Objectives

- Purpose and importance of SQL.
- How to retrieve data from database using SELECT and:
  - Use compound WHERE conditions.
  - Sort query results using ORDER BY.
  - Use aggregate functions.
  - Group data using GROUP BY and HAVING.
  - Use subqueries.
Objectives

– Join tables together.
– Perform set operations (UNION, INTERSECT, EXCEPT).

◆ How to update database using INSERT, UPDATE, and DELETE.
Objectives of SQL

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

- **SQL** is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.

- **Until SQL:1999**, SQL did not contain flow of control commands. These had to be implemented using a programming or job-control language, or interactively by the decisions of user.
Objectives of SQL

SQL is relatively easy to learn:

- it is non-procedural - you specify what information you require, rather than how to get it;
- it is essentially free-format.
Objectives of SQL

◆ Consists of standard English words:

1) **CREATE TABLE** Staff(staffNo VARCHAR(5), lName VARCHAR(15), salary DECIMAL(7,2));

2) **INSERT INTO** Staff
   **VALUES** (‘SG16’, ‘Brown’, 8300);

3) **SELECT** staffNo, lName, salary
   **FROM** Staff
   **WHERE** salary > 10000;
Objectives of SQL

◆ Consists of standard English words:

4)  **UPDATE** Staff
    SET salary = salary * 1.03;

5)  **DELETE FROM** Viewing
    **WHERE** propertyNo = ‘PG4’;
Objectives of SQL

◆ Can be used by range of users including DBAs, management, application developers, and other types of end users.

◆ An ISO standard now exists for SQL, making it both the formal and de facto standard language for relational databases.
History of SQL

◆ In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called ‘Structured English Query Language’ (SEQUEL).

◆ A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.
History of SQL

◆ Still pronounced ‘see-quel’, though official pronunciation is ‘S-Q-L’.

◆ IBM subsequently produced a prototype DBMS called System R, based on SEQUEL/2.

◆ Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.
History of SQL

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

- SQL has become part of application architectures such as IBM’s Systems Application Architecture.
- It is strategic choice of many large and influential organizations (e.g. X/OPEN).
- SQL is Federal Information Processing Standard (FIPS) to which conformance is required for all sales of databases to American Government.
Importance of SQL

- SQL is used in other standards and even influences development of other standards as a definitional tool. Examples include:
Writing SQL Commands

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

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

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

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

◆ More readable with indentation and lineation:
  – Each clause should begin on a new line.
  – Start of a clause should line up with start of other clauses.
  – If clause has several parts, should each appear on a separate line and be indented under start of clause.
Writing SQL Commands

◆ Use extended form of BNF notation:

- Upper-case letters represent reserved words.
- Lower-case letters represent user-defined words.
- | indicates a choice among alternatives.
- Curly braces indicate a required element.
- Square brackets indicate an optional element.
- … indicates optional repetition (0 or more).
Literals

◆ Literals are constants used in SQL statements.

◆ All non-numeric literals must be enclosed in single quotes (e.g. ‘London’).

◆ All numeric literals must not be enclosed in quotes (e.g. 650.00).
SELECT Statement

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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>Specifies table(s) to be used.</td>
</tr>
<tr>
<td>WHERE</td>
<td>Filters rows.</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>Forms groups of rows with same column value.</td>
</tr>
<tr>
<td>HAVING</td>
<td>Filters groups subject to some condition.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Specifies which columns are to appear in output.</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>Specifies the order of the output.</td>
</tr>
</tbody>
</table>

© Pearson Education Limited 1995, 2005
SELECT Statement

- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.
Example 5.1 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 5.1 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 5.2 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 5.2 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 5.3 Use of DISTINCT

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

```
SELECT propertyNo
FROM Viewing;
```
Example 5.3  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 5.4  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 5.4  Calculated Fields

◆ To name column, use AS clause:

```
SELECT staffNo, fName, lName, salary/12 AS monthlySalary
FROM Staff;
```
Example 5.5 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 5.5 Result table for Example 5.5.

<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>

Adopted from "Database Systems: A Practical Approach to Design, Implementation, and Management", Thomas Connolly and Carolyn Begg,
Example 5.6 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.

<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>

Adopted from "Database Systems: A Practical Approach to Design, Implementation, and Management", Thomas Connolly and Carolyn Begg,
Example 5.7  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 5.7 Range Search Condition

Table 5.7 Result table for Example 5.7.

<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>
Example 5.7 Range Search Condition

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

\[
\text{SELECT staffNo, fName, lName, position, salary}
\text{FROM Staff}
\text{WHERE salary} \geq 20000 \text{ AND salary} \leq 30000;
\]

◆ Useful, though, for a range of values.
Example 5.8 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 5.8 Set Membership

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

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

◆ IN is more efficient when set contains many values.
Example 5.9 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 5.9 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 5.10 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:

```sql
SELECT clientNo, viewDate
FROM Viewing
WHERE propertyNo = 'PG4' AND comment IS NULL;
```
Example 5.10  NULL Search Condition

Negated version (IS NOT NULL) can test for non-null values.
Example 5.11 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 5.11  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 5.12 Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type;
```
Example 5.12  Multiple Column Ordering

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

<table>
<thead>
<tr>
<th>propertyNo</th>
<th>type</th>
<th>rooms</th>
<th>rent</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL94</td>
<td>Flat</td>
<td>4</td>
<td>400</td>
</tr>
<tr>
<td>PG4</td>
<td>Flat</td>
<td>3</td>
<td>350</td>
</tr>
<tr>
<td>PG36</td>
<td>Flat</td>
<td>3</td>
<td>375</td>
</tr>
<tr>
<td>PG16</td>
<td>Flat</td>
<td>4</td>
<td>450</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>
Example 5.12  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 5.12  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

◆ 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

- `COUNT(*)` counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use `DISTINCT` before column name to eliminate duplicates.
- `DISTINCT` has no effect with `MIN/MAX`, but may have with `SUM/AVG`.

Adopted from "Database Systems: A Practical Approach to Design, Implementation, and Management", Thomas Connolly and Carolyn Begg,
SELECT Statement - Aggregates

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

◆ If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference a column out with an aggregate function. For example, the following is illegal:

```
SELECT staffNo, COUNT(salary)
FROM Staff;
```
Example 5.13  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 5.14 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 5.15 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 5.16  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;

<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>

Adopted from "Database Systems: A Practical Approach to Design, Implementation, and Management", Thomas Connolly and Carolyn Begg,
SELECT Statement - Grouping

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

◆ All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.

◆ If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.

◆ ISO considers two nulls to be equal for purposes of GROUP BY.
Example 5.17 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 5.17 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 5.18 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 5.18 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 5.19  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 5.19  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 5.19  Subquery with Equality

Table 5.19  Result table for Example 5.19.

<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>

Example 5.20 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 5.20  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 5.20 Subquery with Aggregate

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.
Subquery Rules

◆ When subquery is an operand in a comparison, subquery must appear on right-hand side.

◆ A subquery may not be used as an operand in an expression.
Example 5.21  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 5.21  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 5.22 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 5.22  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 5.23 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 5.23  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).
Multi-Table Queries

◆ Also possible to use an alias for a table named in FROM clause.

◆ Alias is separated from table name with a space.

◆ Alias can be used to qualify column names when there is ambiguity.
Example 5.24 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 5.24 Simple Join

◆ Only those rows from both tables that have identical values in the clientNo columns (c.clientNo = v.clientNo) are included in result.

◆ Equivalent to equi-join in relational algebra.

<table>
<thead>
<tr>
<th>clientNo</th>
<th>fName</th>
<th>lName</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>

Table 5.24 Result table for Example 5.24.
Alternative JOIN Constructs

- SQL provides alternative ways to specify joins:

  FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo
  FROM Client JOIN Viewing USING clientNo
  FROM Client NATURAL JOIN Viewing

- In each case, FROM replaces original FROM and WHERE. However, first produces table with two identical clientNo columns.
Example 5.25 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 5.25 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>

Adopted from "Database Systems: A Practical Approach to Design, Implementation, and Management", Thomas Connolly and Carolyn Begg,
Example 5.26  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 5.26  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 5.27 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 5.27  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>
Computing a Join

Procedure for generating results of a join are:

1. Form Cartesian product of the tables named in FROM clause.

2. If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.

3. For each remaining row, determine value of each item in SELECT list to produce a single row in result table.
Computing a Join

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

5. If there is an ORDER BY clause, sort result table as required.

◆ SQL provides special format of SELECT for Cartesian product:

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

◆ If one row of a joined table is unmatched, row is omitted from result table.

◆ Outer join operations retain rows that do not satisfy the join condition.

◆ Consider following tables:

<table>
<thead>
<tr>
<th>Branch1</th>
<th>PropertyForRent1</th>
</tr>
</thead>
<tbody>
<tr>
<td>branchNo</td>
<td>bCity</td>
</tr>
<tr>
<td>B003</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B004</td>
<td>Bristol</td>
</tr>
<tr>
<td>B002</td>
<td>London</td>
</tr>
</tbody>
</table>

Adopted from "Database Systems: A Practical Approach to Design, Implementation, and Management", Thomas Connolly and Carolyn Begg,
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.

<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>
Outer Joins

◆ Result table has two rows where cities are same.
◆ There are no rows corresponding to branches in Bristol and Aberdeen.
◆ To include unmatched rows in result table, use an Outer join.
Example 5.28  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 5.28  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 5.29  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 5.29  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 5.29  Result table for Example 5.29.

<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>

Adopted from "Database Systems: A Practical Approach to Design, Implementation, and Management", Thomas Connolly and Carolyn Begg,
Example 5.30 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 5.30  Full Outer Join

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

Table 5.30  Result table for Example 5.30.

<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

◆ EXIST S 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.
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 5.31  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 5.31  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 5.31 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 5.31 Query using EXISTS

Could also write this query using join construct:

```sql
SELECT staffNo, fName, lName, position
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND city = 'London';
```
Union, Intersect, and Difference (Except)

◆ Can use normal set operations of Union, Intersection, and Difference to combine results of two or more queries into a single result table.

◆ Union of two tables, A and B, is table containing all rows in either A or B or both.

◆ Intersection is table containing all rows common to both A and B.

◆ Difference is table containing all rows in A but not in B.

◆ Two tables must be *union compatible.*
Union, Intersect, and Difference (Except)

◆ Format of set operator clause in each case is:

\[ op \[ \text{ALL} \] \[ \text{CORRESPONDING} \[ \text{BY} \{ \text{column1 [, ...]} \} \] \]\n
◆ 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

Adopted from "Database Systems: A Practical Approach to Design, Implementation, and Management", Thomas Connolly and Carolyn Begg,
Example 5.32 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 5.32  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 5.32 Use of UNION

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

Table 5.32 Result table for Example 5.32.

<table>
<thead>
<tr>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>London</td>
</tr>
<tr>
<td>Glasgow</td>
</tr>
<tr>
<td>Aberdeen</td>
</tr>
<tr>
<td>Bristol</td>
</tr>
</tbody>
</table>
Example 5.33 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 5.33 Use of INTERSECT

Or

\[
\text{(SELECT * FROM Branch)} \ 
\text{INTERSECT CORRESPONDING BY city} 
\text{(SELECT * FROM PropertyForRent)};
\]

Table 5.33 Result table for Example 5.33.

<table>
<thead>
<tr>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aberdeen</td>
</tr>
<tr>
<td>Glasgow</td>
</tr>
<tr>
<td>London</td>
</tr>
</tbody>
</table>
Example 5.33 Use of INTERSECT

- Could rewrite this query without INTERSECT operator:

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

- Or:

  ```sql
  SELECT DISTINCT city FROM Branch b
  WHERE EXISTS
  (SELECT * FROM PropertyForRent p
  WHERE p.city = b.city);
  ```
Example 5.34 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.
Example 5.34 Use of EXCEPT

Could rewrite this query without EXCEPT:

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

Or

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

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

◆ columnList is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.

◆ Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.
**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 5.35  INSERT … VALUES

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

INSERT INTO Staff
Example 5.36  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
```
INSERT ... SELECT

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

```sql
INSERT INTO TableName [(columnNameList)] SELECT ...
```
Example 5.37 INSERT … SELECT

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

\[
\text{StaffPropCount(staffNo, fName, lName, propCnt)}
\]

Populate StaffPropCount using Staff and PropertyForRent tables.
Example 5.37 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 5.37  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.

Adopted from "Database Systems: A Practical Approach to Design, Implementation, and Management", Thomas Connoily and Carolyn Begg,
UPDATE

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

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

- **WHERE clause** is optional:
  - if omitted, named columns are updated for all rows in table;
  - if specified, only those rows that satisfy \textit{searchCondition} are updated.

- **New \textit{dataValue(s)}** must be compatible with data type for corresponding column.
Example 5.38/39 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 5.40 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 5.41/42  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;