Chapter 5

SQL: Data Manipulation
Chapter 6 - Objectives

◆ Describe the purpose and importance of SQL.
◆ Explain how to use SQL to:
  – Create the database relation structures;
  – Perform data retrieval, insertion, modification and deletion from relations;
  – Perform simple and complex queries.
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.
Literals

- Most components of an SQL statement are case insensitive, except for literal character data.
- 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]
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.

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

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>30000.00</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>12000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>18000.00</td>
</tr>
<tr>
<td>SA9</td>
<td>Mary</td>
<td>Howe</td>
<td>9000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>24000.00</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>9000.00</td>
</tr>
</tbody>
</table>
Example 6.3 Use of DISTINCT

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

```
SELECT propertyNo
FROM Viewing;
```

<table>
<thead>
<tr>
<th>propertyNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA14</td>
</tr>
<tr>
<td>PG4</td>
</tr>
<tr>
<td>PG4</td>
</tr>
<tr>
<td>PA14</td>
</tr>
<tr>
<td>PG36</td>
</tr>
</tbody>
</table>
Example 6.3 Use of DISTINCT

- Use DISTINCT to eliminate duplicates:

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

<table>
<thead>
<tr>
<th>propertyNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA14</td>
</tr>
<tr>
<td>PG4</td>
</tr>
<tr>
<td>PG36</td>
</tr>
</tbody>
</table>
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
FROM Staff;

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>col4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>2500.00</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>1000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>1500.00</td>
</tr>
<tr>
<td>SA9</td>
<td>Mary</td>
<td>Howe</td>
<td>750.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>2000.00</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>750.00</td>
</tr>
</tbody>
</table>
Example 6.4  Use of Column Alias

To name column, use AS clause:

```sql
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
WHERE salary > 10000;

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>30000.00</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>Assistant</td>
<td>12000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Supervisor</td>
<td>18000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
<td>24000.00</td>
</tr>
</tbody>
</table>
Example 6.6  Compound Comparison 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.

<table>
<thead>
<tr>
<th>branchNo</th>
<th>street</th>
<th>city</th>
<th>postcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>B005</td>
<td>22 Deer Rd</td>
<td>London</td>
<td>SW1 4EH</td>
</tr>
<tr>
<td>B003</td>
<td>163 Main St</td>
<td>Glasgow</td>
<td>G11 9QX</td>
</tr>
<tr>
<td>B002</td>
<td>56 Clover Dr</td>
<td>London</td>
<td>NW10 6EU</td>
</tr>
</tbody>
</table>
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 the endpoints of range.
Example 6.7 Range Search Condition

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>30000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
<td>24000.00</td>
</tr>
</tbody>
</table>

Table 5.7 Result table for Example 5.7.
Example 6.7 Range Search Condition with AND

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

```sql
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary >= 20000 AND salary <= 30000;
```

- Useful, though, for a 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>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Supervisor</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
</tr>
</tbody>
</table>
Example 6.8  Set Membership

- **IN does not add much** to SQL’s expressive power. Could have expressed this as:

  ```sql
  SELECT staffNo, fName, lName, position
  FROM Staff
  WHERE position='Manager' OR
       position='Supervisor';
  ```

- **IN is more efficient** when set contains many values.
- **There is a negated version** (NOT IN).
Example 6.9 Pattern Matching

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

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

Table 5.9 Result table for Example 5.9.

<table>
<thead>
<tr>
<th>ownerNo</th>
<th>fName</th>
<th>lName</th>
<th>address</th>
<th>telNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO87</td>
<td>Carol</td>
<td>Farrel</td>
<td>6 Achray St, Glasgow G32 9DX</td>
<td>0141-357-7419</td>
</tr>
<tr>
<td>CO40</td>
<td>Tina</td>
<td>Murphy</td>
<td>63 Well St, Glasgow G42</td>
<td>0141-943-1728</td>
</tr>
<tr>
<td>CO93</td>
<td>Tony</td>
<td>Shaw</td>
<td>12 Park Pl, Glasgow G4 0QR</td>
<td>0141-225-7025</td>
</tr>
</tbody>
</table>
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 a 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

<table>
<thead>
<tr>
<th>clientNo</th>
<th>viewDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR56</td>
<td>26-May-04</td>
</tr>
</tbody>
</table>

- **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.

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>30000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>24000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>18000.00</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>12000.00</td>
</tr>
<tr>
<td>SA9</td>
<td>Mary</td>
<td>Howe</td>
<td>9000.00</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>9000.00</td>
</tr>
</tbody>
</table>
Example 6.12  Multiple Column Ordering

- Four flats in `PropertyForRent` table - if no minor sort key specified, system arranges these rows in any order it chooses.
- To arrange in order of rent, specify `minor order`:

```sql
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.

<table>
<thead>
<tr>
<th>propertyNo</th>
<th>type</th>
<th>rooms</th>
<th>rent</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG16</td>
<td>Flat</td>
<td>4</td>
<td>450</td>
</tr>
<tr>
<td>PL94</td>
<td>Flat</td>
<td>4</td>
<td>400</td>
</tr>
<tr>
<td>PG36</td>
<td>Flat</td>
<td>3</td>
<td>375</td>
</tr>
<tr>
<td>PG4</td>
<td>Flat</td>
<td>3</td>
<td>350</td>
</tr>
<tr>
<td>PA14</td>
<td>House</td>
<td>6</td>
<td>650</td>
</tr>
<tr>
<td>PG21</td>
<td>House</td>
<td>5</td>
<td>600</td>
</tr>
</tbody>
</table>
SELECT Statement – Aggregates Functions

- 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.
SELECT Statement - Aggregates

- Aggregate functions can be used only in SELECT list and in HAVING clause.

- SELECT clause cannot list a single column with an aggregate function without a GROUP BY clause.

- For example, the following is illegal:

  ```sql
  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;
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-04’
AND ‘31-May-04’;

<table>
<thead>
<tr>
<th>myCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
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’;

<table>
<thead>
<tr>
<th>myCount</th>
<th>mySum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>54000.00</td>
</tr>
</tbody>
</table>
Example 6.16 Use of MIN, MAX, AVG

Find **minimum, maximum, and average staff salary**.

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

<table>
<thead>
<tr>
<th>myMin</th>
<th>myMax</th>
<th>myAvg</th>
</tr>
</thead>
<tbody>
<tr>
<td>9000.00</td>
<td>30000.00</td>
<td>17000.00</td>
</tr>
</tbody>
</table>
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 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 6.17 Use of GROUP BY

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

```sql
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

<table>
<thead>
<tr>
<th>branchNo</th>
<th>myCount</th>
<th>mySum</th>
</tr>
</thead>
<tbody>
<tr>
<td>B003</td>
<td>3</td>
<td>54000.00</td>
</tr>
<tr>
<td>B005</td>
<td>2</td>
<td>39000.00</td>
</tr>
<tr>
<td>B007</td>
<td>1</td>
<td>9000.00</td>
</tr>
</tbody>
</table>
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 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.

```sql
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

<table>
<thead>
<tr>
<th>branchNo</th>
<th>myCount</th>
<th>mySum</th>
</tr>
</thead>
<tbody>
<tr>
<td>B003</td>
<td>3</td>
<td>54000.00</td>
</tr>
<tr>
<td>B005</td>
<td>2</td>
<td>39000.00</td>
</tr>
</tbody>
</table>
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 6.19  Subquery with Equality

List staff who work in branch ‘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

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>Assistant</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Supervisor</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
</tr>
</tbody>
</table>

Table 5.19  Result table for Example 5.19.
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;
```
### Table 5.20  Result table for Example 5.20.

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>position</th>
<th>salDiff</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>13000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Supervisor</td>
<td>1000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
<td>7000.00</td>
</tr>
</tbody>
</table>
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**.
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’));
Example 6.21  Nested subquery: use of IN

Table 5.21  Result table for Example 5.21.

<table>
<thead>
<tr>
<th>propertyNo</th>
<th>street</th>
<th>city</th>
<th>postcode</th>
<th>type</th>
<th>rooms</th>
<th>rent</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG16</td>
<td>5 Novar Dr</td>
<td>Glasgow</td>
<td>G12 9AX</td>
<td>Flat</td>
<td>4</td>
<td>450</td>
</tr>
<tr>
<td>PG36</td>
<td>2 Manor Rd</td>
<td>Glasgow</td>
<td>G32 4QX</td>
<td>Flat</td>
<td>3</td>
<td>375</td>
</tr>
<tr>
<td>PG21</td>
<td>18 Dale Rd</td>
<td>Glasgow</td>
<td>G12</td>
<td>House</td>
<td>5</td>
<td>600</td>
</tr>
</tbody>
</table>
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 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 of the values in this set.

Table 5.22 Result table for Example 5.22.

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>30000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Supervisor</td>
<td>18000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
<td>24000.00</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>30000.00</td>
</tr>
</tbody>
</table>
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).
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 the clientNo columns (c.clientNo = v.clientNo) are included in the result.

- Equivalent to equi-join in relational algebra.

Table 5.24 Result table for Example 5.24.

<table>
<thead>
<tr>
<th>clientNo</th>
<th>fName</th>
<th>IName</th>
<th>propertyNo</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR56</td>
<td>Aline</td>
<td>Stewart</td>
<td>PG36</td>
<td></td>
</tr>
<tr>
<td>CR56</td>
<td>Aline</td>
<td>Stewart</td>
<td>PA14</td>
<td>too small</td>
</tr>
<tr>
<td>CR56</td>
<td>Aline</td>
<td>Stewart</td>
<td>PG4</td>
<td></td>
</tr>
<tr>
<td>CR62</td>
<td>Mary</td>
<td>Tregear</td>
<td>PA14</td>
<td>no dining room</td>
</tr>
<tr>
<td>CR76</td>
<td>John</td>
<td>Kay</td>
<td>PG4</td>
<td>too remote</td>
</tr>
</tbody>
</table>
Alternative JOIN Constructs

- SQL provides **alternative ways** to specify joins:

  ```sql
  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 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.

<table>
<thead>
<tr>
<th>branchNo</th>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>propertyNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>B003</td>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>PG16</td>
</tr>
<tr>
<td>B003</td>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>PG21</td>
</tr>
<tr>
<td>B003</td>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>PG36</td>
</tr>
<tr>
<td>B005</td>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>PL94</td>
</tr>
<tr>
<td>B007</td>
<td>SA9</td>
<td>Mary</td>
<td>Howe</td>
<td>PA14</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>branchNo</th>
<th>city</th>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>propertyNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>PG16</td>
</tr>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>PG21</td>
</tr>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>PG36</td>
</tr>
<tr>
<td>B005</td>
<td>London</td>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>PL94</td>
</tr>
<tr>
<td>B007</td>
<td>Aberdeen</td>
<td>SA9</td>
<td>Mary</td>
<td>Howe</td>
<td>PA14</td>
</tr>
</tbody>
</table>

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**.

```
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

<table>
<thead>
<tr>
<th>branchNo</th>
<th>staffNo</th>
<th>myCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>B003</td>
<td>SG14</td>
<td>1</td>
</tr>
<tr>
<td>B003</td>
<td>SG37</td>
<td>2</td>
</tr>
<tr>
<td>B005</td>
<td>SL41</td>
<td>1</td>
</tr>
<tr>
<td>B007</td>
<td>SA9</td>
<td>1</td>
</tr>
</tbody>
</table>
Outer Joins

- **Normally** 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.**

<table>
<thead>
<tr>
<th>Branch1</th>
<th>PropertyForRent1</th>
</tr>
</thead>
<tbody>
<tr>
<td>branchNo</td>
<td>propertyNo</td>
</tr>
<tr>
<td>B003</td>
<td>PA14</td>
</tr>
<tr>
<td>B004</td>
<td>PL94</td>
</tr>
<tr>
<td>B002</td>
<td>PG4</td>
</tr>
</tbody>
</table>
Outer Joins

◆ The (inner) join of these two tables:

\[
\text{SELECT } b.*, p.* \\
\text{FROM Branch1 } b, \text{ PropertyForRent1 } p \\
\text{WHERE } b.bCity = p.pCity;
\]

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

<table>
<thead>
<tr>
<th>branchNo</th>
<th>bCity</th>
<th>propertyNo</th>
<th>pCity</th>
</tr>
</thead>
<tbody>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B002</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>
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 those rows of first (left) table unmatched with rows from second (right) table.
- Columns from second table are filled with NULLs.

Table 5.28 Result table for Example 5.28.

<table>
<thead>
<tr>
<th>branchNo</th>
<th>bCity</th>
<th>propertyNo</th>
<th>pCity</th>
</tr>
</thead>
<tbody>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B004</td>
<td>Bristol</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>B002</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>
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 those rows of second (right) table that are unmatched with rows from first (left) table.

- **Columns from first table are filled with NULLs.**

<table>
<thead>
<tr>
<th>branchNo</th>
<th>bCity</th>
<th>propertyNo</th>
<th>pCity</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>PA14</td>
<td>Aberdeen</td>
</tr>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B002</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>
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 that are unmatched in both tables.
- Unmatched columns are filled with NULLs.

<table>
<thead>
<tr>
<th>branchNo</th>
<th>bCity</th>
<th>propertyNo</th>
<th>pCity</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>PA14</td>
<td>Aberdeen</td>
</tr>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B004</td>
<td>Bristol</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>B002</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>
EXISTS and NOT EXISTS Correlated Queries

- **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**.
Example 6.31  Query using EXISTS

Find all staff who work in 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.

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>Assistant</td>
</tr>
</tbody>
</table>
Example 6.31 Query using EXISTS

- Note, search condition \texttt{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:
  
  \[
  \text{SELECT } * \text{ FROM Branch WHERE city='London'}
  \]

- would always be true and query would be:
  
  \[
  \text{SELECT staffNo, fName, lName, position FROM Staff WHERE true;}
  \]
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';
```
**INSERT**

**INSERT INTO** Table<b>Name** [ (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.
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)

◆ Or

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

```sql
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(staffNo, 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));

Pearson Education © 2009
Example 6.37 INSERT … SELECT

Table 5.35 Result table for Example 5.37.

<table>
<thead>
<tr>
<th>staffNo</th>
<th>fName</th>
<th>lName</th>
<th>propCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>1</td>
</tr>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>0</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>2</td>
</tr>
<tr>
<td>SA9</td>
<td>Mary</td>
<td>Howe</td>
<td>1</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>0</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>1</td>
</tr>
</tbody>
</table>

◆ 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]
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 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 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 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;