5

SQL: Data Manipulation

Prerequisites for This Section

- **Readings:**
  - **Required:** Connolly and Begg, section 5.3
- **Assessments:**
  - Multiple-Choice Quiz 3

Section Objectives

In this section you will learn:

1. How to retrieve data from database using SELECT and clauses:
   ① use compound WHERE conditions.
   ② sort query results using ORDER BY.
   ③ use aggregate functions.
   ④ group data using GROUP BY and HAVING.
   ⑤ use subqueries.
   ⑥ join tables together.
   ⑦ perform set operations (UNION, INTERSECT, EXCEPT).
2. How to update database using INSERT, UPDATE, and DELETE.

DreamHome Rental Database

The relational schema for part of DreamHome case study is:

- **Branch** (branchNo, street, city, postcode)
- **Staff** (staffNo, fName, lName, position, sex, DOB, salary, branchNo)
- **PropertyForRent** (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
- **Client** (clientNo, fName, lName, telNo, prefType, maxRent)
- **PrivateOwner** (ownerNo, fName, lName, address, telNo)
- **Viewing** (clientNo, propertyNo, viewDate, comment)
- **Registration** (clientNo, branchNo, staffNo, dateJoined)

Agenda

1. Simple Queries
2. Sorting Results
3. Aggregate Functions
4. Grouping Results
5. Nested Queries and Set Comparison
6. Multi-Table Queries
7. Combining Result Tables
8. Database Updates

SELECT Statement

- The statement syntax:

SELECT [DISTINCT | ALL]

{* | [columnExpression [**AS** newName]] [,...] }

FROM TableName [alias] [, ...]

[**WHERE** condition]

[**GROUP BY** columnList [**HAVING** condition]]

[**ORDER BY** columnList]

SELECT Statement

- The statement explanation:

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
SELECT	specifies which columns are to appear in output
ORDER BY	specifies the order of the output

- 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;
```

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 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;

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 5.3 Use of DISTINCT

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

  SELECT propertyNo

  FROM Viewing;

propertyNo
PA14
PG4
PG4
PA14
PG36

- Use **DISTINCT** to eliminate duplicates:

  SELECT DISTINCT propertyNo

  FROM Viewing;

propertyNo
PA14
PG4
PG36

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;

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

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

FROM Staff;

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 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.

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 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.

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

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

Example 5.8 Set Membership

- List all managers and supervisors.

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

- 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%';
```

- 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.

- Have to test for null explicitly using special keyword **IS NULL**:

```
SELECT clientNo, viewDate
FROM   Viewing
WHERE propertyNo = 'PG4' AND
      comment IS NULL;
```

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

Agenda

1. Simple Queries
2. Sorting Results
3. Aggregate Functions
4. Grouping Results
5. Nested Queries and Set Comparison
6. Multi-Table Queries
7. Combining Result Tables
8. Database Updates

ORDER BY Clause

- **ORDER BY** Clause syntax:

  [ORDER BY columnName [**ASC** | **DESC**] [, …]

- **ORDER BY** clause allows the retrieved rows to be ordered in ascending (**ASC**) or decending (**DESC**) order on any column or combination of columns.

- **ORDER BY** clause must always be the last clause of the **SELECT** statement.

- **ASC** is default.

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;
```

- Or

```
SELECT    staffNo, fName, lName, salary
FROM      Staff
ORDER BY  4 DESC;
```

Example 5.12 Multiple Column Ordering

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

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

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

Agenda

1. Simple Queries
2. Sorting Results
3. Aggregate Functions
4. Grouping Results
5. Nested Queries and Set Comparison
6. Multi-Table Queries
7. Combining Result Tables
8. Database Updates

Aggregate 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.

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

- Each operates on a single column of a table and returns a single value.

Aggregate Functions

- **SUM** and **AVG** may be used on numeric fields only.
- Apart from **COUNT(*)**, each function eliminates nulls first and operates only on remaining non-null values.
- **COUNT(*)** counts all rows of a table.
- Can use **DISTINCT** before column name to eliminate duplicates.
- 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. 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';

myCount
2

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';
```

myCount	mySum
2	54000.00

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;
```

myMin	myMax	myAvg
9000.00	30000.00	17000.00

Agenda

1. Simple Queries
2. Sorting Results
3. Aggregate Functions
4. Grouping Results
5. Nested Queries and Set Comparison
6. Multi-Table Queries
7. Combining Result Tables
8. Database Updates

GROUP BY Clause

- **GROUP BY** clause syntax:

  [GROUP BY columnName [, …] [**HAVING** condition]**]**
- 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:
  - ① column names
  - ② aggregate functions
  - ③ constants
  - ④ expression involving combinations of the above.
- All column names in **SELECT** list must appear in **GROUP BY** clause unless name is used only in an aggregate function.
- 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;
```

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

Restricted Groupings – HAVING clause

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

Example 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;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Agenda

1. Simple Queries
2. Sorting Results
3. Aggregate Functions
4. Grouping Results
5. Nested Queries and Set Comparison
6. Multi-Table Queries
7. Combining Result Tables
8. Database Updates

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 5.20 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);
```

- Cannot write '**WHERE** salary > **AVG**(salary)'

Subquery Rules

① **ORDER BY** clause may not be used in a subquery (although it may be used in outermost SELECT).

② Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.

③ By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an *alias*.

④ When subquery is an operand in a comparison, subquery must appear on right-hand side.

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’));
```

ANY and ALL

① **ANY and ALL** may be used with subqueries that produce a single column of numbers.

② With **ALL**, condition will only be true if it is satisfied by *all* values produced by subquery.

③ With **ANY**, condition will be true if it is satisfied by *any* values produced by subquery.

④ If subquery is empty, **ALL** returns true, **ANY** returns false.

⑤ **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 5.23 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');
```

Agenda

1. Simple Queries
2. Sorting Results
3. Aggregate Functions
4. Grouping Results
5. Nested Queries and Set Comparison
6. Multi-Table Queries
7. Combining Result Tables
8. Database Updates

Multi-Table Queries

- Multi-table queries support:

  FROM TableName [[**AS**] alias] [, ...]

  WHERE join condition

- Use join if result columns come from more than one table.
- 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).
- 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.

Computing a Join

- Procedure for generating results of a join are:

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

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

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

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

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

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;
```

- 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.

Alternative JOIN Constructs

- SQL provides alternative ways to specify joins (Table1 and Table2 both have attributes A):

  ① FROM Table1 **JOIN** Table2 **USING** A

  ② FROM Table1 **JOIN** Table2 **ON** Table1.A = Table2.A

  ③ FROM Table1 **NATURAL JOIN** Table2

  ④ FROM Table1 **FULL|LEFT|RIGHT JOIN** Table2 **ON** Table1.A = Table2.A

- In each of the above cases, FROM replaces original FROM and WHERE.

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;
```

- 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.

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

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;
```

- 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.

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

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;
```

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

① **(NOT) EXISTS** is for use only with subqueries.

② **EXISTS** produces a simple true/false result.

I. **True** if and only if there exists at least one row in result table returned by subquery.

II. **False** if subquery returns an empty result table.

③ **NOT EXISTS** is the opposite of EXISTS.

④ As **(NOT) EXISTS** check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.

⑤ 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');
```

- 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';
```

Agenda

1. Simple Queries
2. Sorting Results
3. Aggregate Functions
4. Grouping Results
5. Nested Queries and Set Comparison
6. Multi-Table Queries
7. Combining Result Tables
8. Database Updates

Union, Intersect, and Difference (Except)

- Format of set operator clause in each case is:

  Subquery ***op*** [**ALL**] Subquery

- ***Op*** *could be* ***UNION, INTERSECT,*** *and* ***EXCEPT***.
- If **ALL** specified, result can include duplicate rows.

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 5.33 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);
```

- 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 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);
```

- 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);
```

Agenda

1. Simple Queries
2. Sorting Results
3. Aggregate Functions
4. Grouping Results
5. Nested Queries and Set Comparison
6. Multi-Table Queries
7. Combining Result Tables
8. Database Updates

Database Update Operations

- **INSERT**
  - **INSERT … VALUES**
  - **INSERT … SELECT**
- **UPDATE**
- **DELETE**

INSERT … VALUES

- The **INSERT** statement syntax:

  INSERT INTO TableName [(columnList)]

  VALUES (dataValueList)

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

Example 5.35 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 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
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 5.37 INSERT … SELECT

- Assume there is a table **StaffList** that contains staffNo and names of staff:

  StaffList (staffNo, fName, lName)

- Populate **StaffList** using Staff table.

```
INSERT INTO StaffList
    (SELECT staffNo, fName, lName
     FROM Staff);
```

UPDATE

- The **UPDATE** statement syntax:

  UPDATE TableName

  SET columnName = dataValue [, ...]

  [**WHERE** searchCondition]

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

- The **DELETE** statement syntax:

  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;
```

Section Objectives

In this section you will learn:

1. How to retrieve data from database using SELECT and clauses:
   ① use compound WHERE conditions.
   ② sort query results using ORDER BY.
   ③ use aggregate functions.
   ④ group data using GROUP BY and HAVING.
   ⑤ use subqueries.
   ⑥ join tables together.
   ⑦ perform set operations (UNION, INTERSECT, EXCEPT).
2. How to update database using INSERT, UPDATE, and DELETE.

Questions?

Assignments

- **Multiple-Choice Quiz 3**
- **Exercise 3: part Ⅱ**

Prerequisites for Next Section

- **Readings:**
  - **Required:** Connolly and Begg, sections 3.4 and 6.4
  - **Required:** Connolly and Begg, sections 6.2 and 6.5
  - **Required:** Connolly and Begg, section 6.6

- **Assessments:**
  - Multiple-Choice Quiz 4