Chapter 5

SQL: Data Manipulation

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 order of output

SELECT Statement

- **Order of clauses cannot be changed**
- **Only SELECT and FROM are mandatory**

Example 6.1 All Columns, All Rows

List full details of all staff.

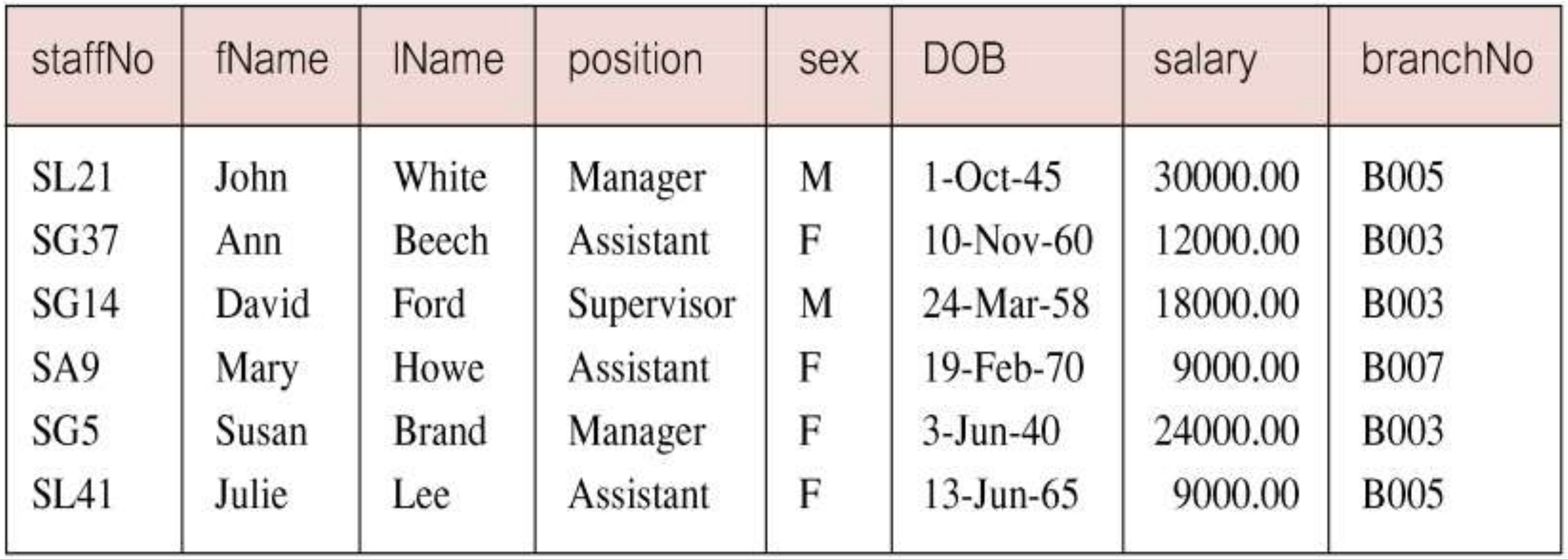
```
SELECT staffNo, fName, lName, address,
            position, sex, DOB, salary, branchNo
FROM Staff;
```

- **Can use * as an abbreviation for ‘all columns’:**

```
SELECT *
FROM Staff;
```

Example 6.1 All Columns, All Rows

Table 5.1 Result table for Example 5.1.



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

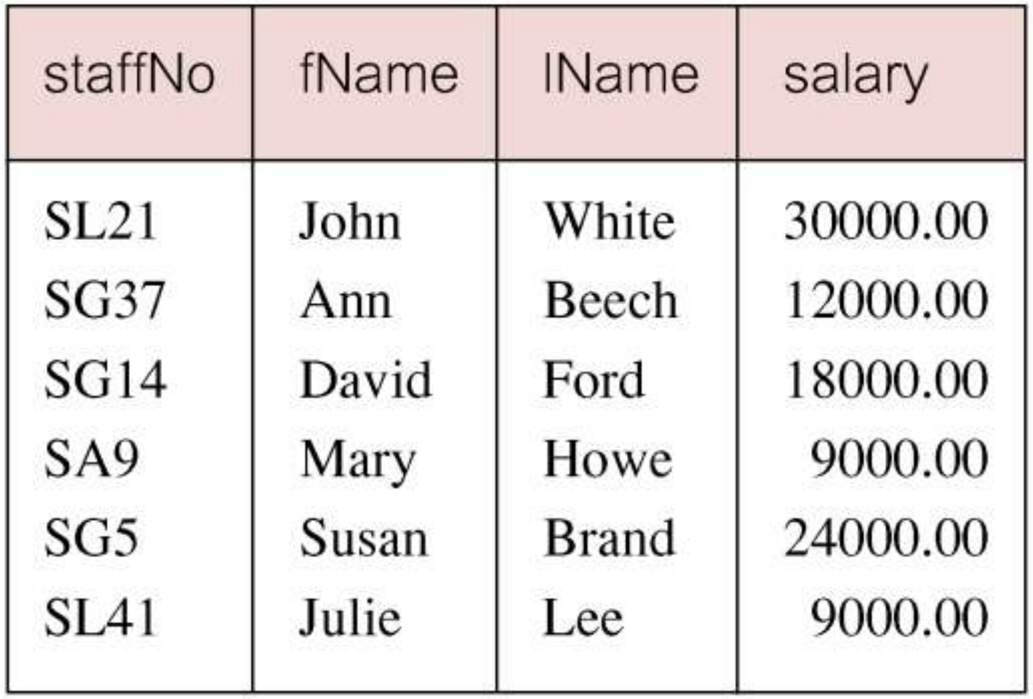
Example 6.2 Specific Columns, All Rows

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 6.2 Specific Columns, All Rows

Table 5.2 Result table for Example 5.2.

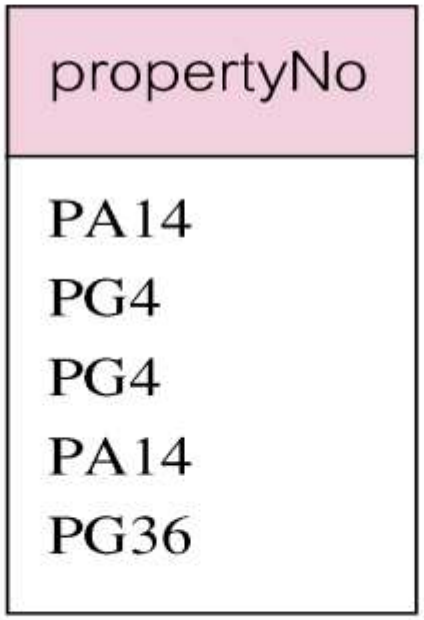


staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

Example 6.3 Use of DISTINCT

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

```
SELECT propertyNo
FROM Viewing;
```

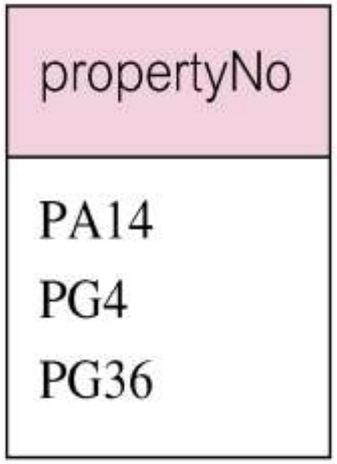


propertyNo
PA14
PG4
PG4
PA14
PG36

Example 6.3 Use of DISTINCT

- **Use DISTINCT to eliminate duplicates:**

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



propertyNo
PA14
PG4
PG36

Example 6.4 Calculated Fields

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

SELECT staffNo, fName, lName, salary/12

Fl

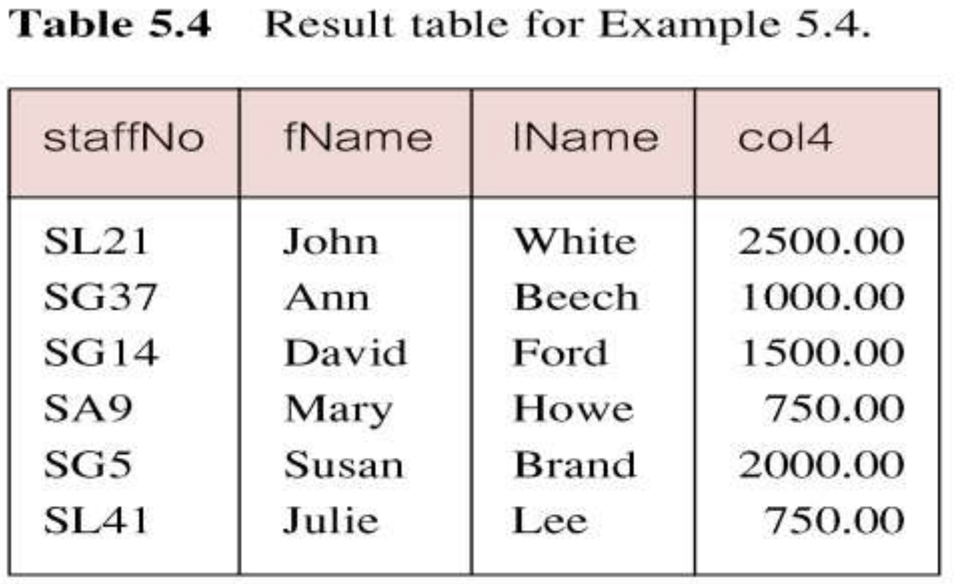


Table 5.4 Result table for Example 5.4.

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 6.4 Calculated Fields

- **To name column, use AS clause:**

```
SELECT staffNo, fName, lName, salary/12
            AS monthlySalary
FROM Staff;
```

Example 6.5 Comparison Search Condition

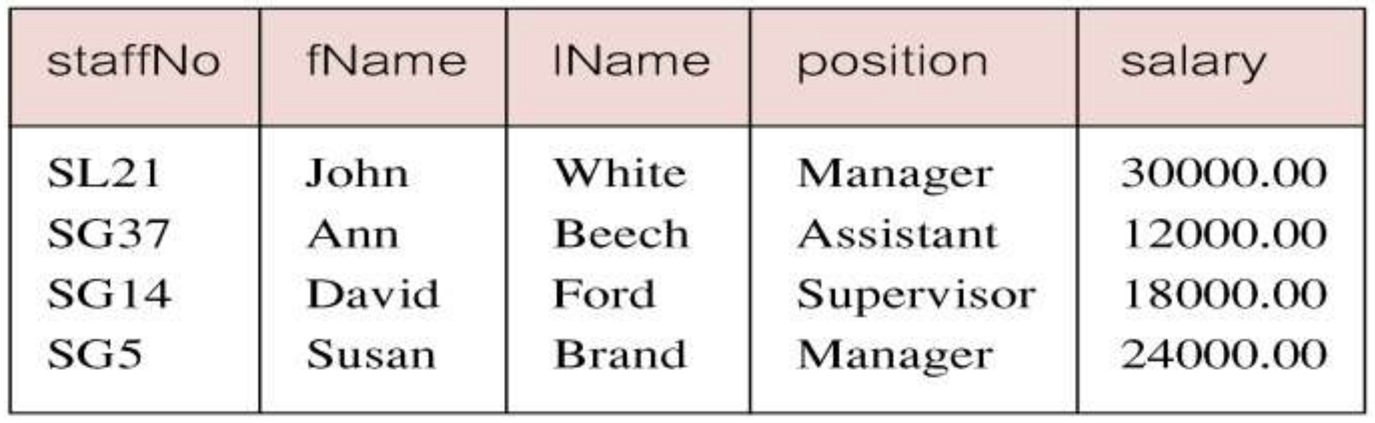
List all staff with a salary greater than 10,000.

SELECT staffNo, fName, lName, position, salary

FROM Staff

W

Table 5.5 Result table for Example 5.5.



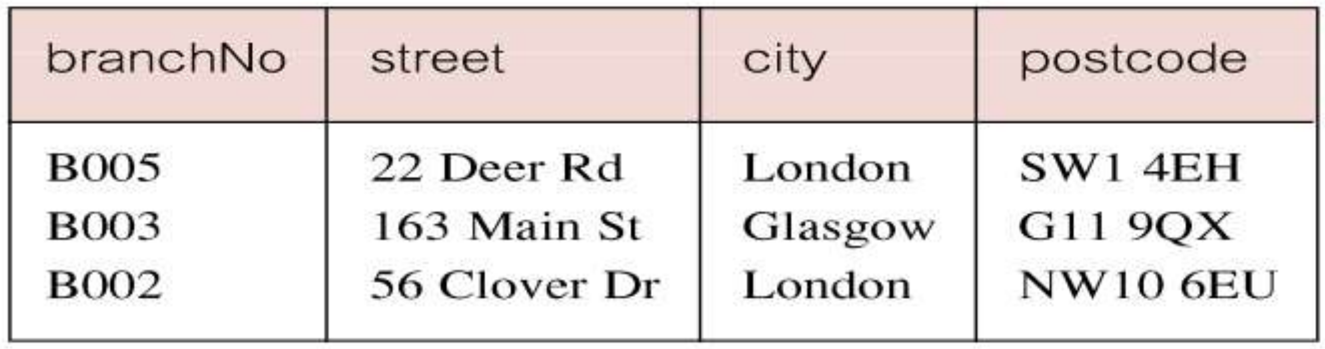
staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Example 6.6 Compound Comparison Search Condition

List addresses of all branch offices in London or Glasgow.

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

Table 5.6 Result table for Example 5.6.



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 6.7 Range Search Condition

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 endpoints of range**

Example 6.7 Range Search Condition

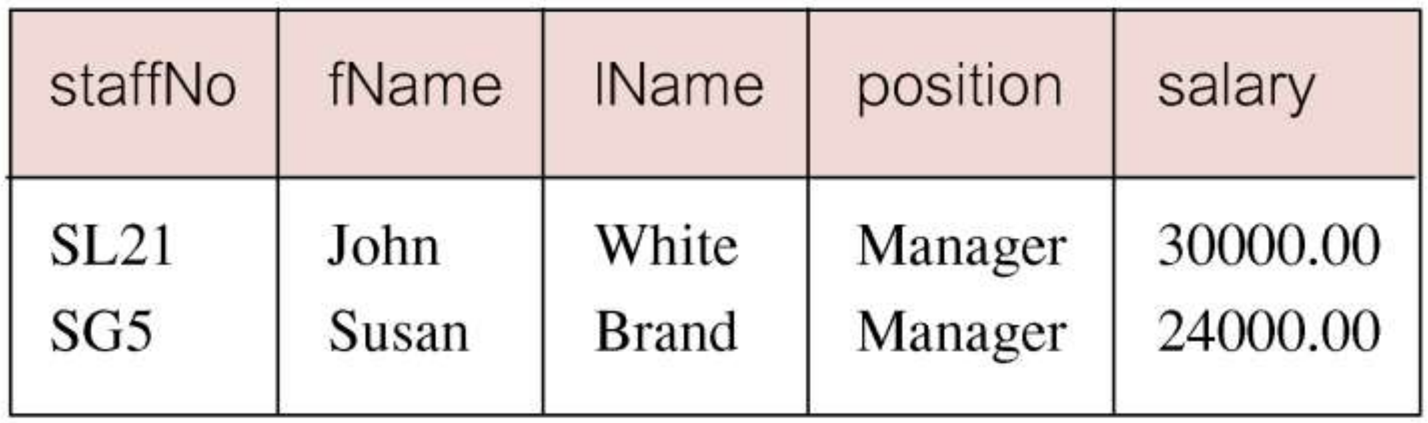


Table 5.7 Result table for Example 5.7.

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

Example 6.7 Range Search Condition

- 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;
```

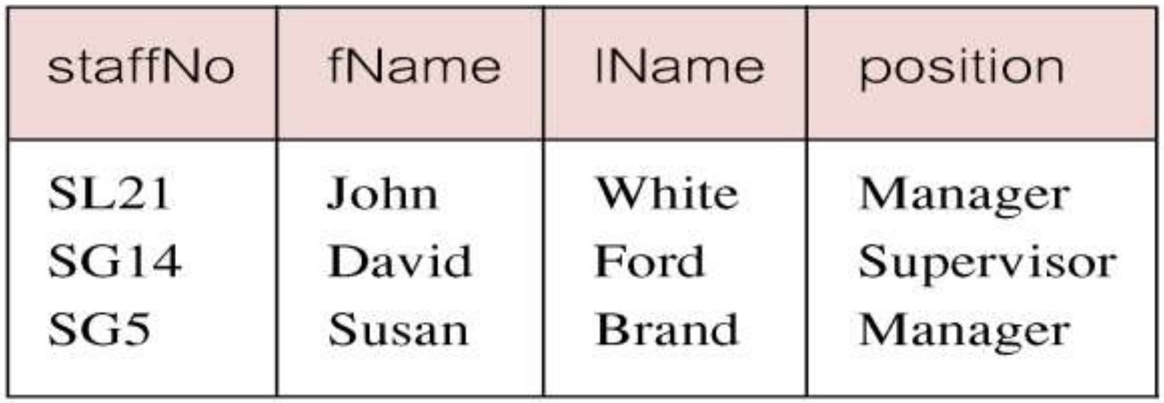
- Useful for range of values

Example 6.8 Set Membership

List all managers and supervisors.

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

Table 5.8 Result table for Example 5.8.



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

Example 6.8 Set Membership

- 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 more efficient when set contains many values

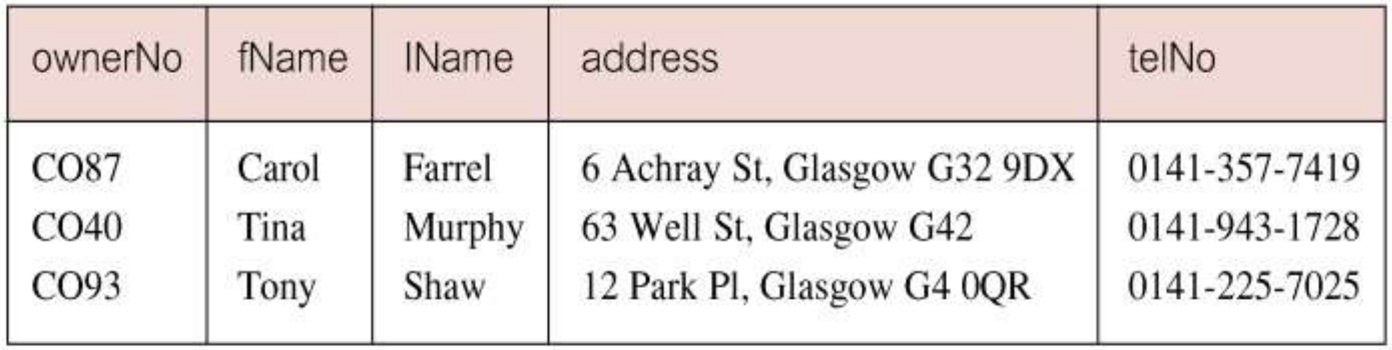
Example 6.9 Pattern Matching

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

SELECT ownerNo, fName, lName, address, telNo

FROM PrivateOwner

Table 5.9 Result table for Example 5.9.



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 6.9 Pattern Matching

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

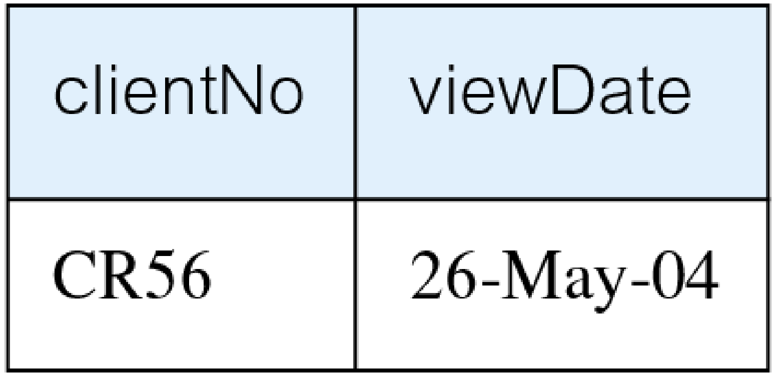
Example 6.10 NULL Search Condition

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 6.10 NULL Search Condition



clientNo	viewDate
CR56	26-May-04

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

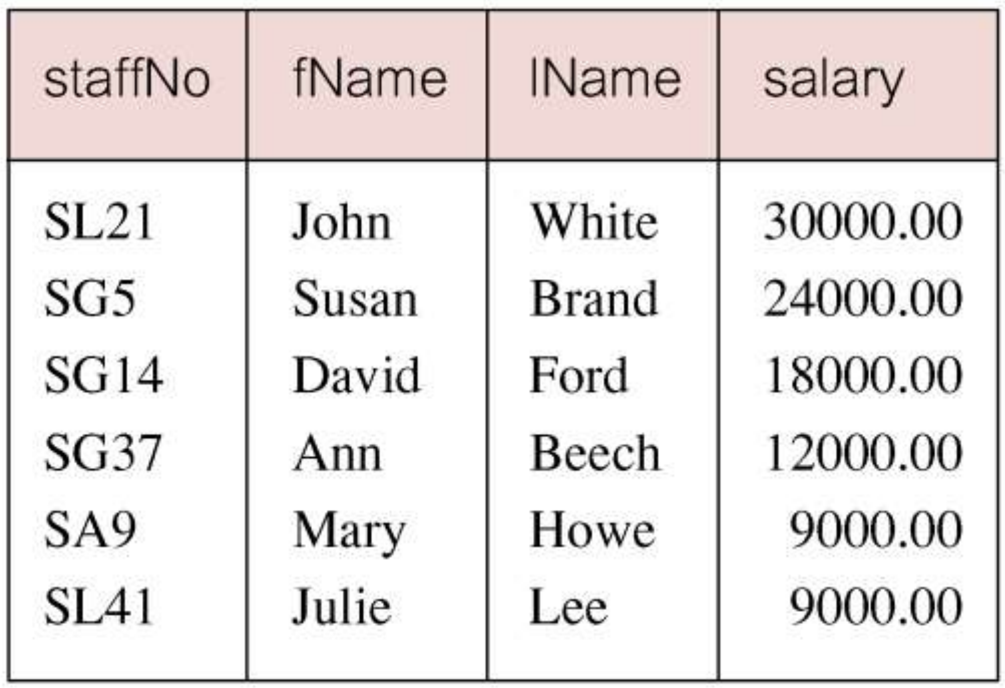
Example 6.11 Single Column Ordering

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

```
SELECT staffNo, fName, lName, salary
FROM Staff
ORDER BY salary DESC;
```

Example 6.11 Single Column Ordering

Table 5.11 Result table for Example 5.11.



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 6.12 Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

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

Example 6.12 Multiple Column Ordering

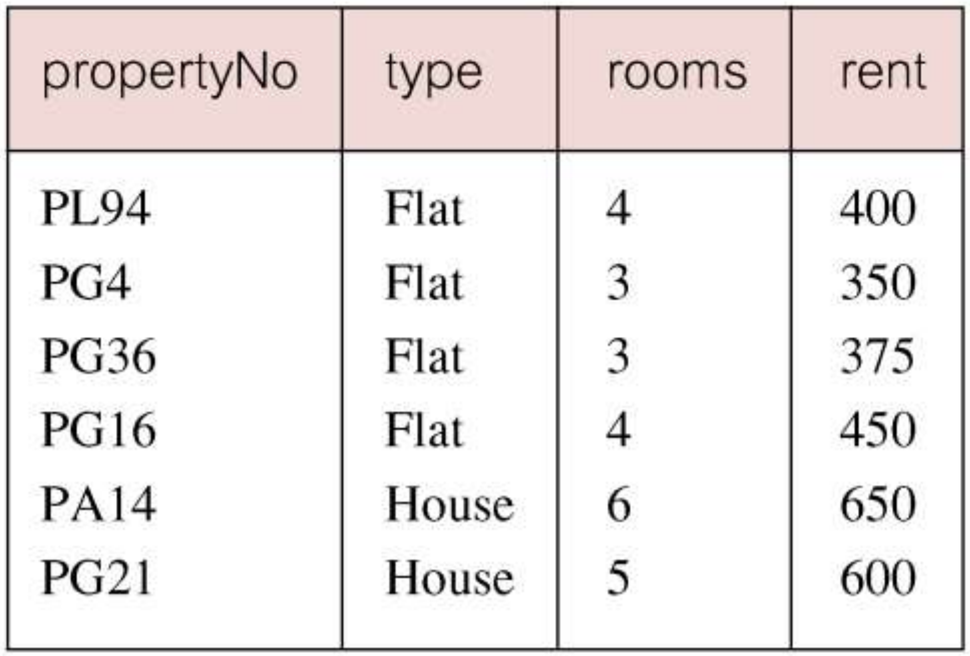


Table 5.12(a) Result table for Example 5.12 with one sort key.

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 6.12 Multiple Column Ordering

- **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;
```

Example 6.12 Multiple Column Ordering

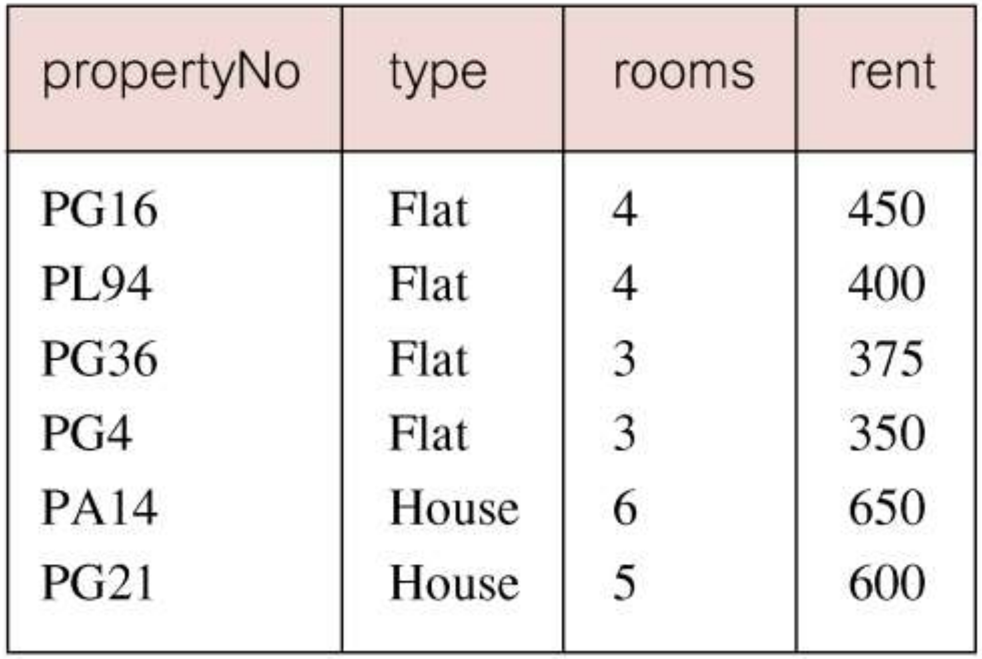


Table 5.12(b) Result table for Example 5.12 with two sort keys.

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 single column of table and returns single value
- COUNT, MIN, and MAX apply to numeric and non-numeric fields
  - SUM and AVG used on numeric fields only
- Each function eliminates nulls first and operates only on remaining non-null values
  - Except COUNT

SELECT Statement - Aggregates

- COUNT(*) counts all rows of table
  - Includes nulls and duplicate values
- Can use DISTINCT before column name to eliminate duplicates
- DISTINCT has no effect with MIN/MAX
  - Has effect with SUM/AVG

SELECT Statement - Aggregates

- Aggregate functions used only in SELECT list and HAVING clause

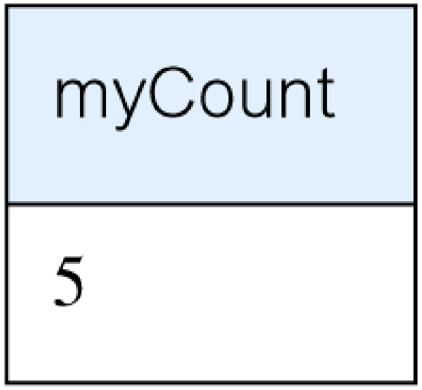
- If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference column out with aggregate function
- Illegal:

```
SELECT staffNo, COUNT(salary)
 FROM Staff;
```

Example 6.13 Use of COUNT(*)

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

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

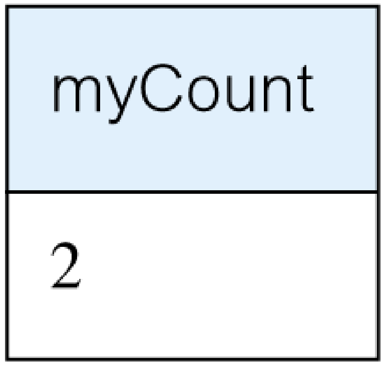


myCount
5

Example 6.14 Use of COUNT(DISTINCT)

How many different properties viewed in May ‘04?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN ‘1-May-(
       AND ‘31-May-04’;
```

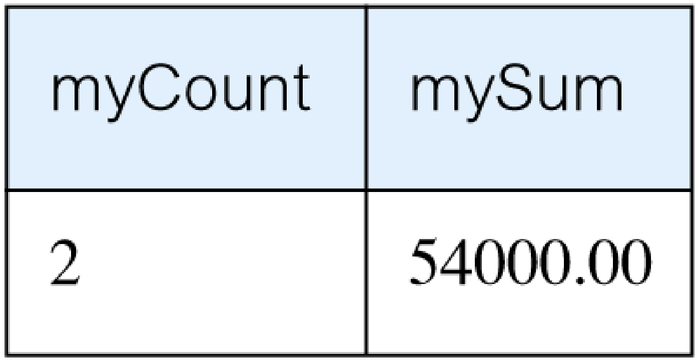


myCount
2

Example 6.15 Use of COUNT and SUM

Find number of Managers and sum of their salaries.

```
SELECT COUNT(staffNo) AS myCount,
        SUM(salary) AS mySum
FROM Staff
WHERE position = ‘Manager’;
```

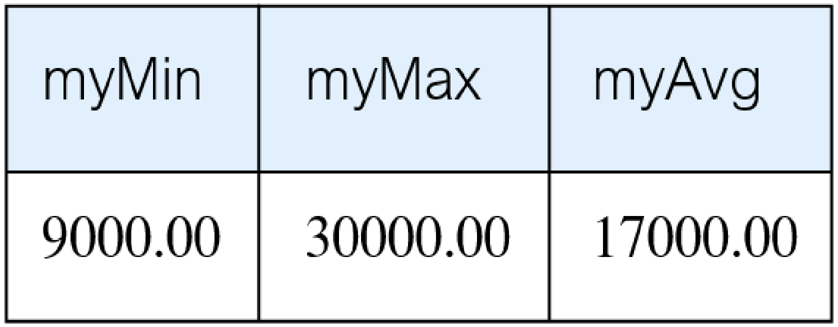


myCount	mySum
2	54000.00

Example 6.16 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***
  - **SELECT clause may only contain:**
    - **column names**
    - **aggregate functions**
    - **constants**
    - **expression involving combinations of above**

SELECT Statement - Grouping

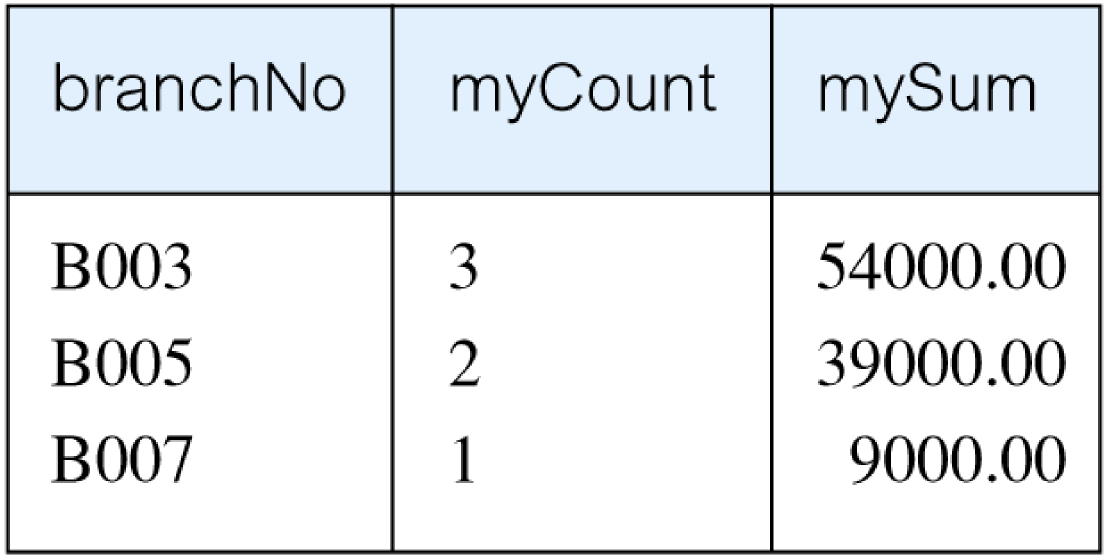
- All column names in SELECT list must appear in GROUP BY clause unless name used only in aggregate function
- If WHERE used with GROUP BY:
  - WHERE applied first
  - Then groups formed from remaining rows satisfying predicate
- ISO considers two nulls to be equal for purposes of GROUP BY

Example 6.17 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;
```

Example 6.17 Use of GROUP BY



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

Restricted Groupings – HAVING clause

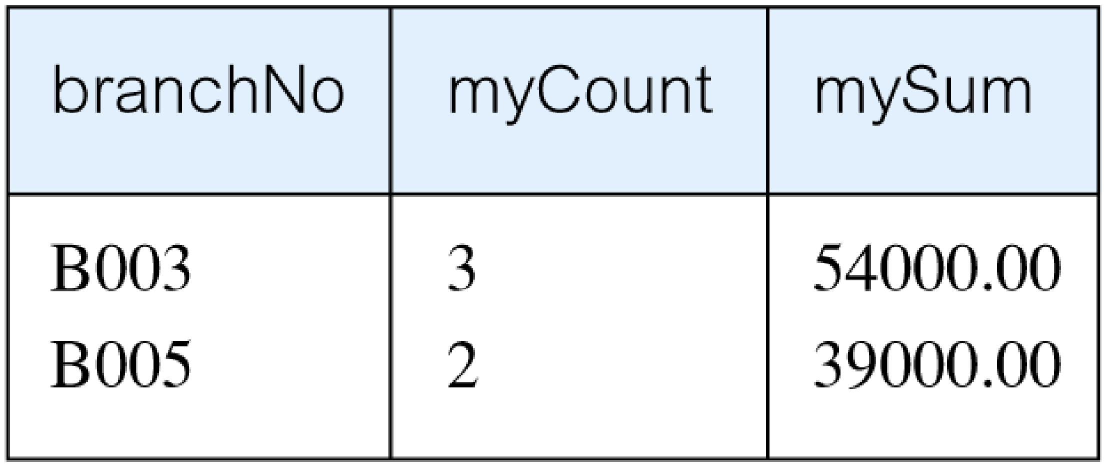
- **HAVING clause designed for use with GROUP BY to restrict groups that appear in final result table**
- **Similar to WHERE:**
  - **WHERE filters individual rows**
  - **HAVING filters groups**
- **Column names in HAVING clause must appear in GROUP BY list or be contained within aggregate function**

Example 6.18 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;
```

Example 6.18 Use of HAVING



branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Subqueries

- Some SQL statements can have SELECT embedded within them
- Ssubselect can be used in WHERE and HAVING clauses of an outer SELECT
  - Called *subquery* or *nested query*
- Subselects may also appear in INSERT, UPDATE, and DELETE statements

Example 6.19 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 6.19 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 6.19 Subquery with Equality

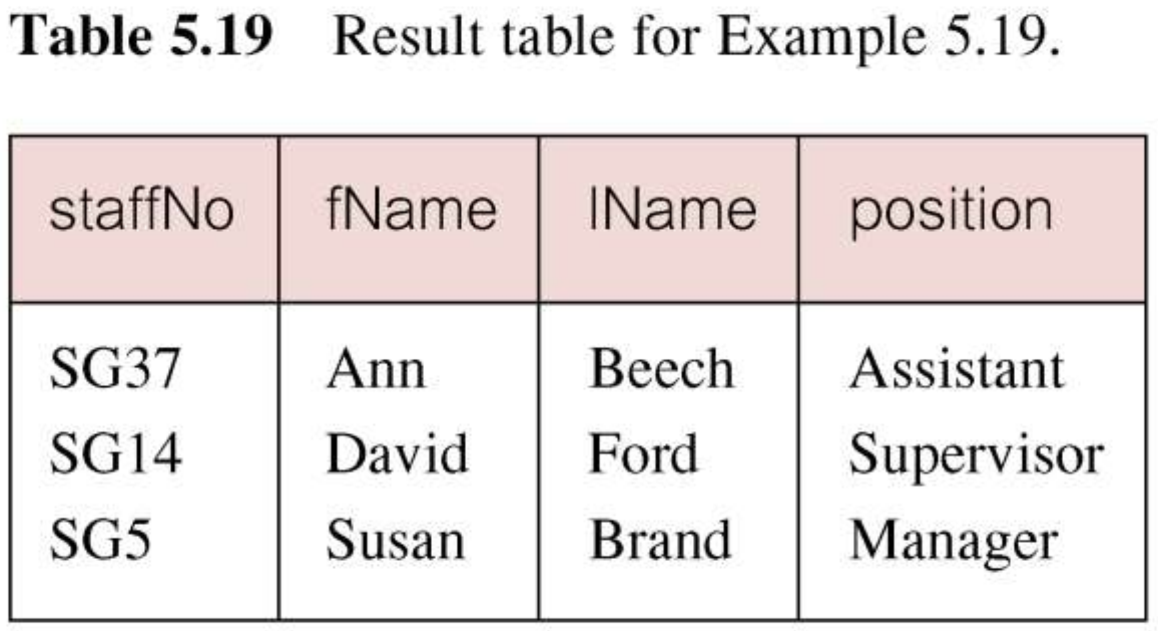


Table 5.19 Result table for Example 5.19.

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

Example 6.20 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 6.20 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;
```

Example 6.20 Subquery with Aggregate

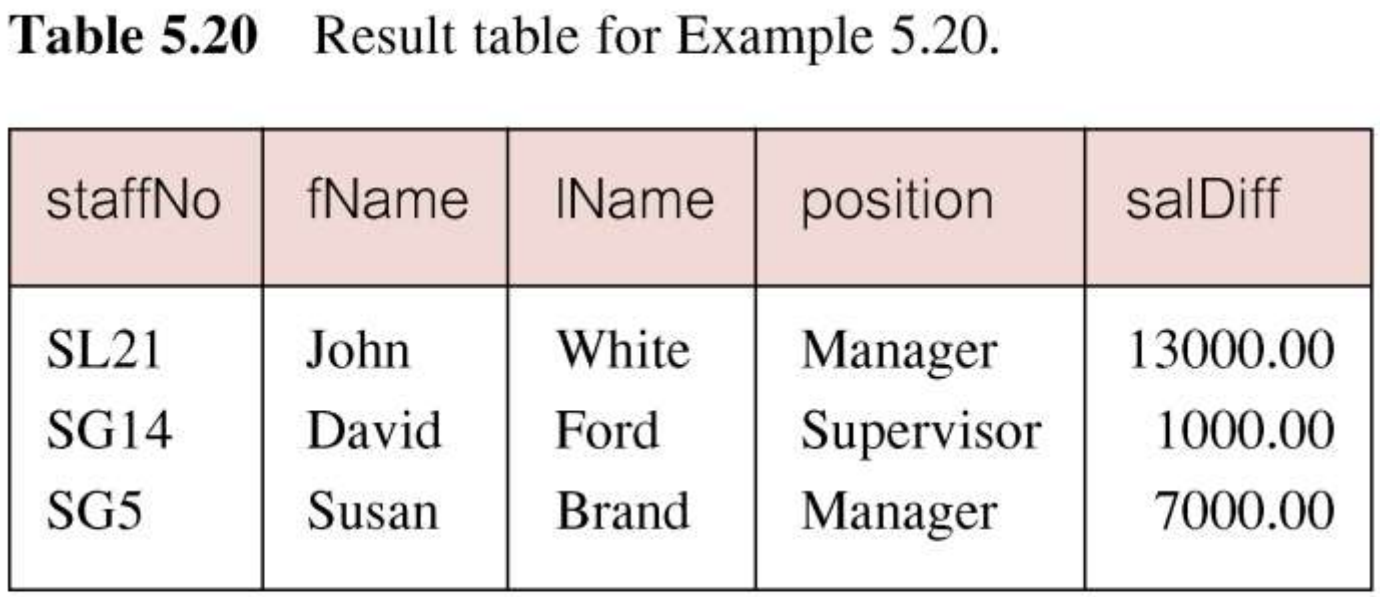


Table 5.20 Result table for Example 5.20.

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 subquery**
  - **May be used in outermost SELECT**
- **Subquery SELECT list must consist of 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 table in FROM using *alias***

Subquery Rules

- **When subquery is operand in comparison**
  - **Subquery must appear on right-hand side**
- **Subquery may not be used as operand in an expression**

Example 6.21 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’));
```

Chapter 6

SQL: Data Manipulation Cont’d

ANY and ALL

- ANY and ALL used with subqueries that produce single column of numbers
- ALL
  - Condition only true if satisfied by *all* values produced by subquery
- ANY
  - Condition true if satisfied by *any* values produced by subquery
- If subquery empty
  - ALL returns true
  - ANY returns false
- SOME may be used in place of ANY

Example 6.22 Use of ANY/SOME

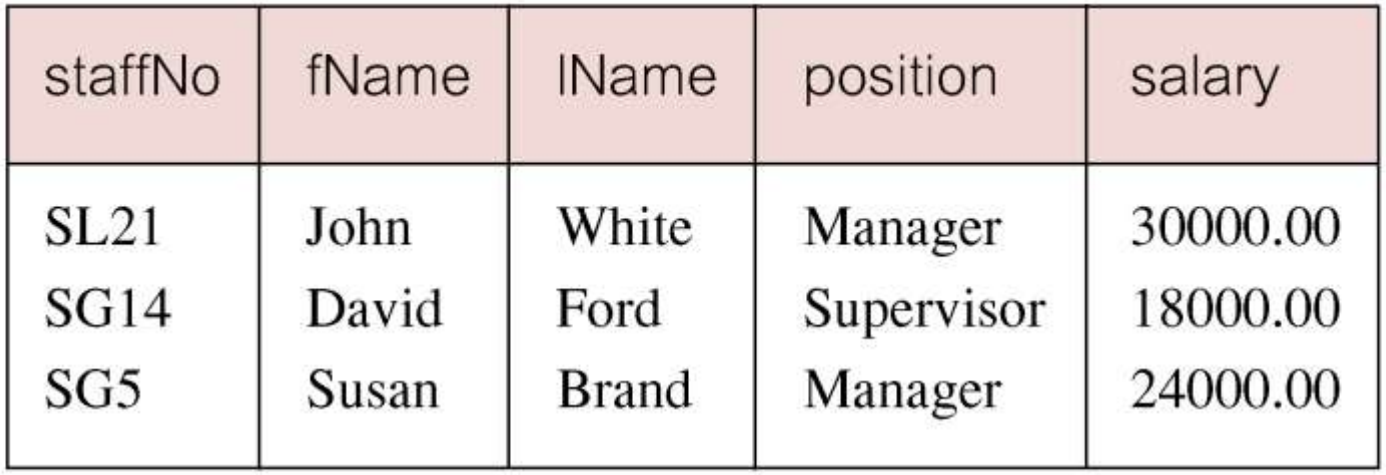
Find staff whose salary is larger than salary of at least one member of staff at branch B003.

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

Example 6.22 Use of ANY/SOME

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

Table 5.22 Result table for Example 5.22.



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

Example 6.23 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');
```

Example 6.23 Use of ALL

Table 5.23 Result table for Example 5.23.

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 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

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

Example 6.24 Simple Join

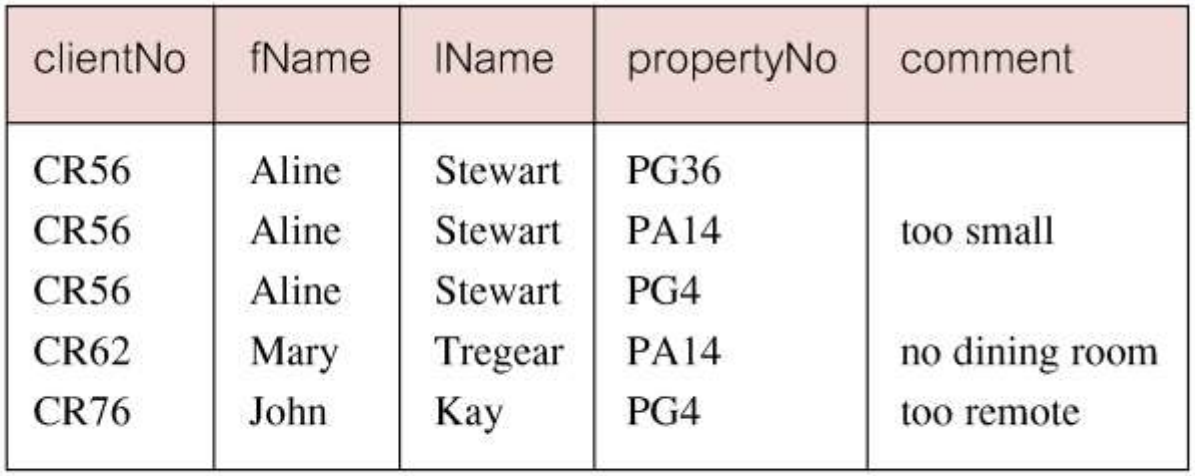
List names of all clients who have viewed a property along with any comment supplied.

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

Example 6.24 Simple Join

- **Only those rows from both tables that have identical values in clientNo columns (c.clientNo = v.clientNo) included in result**
- **Equivalent to equi-join in relational algebra**

Table 5.24 Result table for Example 5.24.



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
```

- **FROM replaces original FROM and WHERE**

Example 6.25 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 6.25 Sorting a join

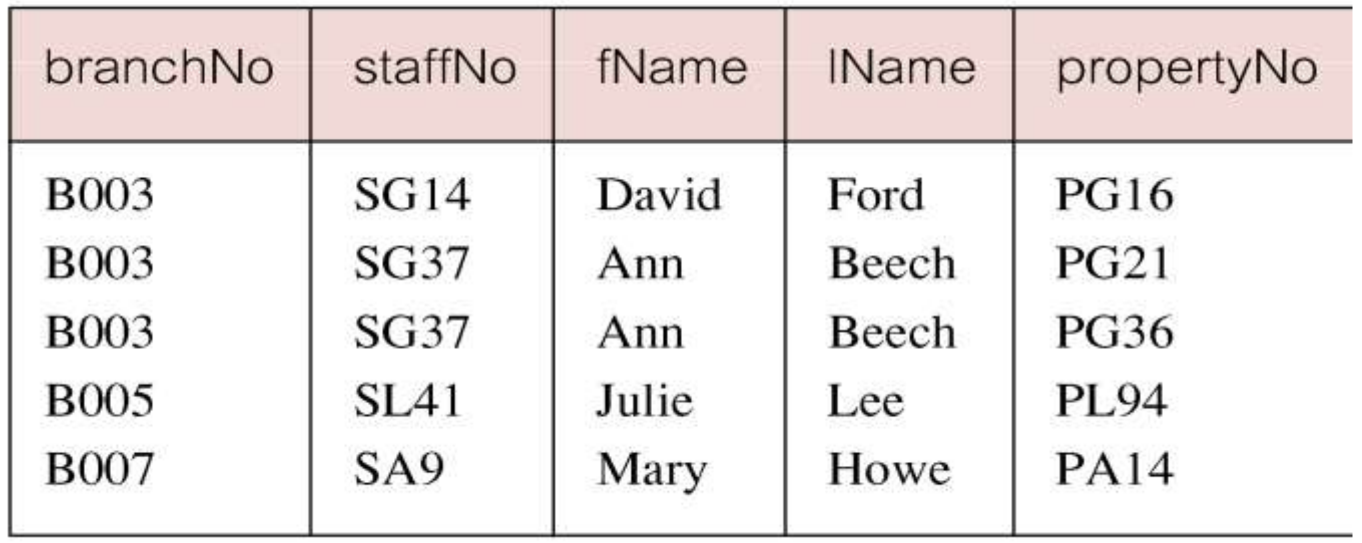


Table 5.25 Result table for Example 5.25.

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 6.26 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 6.26 Three Table Join

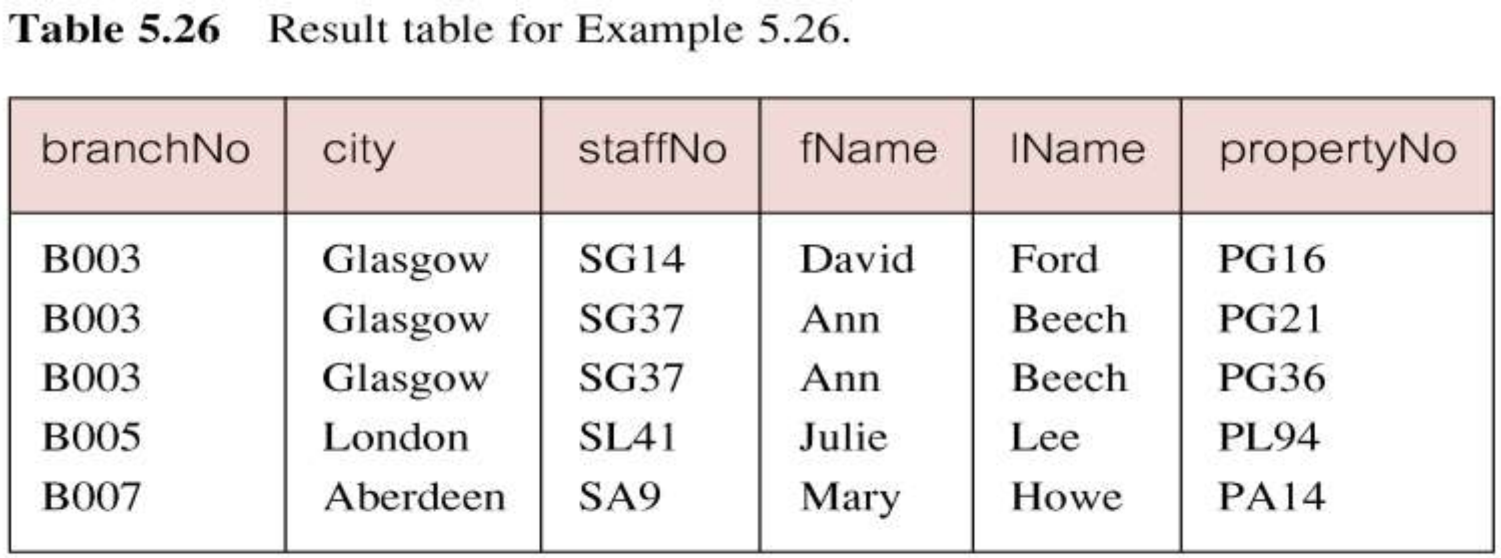


Table 5.26 Result table for Example 5.26.

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 6.27 Multiple Grouping Columns

Find number of properties handled by each staff member by branch.

```
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 6.27 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 tables named in FROM clause**
2. **If WHERE clause:**
   - **Apply search condition to each row of product table**
   - **Retain rows that satisfy condition**
3. **For each remaining row, determine value of each item in SELECT list to produce single row in result table**

Computing a Join

4. If DISTINCT specified, eliminate any duplicate rows from result table

6. If ORDER BY clause, sort result table as required

Outer Joins

- **If one row of joined table is unmatched, row omitted from result table**
- **Outer join operations retain rows that do not satisfy join condition**
- **Consider following tables:**

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

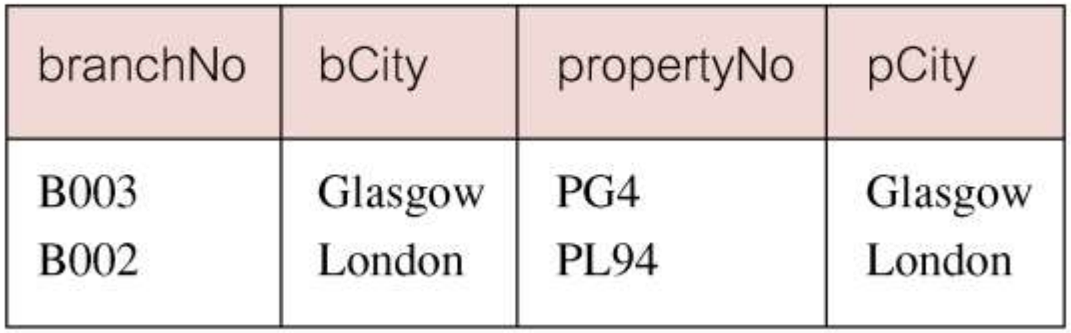
propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Outer Joins

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

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

Table 5.27(b) Result table for inner join of Branch1 and PropertyForRent1 tables.



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

Example 6.28 Left Outer Join

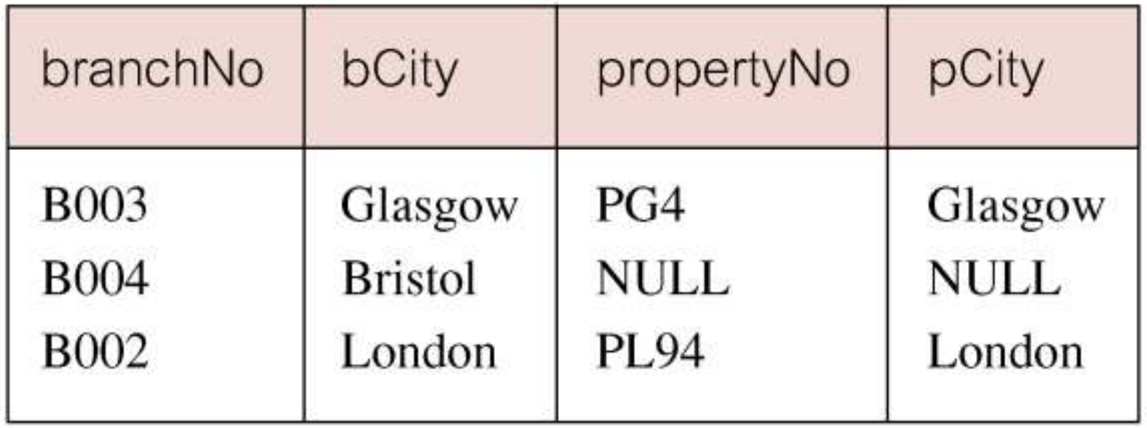
List branches and properties that are in same city along with any unmatched branches.

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

Example 6.28 Left Outer Join

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

Table 5.28 Result table for Example 5.28.



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

Example 6.29 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;
```

Example 6.29 Right Outer Join

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

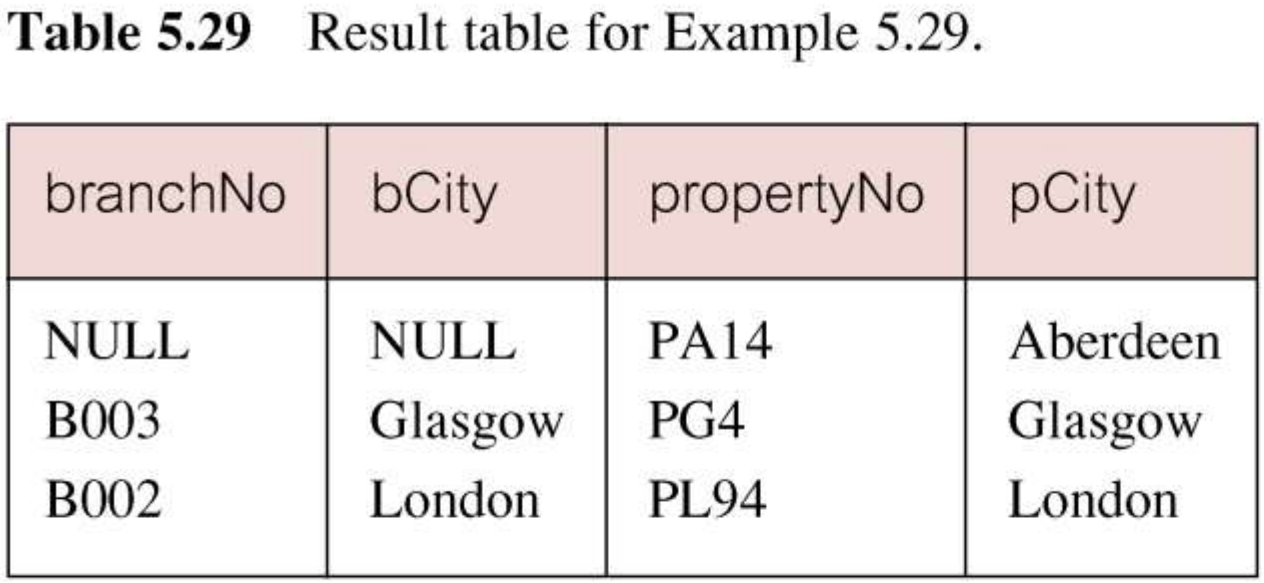


Table 5.29 Result table for Example 5.29.

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

Example 6.30 Full Outer Join

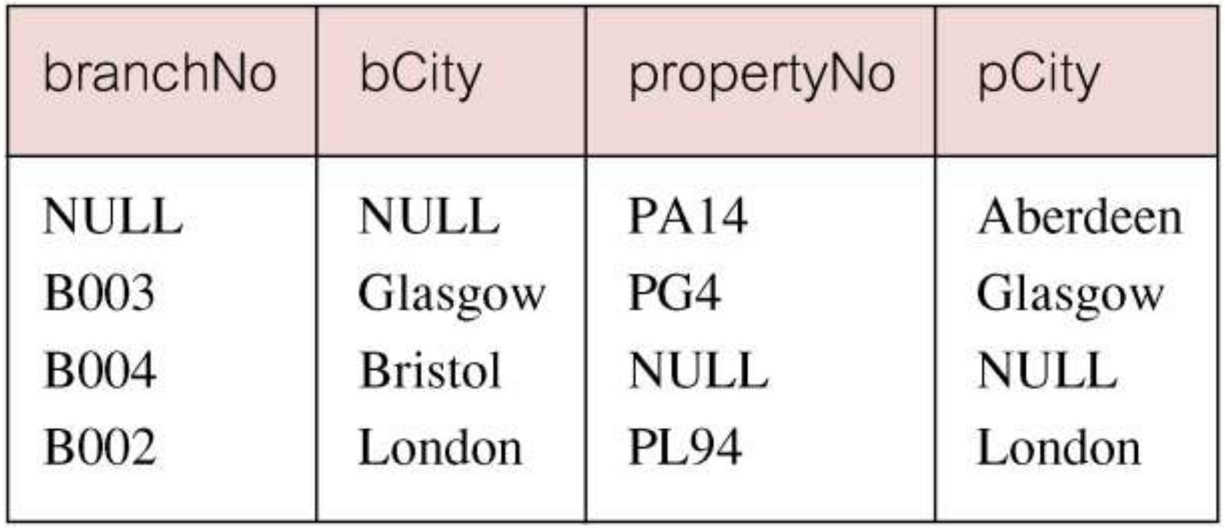
List branches and properties in same city and any unmatched branches or properties.

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

Example 6.30 Full Outer Join

- **Includes rows unmatched in both tables**
- **Unmatched columns filled with NULLs**

Table 5.30 Result table for Example 5.30.



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 used only with subqueries**
- **Produce 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 empty result table**
- **NOT EXISTS is the opposite of EXISTS**

EXISTS and NOT 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 * ...)
```

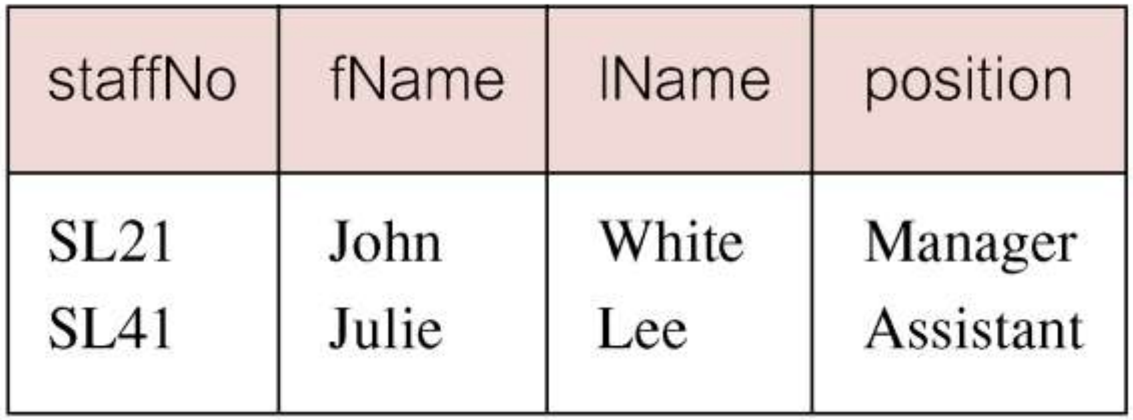
Example 6.31 Query using EXISTS

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');
```

Example 6.31 Query using EXISTS

Table 5.31 Result table for Example 5.31.



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

Example 6.31 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 6.31 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 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***

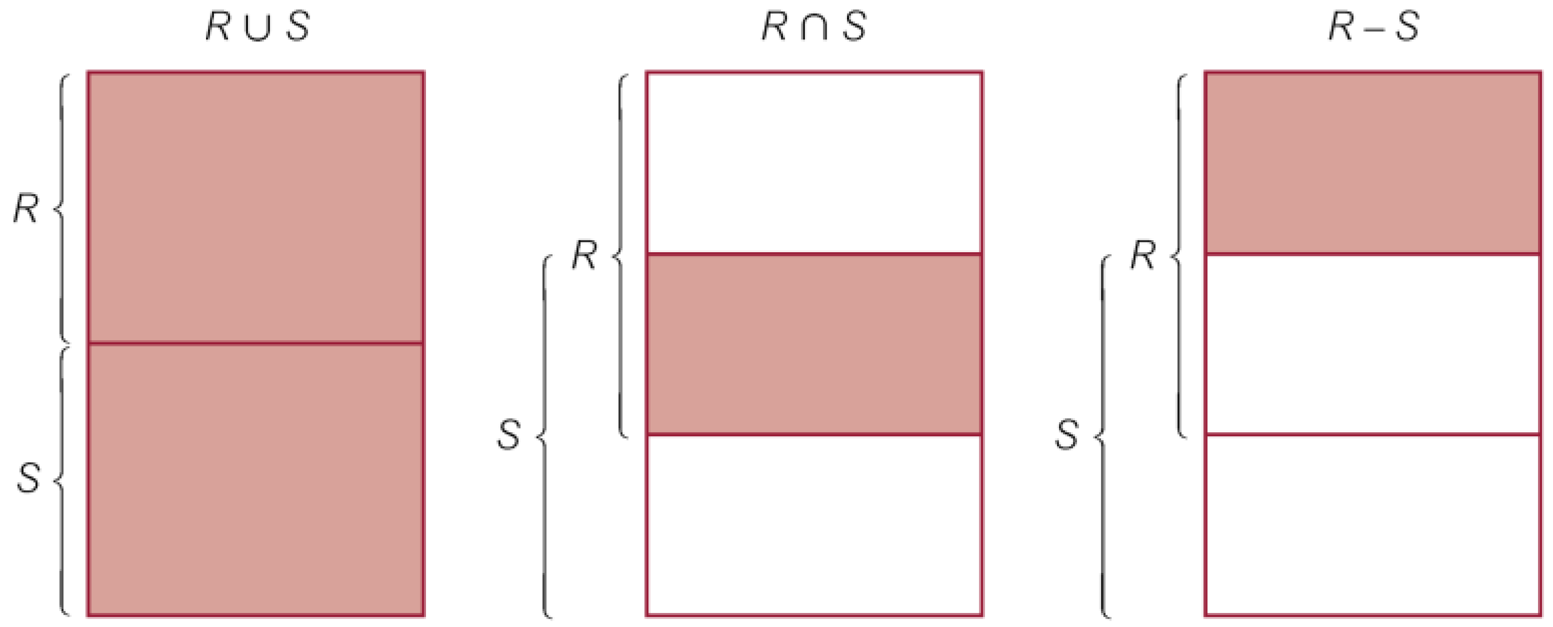
Union, Intersect, and Difference (Except)

- **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 6.32 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 6.32 Use of UNION

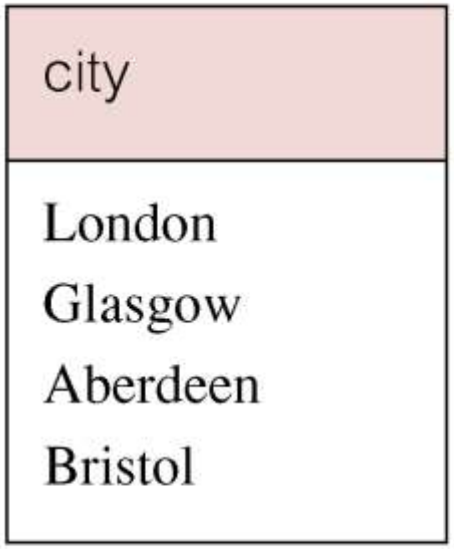
- Or

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

Example 6.32 Use of UNION

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

Table 5.32 Result table for Example 5.32.



city
London
Glasgow
Aberdeen
Bristol

Example 6.33 Use of INTERSECT

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

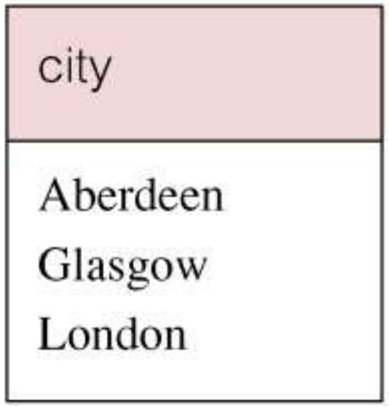
```
(SELECT city FROM Branch)
INTERSECT
(SELECT city FROM PropertyForRent);
```

Example 6.33 Use of INTERSECT

- **Or**

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

Table 5.33 Result table for Example 5.33.



city
Aberdeen
Glasgow
London

Example 6.33 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 6.34 Use of EXCEPT

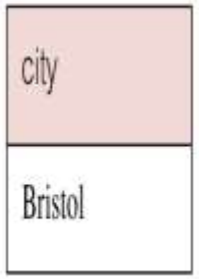
List of all cities where there is a branch office but no properties.

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

- Or

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

Table 5.34 Result table for Example 5.34.



city
Bristol

Example 6.34 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* optional; if omitted, SQL assumes list of all columns in original CREATE TABLE order
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT specified when creating column

INSERT

- ***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**

Example 6.35 INSERT … VALUES

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

```
INSERT INTO Staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M',
   Date'1957-05-25', 8300, 'B003');
```

Example 6.36 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 6.37 INSERT … SELECT

Assume there is a table StaffPropCount that contains names of staff and number of properties they manage:

StaffPropCount(<u>staffNo</u>, fName, lName, propCnt)

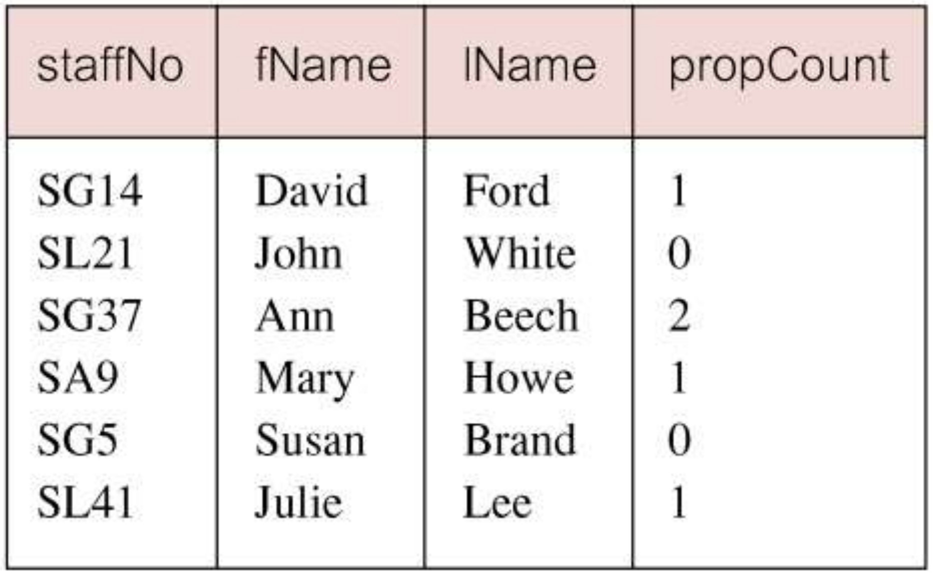
Populate StaffPropCount using Staff and PropertyForRent tables.

Example 6.37 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 6.37 INSERT … SELECT

Table 5.35 Result table for Example 5.37.



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

- **ted, 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 base table or updatable view
- SET clause specifies names of one or more columns to be updated

UPDATE

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

Example 6.38/39 UPDATE All Rows

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 6.40 UPDATE Multiple Columns

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 base table or updatable view
- *searchCondition* optional; if omitted, all rows deleted from table
  - Table not deleted
- If *search_condition* specified, only rows that satisfy condition deleted

Example 6.41/42 DELETE Specific Rows

Delete all viewings that relate to property PG4.

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

Delete all records from the Viewing table.

```
DELETE FROM Viewing;
```

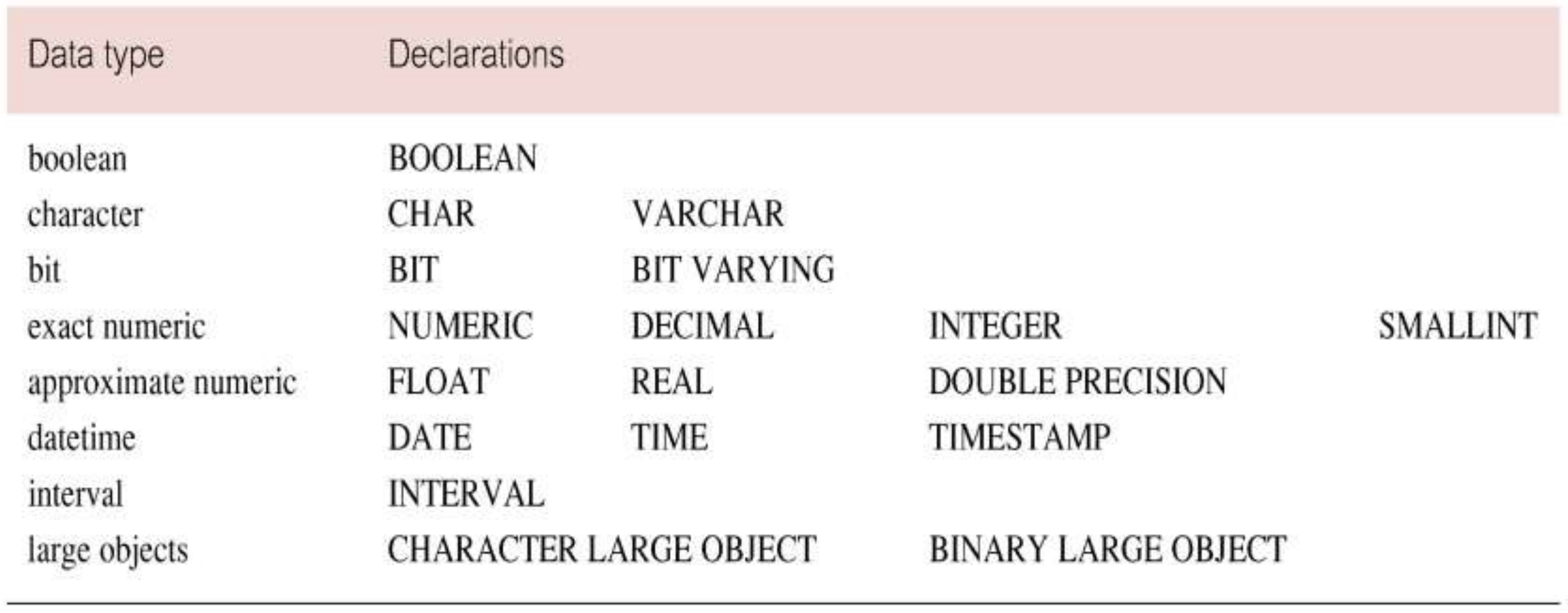
Chapter 7

SQL: Data Definition

Pearson Education © 2009

ISO SQL Data Types

Table 6.1 ISO SQL data types.



Data type	Declarations			
boolean	BOOLEAN			
character	CHAR	VARCHAR		
bit	BIT	BIT VARYING		
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION	
datetime	DATE	TIME	TIMESTAMP	
interval	INTERVAL			
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

Pearson Education © 2009

Integrity Enhancement Feature

- Integrity constraints:
  - required data
  - domain constraints
  - entity integrity
  - referential integrity
  - general constraints.

Pearson Education © 2009

Integrity Enhancement Feature

Required Data

```
position        VARCHAR(10)        NOT NULL
```

Domain Constraints

(a) CHECK

```
sex      CHAR NOT NULL
         CHECK (sex IN (‘M’, ‘F’))
```

Integrity Enhancement Feature

(b) <u>CREATE DOMAIN</u>

```
CREATE DOMAIN DomainName [AS] dataType
[DEFAULT defaultOption]
[CHECK (searchCondition)]
```

For example:

```
CREATE DOMAIN SexType AS CHAR
        CHECK (VALUE IN (‘M’, ‘F’));
sex     SexType     NOT NULL
```

Integrity Enhancement Feature

- *searchCondition* can involve a table lookup:

```
CREATE DOMAIN BranchNo AS CHAR(4)
CHECK (VALUE IN (SELECT branchNo
                        FROM Branch));
```

- Domains can be removed using DROP DOMAIN:

```
DROP DOMAIN DomainName
  [RESTRICT | CASCADE]
```

IEF - Entity Integrity

- **Primary key of table must contain unique, non-null value for each row**
- **ISO standard supports FOREIGN KEY clause in CREATE and ALTER TABLE statements:**

```
PRIMARY KEY(staffNo)
PRIMARY KEY(clientNo, propertyNo)
```

- **Can only have one PRIMARY KEY clause per table**
- **Can still ensure uniqueness for alternate keys using UNIQUE:**

```
UNIQUE(telNo)
```

IEF - Referential Integrity

- **FK is column or set of columns that links each row in child table containing foreign FK to row of parent table containing matching PK**
- **Referential integrity means that, if FK contains value, that value must refer to existing row in parent table**
- **ISO standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE:**

```
FOREIGN KEY(branchNo) REFERENCES Branch
```

Pearson Education © 2009

IEF - Referential Integrity

- **Any INSERT/UPDATE attempting to create FK value in child table without matching CK value in parent is rejected**
- **Action taken attempting to update/delete CK value in parent table with matching rows in child is dependent on <u>referential action</u> specified using ON UPDATE and ON DELETE subclauses:**
  - **CASCADE** - **SET NULL**
  - **SET DEFAULT** - **NO ACTION**

Pearson Education © 2009

IEF - Referential Integrity

CASCADE: Delete row from parent and delete matching rows in child, in cascading manner

SET NULL: Delete row from parent and set FK column(s) in child to NULL

Only valid if FK columns are NOT NULL

SET DEFAULT: Delete row from parent and set each component of FK in child to specified default

Only valid if DEFAULT specified for FK columns

NO ACTION: Reject delete from parent - Default

Pearson Education © 2009

IEF - Referential Integrity

FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL

FOREIGN KEY (ownerNo) REFERENCES Owner ON UPDATE CASCADE

Pearson Education © 2009

IEF - General Constraints

- **Could use CHECK/UNIQUE in CREATE and ALTER TABLE**
- **Similar to CHECK clause:**

```
CREATE ASSERTION AssertionName
CHECK (searchCondition)
```

Pearson Education © 2009

IEF - General Constraints

```
CREATE ASSERTION StaffNotHandlingTooMuch
CHECK (NOT EXISTS          (SELECT staffNo
                            FROM PropertyForRent
                            GROUP BY staffNo
                            HAVING COUNT(*) > 100))
```

Pearson Education © 2009

Data Definition

- **SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed**
- **Main SQL DDL statements:**

CREATE SCHEMA	**DROP SCHEMA**
CREATE/ALTER DOMAIN	**DROP DOMAIN**
CREATE/ALTER TABLE	**DROP TABLE**
CREATE VIEW	**DROP VIEW**

- **Many DBMSs also provide:**

CREATE INDEX DROP INDEX

Pearson Education © 2009

Data Definition

- Relations and other database objects exist in an *environment*
- Each environment contains one or more *catalogs*, and each catalog consists of set of schemas
- Schema is named collection of related database objects
- Objects in schema can be tables, views, domains, assertions
    - All have same owner

CREATE SCHEMA

```
CREATE SCHEMA [Name |
                    AUTHORIZATION CreatorId ]
DROP SCHEMA Name [RESTRICT | CASCADE ]
```

- **With RESTRICT (default)**
  - **Schema must be empty or operation fails**
- **With CASCADE**
  - **Operation cascades to drop all objects associated with schema in order defined above**
  - **If any operations fail → DROP SCHEMA fails**

Pearson Education © 2009

CREATE TABLE

```
CREATE TABLE TableName
{(colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] […,]}
{[FOREIGN KEY (listOfFKColumns)
  REFERENCES ParentTableName [(listOfCKColumns)],
  [ON UPDATE referentialAction]
  [ON DELETE referentialAction ]] [,…]}
 {[CHECK (searchCondition)] [,…] })
```

Pearson Education © 2009

CREATE TABLE

- Creates table with one or more columns of specified *dataType*
- With NOT NULL
  - System rejects any attempt to insert null in column
- Can specify DEFAULT value for column
- Primary keys should always be specified as NOT NULL
- FOREIGN KEY clause specifies FK along with referential action

Pearson Education © 2009

Example 7.1 - CREATE TABLE

```
CREATE DOMAIN OwnerNumber AS VARCHAR(5)
  CHECK (VALUE IN (SELECT ownerNo FROM PrivateOwner));
CREATE DOMAIN StaffNumber AS VARCHAR(5)
  CHECK (VALUE IN (SELECT staffNo FROM Staff));
CREATE DOMAIN PNumber AS VARCHAR(5);
CREATE DOMAIN PRooms AS SMALLINT;
        CHECK(VALUE BETWEEN 1 AND 15);
CREATE DOMAIN PRent AS DECIMAL(6,2)
        CHECK(VALUE BETWEEN 0 AND 9999.99);
```

Pearson Education © 2009

Example 7.1 - CREATE TABLE

```
CREATE TABLE PropertyForRent (
   propertyNo   PNumber          NOT NULL, ….
   rooms        PRooms           NOT NULL   DEFAULT 4,
   rent         PRent          NOT NULL, DEFAULT 600,
   ownerNo      OwnerNumber                NOT NULL,
   staffNo      StaffNumber
                Constraint StaffNotHandlingTooMuch ….
   branchNo     BranchNumber           NOT NULL,
   PRIMARY KEY (propertyNo),
   FOREIGN KEY (staffNo) REFERENCES Staff
    ON DELETE SET NULL ON UPDATE CASCADE ….);
```

Pearson Education © 2009

ALTER TABLE

- Add new column
- Drop column
- Add new table constraint
- Drop table constraint
- Set default for column
- Drop default for column

Pearson Education © 2009

Example 7.2(a) - ALTER TABLE

Change Staff table by removing default of ‘Assistant’ for position column and setting default for sex column to female (‘F’).

```
ALTER TABLE Staff
        ALTER position DROP DEFAULT;
ALTER TABLE Staff
        ALTER sex SET DEFAULT ‘F’;
```

Pearson Education © 2009

Example 7.2(b) - ALTER TABLE

Remove constraint from PropertyForRent that staff are not allowed to handle more than 100 properties at a time. Add new column to Client table.

```
ALTER TABLE PropertyForRent
    DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client
    ADD prefNoRooms PRooms;
```

Pearson Education © 2009

DROP TABLE

```
DROP TABLE TableName [RESTRICT | CASCADE]

        e.g.        DROP TABLE PropertyForRent;
```

- Removes named table and all rows
- With RESTRICT
  - If any other objects depend for their existence on continued existence of this table → SQL does not allow request
- With CASCADE
  - SQL drops all dependent objects (and objects dependent on these objects)

Views

View

Dynamic result of one or more relational operations operating on base relations to produce another relation

- Virtual relation that does not necessarily actually exist in database but is produced upon request, at time of request

Views

- **Contents of a view are defined as query on one or more base relations**
- **View resolution**
  - **Any operations on view automatically translated into operations on relations from which derived**
- **View materialization**
  - **View stored as temporary table**
  - **Maintained as underlying base tables are updated**

SQL - CREATE VIEW

```
CREATE VIEW ViewName [ (newColumnName [,...]) ]
        AS subselect
        [WITH [CASCADED | LOCAL] CHECK OPTION]
```

- **Can assign name to each column in view**
- **If list of column names specified**
  - **Must have same number of items as number of columns produced by *subselect***
- **If omitted**
  - **Each column takes name of corresponding column in *subselect***

Pearson Education © 2009

SQL - CREATE VIEW

- **List must be specified if any ambiguity in column name**
- ***Subselect* known as <u>defining query</u>**
- **WITH CHECK OPTION**
  - **Ensures if row fails to satisfy WHERE clause of defining query - not added to underlying base table**
- **Need SELECT privilege on all tables referenced in subselect**
- **Need USAGE privilege on any domains used in referenced columns**

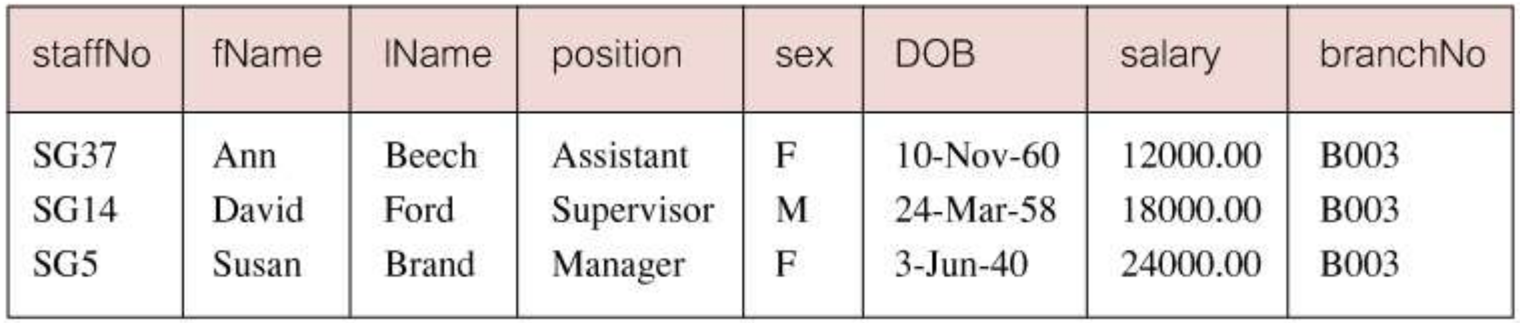
Pearson Education © 2009

Example 7.3 - Create Horizontal View

Create view so that manager at branch B003 can only see details for staff who work in his or her office.

```
CREATE VIEW Manager3Staff
AS      SELECT *
        FROM Staff
        WHERE branchNo = ‘B003’;
```

Table 6.3 Data for view Manager3Staff.



staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

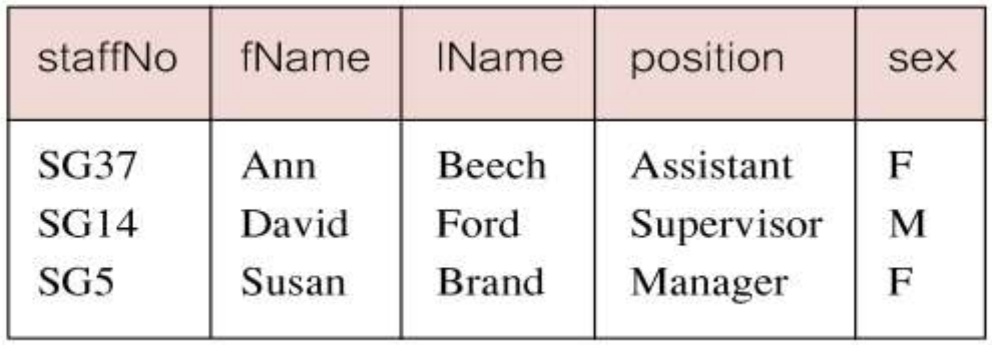
Pearson Education © 2009

Example 7.4 - Create Vertical View

Create view of staff details at branch B003 excluding salaries.

```
 CREATE VIEW Staff3
AS  SELECT staffNo, fName, lName, position, sex
    FROM Staff
    WHERE branchNo = ‘B003’;
```

Table 6.4 Data for view Staff3.



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

Pearson Education © 2009

Example 7.5 - Grouped and Joined Views

Create view of staff who manage properties for rent, including branch number they work at, staff number, and number of properties they manage.

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
 AS SELECT s.branchNo, s.staffNo, COUNT(*)
      FROM Staff s, PropertyForRent p
      WHERE s.staffNo = p.staffNo
      GROUP BY s.branchNo, s.staffNo;
```

Pearson Education © 2009

Example 7.3 - Grouped and Joined Views

Table 6.5 Data for view StaffPropCnt.

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

Pearson Education © 2009

SQL - DROP VIEW

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

- **Causes definition of view to be deleted from database**
- **For example:**

```
DROP VIEW Manager3Staff;
```

SQL - DROP VIEW

- With CASCADE
  - All related dependent objects deleted; i.e. any views defined on view being dropped.
- With RESTRICT (default)
  - If any other objects depend for existence on continued existence of view being dropped → command rejected

View Resolution

Count number of properties managed by each member at branch B003.

```
SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```

View Resolution

(a) View column names in SELECT list are translated into corresponding column names in defining query:

```
SELECT s.staffNo As staffNo, COUNT(*) As cnt
```

(b) View names in FROM replaced with corresponding FROM lists of defining query:

```
FROM Staff s, PropertyForRent p
```

View Resolution

(c) WHERE from user query combined with WHERE of defining query using AND:

```
WHERE s.staffNo = p.staffNo AND branchNo = 'B003'
```

(d) GROUP BY and HAVING clauses copied from defining query:

```
GROUP BY s.branchNo, s.staffNo
```

(e) ORDER BY copied from query with view column name translated into defining query column name

```
ORDER BY s.staffNo
```

View Resolution

(f) Final merged query executed to produce result:

```
SELECT s.staffNo AS staffNo, COUNT(*) AS cnt
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo AND
              branchNo = ‘B003’
GROUP BY s.branchNo, s.staffNo
ORDER BY s.staffNo;
```

Pearson Education © 2009

Restrictions on Views

SQL imposes several restrictions on creation and use of views.

(a) If column in view based on aggregate function:

- Column may appear only in SELECT and ORDER BY clauses of queries that access view
- Column may not be used in WHERE nor be an argument to aggregate function in any query based on view

Restrictions on Views

- For example, following queries would fail:

```
SELECT COUNT(cnt)
FROM StaffPropCnt;

SELECT *
FROM StaffPropCnt
WHERE cnt > 2;
```

Restrictions on Views

(b) Grouped view may never be joined with base table or view

- For example
  - StaffPropCnt view is grouped view, any attempt to join this view with another table or view fails

Pearson Education © 2009

View Updatability

- All updates to base table reflected in all views that encompass base table
- May expect that if view updated then base table(s) will reflect change

Pearson Education © 2009

View Updatability

- Consider again view StaffPropCnt
- If we tried to insert record showing that at branch B003, SG5 manages 2 properties:

```
INSERT INTO StaffPropCnt
VALUES ('B003', 'SG5', 2);
```

- Have to insert 2 records into PropertyForRent showing which properties SG5 manages. However, do not know which properties they are; i.e. do not know primary keys!

View Updatability

- **If change definition of view and replace count with actual property numbers:**

```
CREATE VIEW StaffPropList (branchNo,
                         staffNo, propertyNo)
AS SELECT s.branchNo, s.staffNo, p.propertyNo
     FROM Staff s, PropertyForRent p
     WHERE s.staffNo = p.staffNo;
```

Pearson Education © 2009

View Updatability

- **Now try to insert the record:**

```
INSERT INTO StaffPropList
VALUES ('B003', 'SG5', 'PG19');
```

- **Still problem - in PropertyForRent all columns except postcode/staffNo are not allowed nulls**
- **No way of giving remaining non-null columns values**

View Updatability

- **ISO specifies that view is updatable if and only if:**

  - DISTINCT is not specified

  - Every element in SELECT list of defining query is column name and no column appears more than once

  - FROM clause specifies only one table

    - **If source table a view – same conditions apply, excludes any views based on join, union, intersection or difference**

  - No nested SELECT referencing outer table

  - No GROUP BY or HAVING clause

  - Every row added through view must not violate integrity constraints of base table

Updatable View

For view to be updatable, DBMS must be able to trace any row or column back to its row or column in source table

Pearson Education © 2009

WITH CHECK OPTION

- Rows exist in view because they satisfy WHERE condition of defining query
- If row changes and no longer satisfies condition - disappears from view
- New rows appear within view when insert/update on view cause them to satisfy WHERE condition
- Rows that enter or leave view called *migrating rows*
- WITH CHECK OPTION generally prohibits row migrating out of view

WITH CHECK OPTION

- LOCAL/CASCADED apply to view hierarchies
- With LOCAL
  - Any row insert/update on view and any view directly or indirectly defined on this view must not cause row to disappear from view unless row also disappears from derived view/table
- With CASCADED (default)
  - Any row insert/ update on view and on any view directly or indirectly defined on this view must not cause row to disappear from the view

Pearson Education © 2009

Example 7.6 - WITH CHECK OPTION

```
CREATE VIEW Manager3Staff
AS        SELECT *
          FROM Staff
          WHERE branchNo = ‘B003’
WITH CHECK OPTION;
```

- **Cannot update branch number of row B003 to B002 - would cause row to migrate from view**
- **Cannot insert row into view with branch number that does not equal B003**

Example 7.6 - WITH CHECK OPTION

- **Consider the following:**

```
CREATE VIEW LowSalary
 AS  SELECT * FROM Staff WHERE salary > 9000;
CREATE VIEW HighSalary
 AS  SELECT * FROM LowSalary
     WHERE salary > 10000
 WITH LOCAL CHECK OPTION;
CREATE VIEW Manager3Staff
 AS  SELECT * FROM HighSalary
     WHERE branchNo = ‘B003’;
```

Example 7.6 - WITH CHECK OPTION

```
UPDATE Manager3Staff
SET salary = 9500
WHERE staffNo = ‘SG37’;
```

- **This update would fail: although update would cause row to disappear from HighSalary, row would not disappear from LowSalary**
- **If update tried to set salary to 8000, update would succeed as row would no longer be part of LowSalary**

Pearson Education © 2009

Example 7.6 - WITH CHECK OPTION

- If HighSalary had specified WITH CASCADED CHECK OPTION, setting salary to 9500 or 8000 would be rejected because row would disappear from HighSalary
- To prevent anomalies like this
  - Each view should be created using WITH CASCADED CHECK OPTION

Pearson Education © 2009

Advantages of Views

- Data independence
- Currency
- Improved security
- Reduced complexity
- Convenience
- Customization
- Data integrity

Pearson Education © 2009

Disadvantages of Views

- **Update restriction**
- **Structure restriction**
- **Performance**

Pearson Education © 2009

View Materialization

- **View resolution mechanism may be slow, if view accessed frequently**
- **View materialization stores view as temporary table when view first queried**
- **Queries based on materialized view can be faster than recomputing view each time**
- **Difficulty in maintaining currency of view while base tables(s) updated**

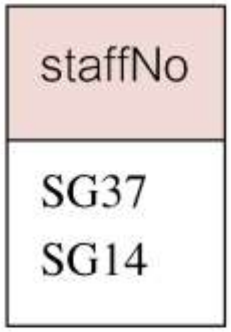
Pearson Education © 2009

View Maintenance

- **<u>View maintenance</u> aims to apply only those changes necessary to keep view current.**
- **Consider following view:**

```
CREATE VIEW StaffPropRent(staffNo)
AS  SELECT DISTINCT staffNo
    FROM PropertyForRent
    WHERE branchNo = ‘B003’ AND
                rent > 400;
```

Table 6.8 Data for view StaffPropRent.



staffNo
SG37
SG14

View Materialization

- If insert row into PropertyForRent with rent $\leq$400 then view would be unchanged
- If insert row for property PG24 at branch B003 with staffNo = SG19 and rent = 550, then row would appear in materialized view
- If insert row for property PG54 at branch B003 with staffNo = SG37 and rent = 450, then no new row would need to be added to materialized view
- If delete property PG24, row should be deleted from materialized view
- If delete property PG54, then row for PG37 should not be deleted (because of existing property PG21)

JOIN TYPES

Six types of JOINs:

1. JOIN or INNER JOIN
2. OUTER JOIN
   - 2.1 LEFT OUTER JOIN or LEFT JOIN
   - 2.2 RIGHT OUTER JOIN or RIGHT JOIN
   - 2.3 FULL OUTER JOIN or FULL JOIN
3. NATURAL JOIN
4. CROSS JOIN
5. SELF JOIN
6. JOINs based on Operators

1. JOIN or INNER JOIN

- We get all records that match the condition in both the tables
- Records in both the tables that do not match are not reported
- ONLY the matching entries in BOTH the tables SHOULD be listed
- JOIN without any other JOIN keywords (like OUTER, LEFT, etc) is an INNER JOIN

Examples:

```
selectdepartment_name, first_name from departments d inner
    join employees e on d.department_id = e.department_id;
```

OR

```
selectdepartment_name, first_name from departments d join
    employees e on d.department_id = e.department_id;
```

2. OUTER JOIN

- Retrieves either, the matched rows from one table and all rows in the other table Or, all rows in all tables
- There are three kinds:
  - 2.1 LEFT OUTER JOIN or LEFT JOIN
    - Returns all rows from the left table in conjunction with the matching rows from the right table
    - If there are no columns matching in the right table, it returns NULL values
  - 2.2 RIGHT OUTER JOIN or RIGHT JOIN
    - Returns all rows from the right table in conjunction with the matching rows from the left table
    - If there are no columns matching in the left table, it returns NULL values
  - 2.3 FULL OUTER JOIN or FULL JOIN
  - Combines LEFT OUTER JOIN and RIGHT OUTER JOIN
  - Returns row from either table when the conditions are met and returns NULL value when there is no match

EXAMPLES OF OUTER JOIN

- OUTER JOIN (full outer join)

Select *

FROM Table1 A FULL OUTER JOIN Table2 B OnA.Pk = B.Fk;

- LEFT JOIN

Select *

FROM Table1 A LEFT OUTER JOIN Table2 B OnA.Pk = B.Fk;

- RIGHT JOIN

Select *

FROM Table1 A RIGHT OUTER JOIN Table2 B OnA.Pk = B.Fk;

3. NATURAL JOIN

- A type of Inner join which is based on column having same name and same datatype present in both the tables to be joined
- Based on the two conditions :
  - JOIN is made on all the columns with the same name for equality
  - Removes duplicate columns from the result
- Examples:

selectdepartment_name, first_name from departments d
natural join employees e ;

SELECT *
from table-name1
NATURAL JOIN
table-name2;

4. CROSS JOIN

- Cartesian product of the two tables
- Result does not make sense in most of the situations

- Examples:

```
Select *
 FROM TableA CROSS JOIN TableB;
```

- OR

```
Select *
 FROM Table1 A1,Table1 A2;
```

5. SELF JOIN

- Not a different form of JOIN, rather it is a JOIN of a table to itself
- Examples

```
Select m.first_name manager, w.first_name worker
 From employees m inner join employees w
On m.employee_id = w.manager_id;
```

- OR

```
Select m.first_name manager, w.first_name worker
 From employees m , employees w
where m.employee_id = w.manager_id;
```

6. JOINs based on Operators

- Depending on the operator used for a JOIN clause, there can be two types of JOINs
  - Equi JOIN
    - For whatever JOIN type (INNER, OUTER, etc), if we use ONLY the equality operator (=), then we say that the JOIN is an EQUI JOIN
  - Theta or Non-Equi JOIN
    - Same as EQUI JOIN but allows all other operators like >, <, >= etc
- Examples:
- Equi join

```
select * from departments d, employees e where
    d.department_id = e.department_id;
```

- Theta or Non-Equi JOIN

```
select * from departments d, employees e where
    d.department_id <> e.department_id;
```