERY USING EXISTS

- Could also write this query using join construct:

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

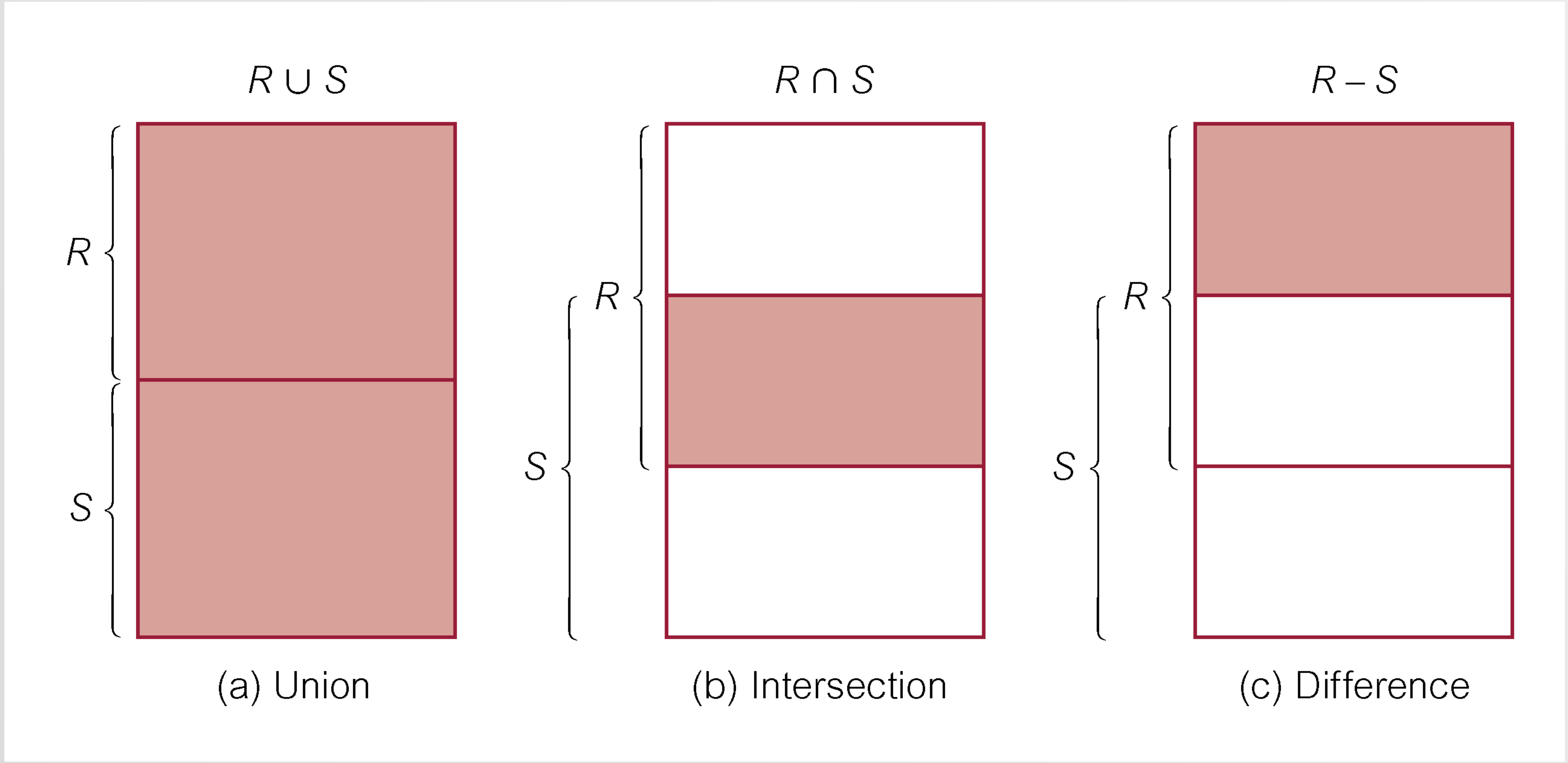
UNION, INTERSECT, AND DIFFERENCE (EXCEPT)

- 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)



(a) Union (b) Intersection (c) Difference

EXAMPLE 32 – 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);
```

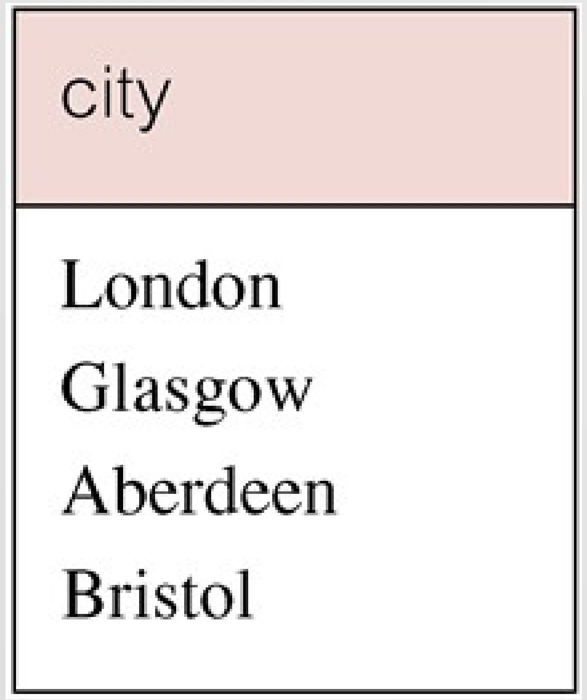
EXAMPLE 32 – USE OF UNION

- Or

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

EXAMPLE 32 – USE OF UNION

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



city
London
Glasgow
Aberdeen
Bristol

EXAMPLE 33 – USE OF INTERSECT

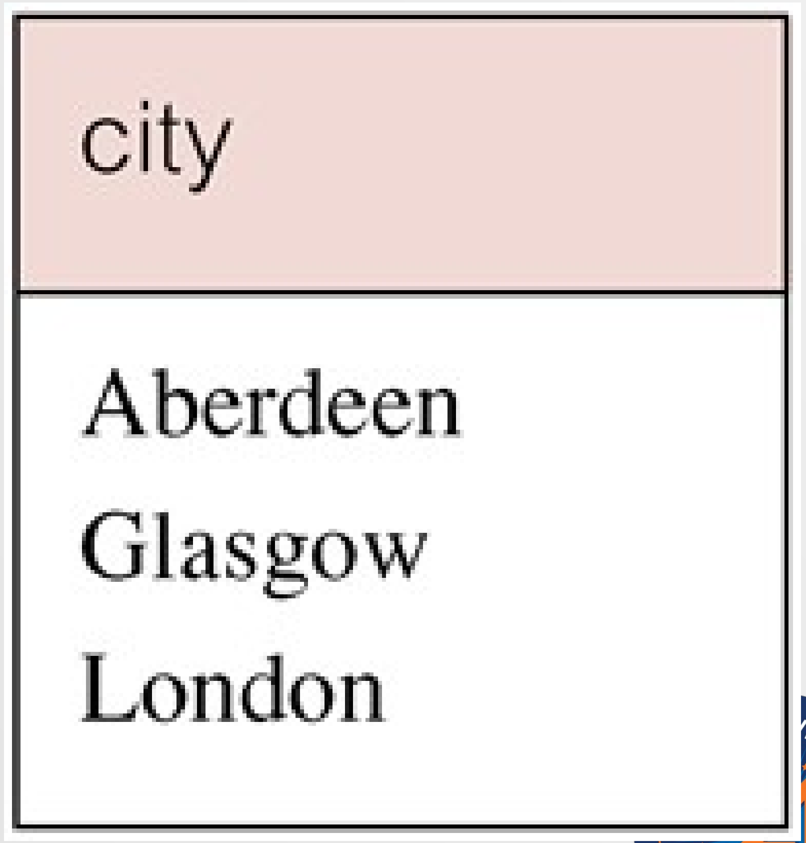
- List all cities where there is both a branch office and a property.

```
(SELECT city FROM Branch)

INTERSECT

(SELECT city FROM PropertyForRent);
```

EXAMPLE 33 – USE OF INTERSECT

- Or

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



city
Aberdeen
Glasgow
London

EXAMPLE 33 – 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);
```

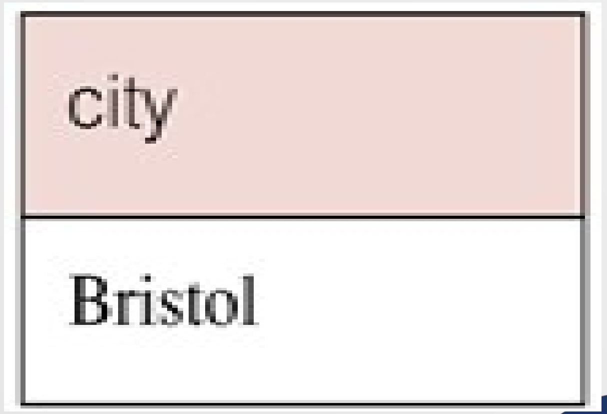
EXAMPLE 34 – USE OF EXCEPT

- 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 34 – USE OF EXCEPT

- Could rewrite this query without EXCEPT:

```
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);
```

INSERT

```
INSERT INTO TableName [ (columnList) ]
VALUES (dataValueList)
```

- columnList is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.
- dataValueList must match columnList as follows:
  - number of items in each list must be same;
  - must be direct correspondence in position of items in two lists;
  - data type of each item in dataValueList must be compatible with data type of corresponding column.

INSERT

```
INSERT INTO TableName [ (columnList) ]
VALUES (dataValueList)
```

- columnList is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.

EXAMPLE 35 – INSERT INTO … VALUES

- Insert a new row into Staff table supplying data for all column

```
INSERT INTO Staff

VALUES (‘SG16’, ‘Alan’, ‘Brown’, ‘Assistant’, ‘M’,
Date‘1957-05-25’, 8300, ‘B003’);
```

EXAMPLE 36 – INSERT USING DEFAULTS

- Insert a new row into Staff table supplying data for all mandatory columns.

```
INSERT INTO Staff (staffNo, fName, lName,
                   position, salary, branchNo)
VALUES (‘SG44’, ‘Anne’, ‘Jones’,
                   ‘Assistant’, 8100, ‘B003’);
```

- Or

```
INSERT INTO Staff
VALUES (‘SG44’, ‘Anne’, ‘Jones’, ‘Assistant’, NULL,
                   NULL, 8100, ‘B003’);
```

INSERT…SELECT

- Second form of INSERT allows multiple rows to be copied from one or more tables to another:

```
INSERT INTO TableName [ (columnList) ]

SELECT ...
```

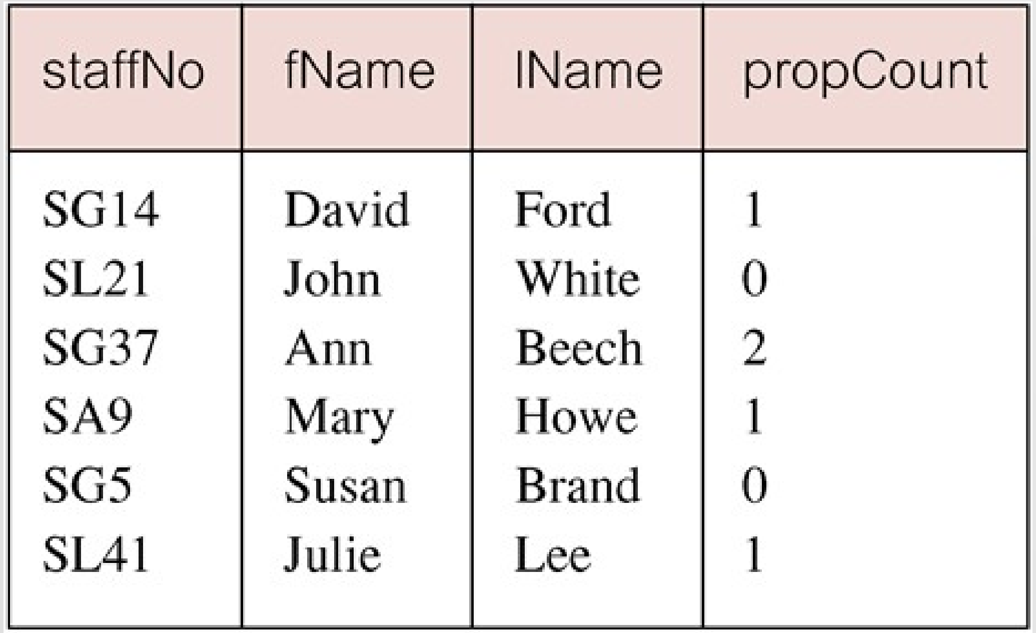
EXAMPLE 37 – INSERT … SELECT

- Assume there is a table StaffPropCount that contains names of staff and number of properties they manage:
- `StaffPropCount(staffNo, fName, lName, propCnt)`
- Populate StaffPropCount using Staff and PropertyForRent tables.

EXAMPLE 37 – INSERT … SELECT

```
INSERT INTO StaffPropCount
(SELECT s.staffNo, fName, lName, COUNT(*)
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.staffNo, fName, lName)
UNION
(SELECT staffNo, fName, lName, 0
FROM Staff
WHERE staffNo NOT IN
(SELECT DISTINCT staffNo
     FROM PropertyForRent));
```

EXAMPLE 37 – INSERT … SELECT



staffNo	fName	lName	propCount
SG14	David	Ford	1
SL21	John	White	0
SG37	Ann	Beech	2
SA9	Mary	Howe	1
SG5	Susan	Brand	0
SL41	Julie	Lee	1

- If second part of UNION is omitted, excludes those staff who currently do not manage any properties.

UPDATE

```
UPDATE TableName
SET columnName1 = dataValue1
          [, columnName2 = dataValue2...]
[WHERE searchCondition]
```

- TableName can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.

UPDATE

- WHERE clause is optional:
  - if omitted, named columns are updated for all rows in table;
  - if specified, only those rows that satisfy searchCondition are updated.
- New dataValue(s) must be compatible with data type for corresponding column.

EXAMPLE 38/39 – UPDATE

- Give all staff a 3% pay increase.

```
UPDATE Staff
SET salary = salary * 1.03;
```

- Give all Managers a 5% pay increase.

```
UPDATE Staff
SET salary = salary * 1.05
WHERE position = ‘Manager’;
```

EXAMPLE 40 – UPDATE

- Promote David Ford (staffNo=‘SG14’) to Manager and change his salary to £18,000.

```
UPDATE Staff

SET position = ‘Manager’, salary = 18000

WHERE staffNo = ‘SG14’;
```

DELETE

```
DELETE FROM TableName
[WHERE searchCondition]
```

- TableName can be name of a base table or an updatable view.
- searchCondition is optional; if omitted, all rows are deleted from table. This does not delete table. If search_condition is specified, only those rows that satisfy condition are deleted.

EXAMPLE 41/42 – DELETE

- Delete all viewings that relate to property PG4.

```
DELETE FROM Viewing
WHERE propertyNo = ‘PG4’;
```

- Delete all records from the Viewing table.

```
DELETE FROM Viewing;
```