Chapter 6
SQL: Data Manipulation

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
  - Join tables together
- How to update database using INSERT, UPDATE, and DELETE.

Objectives of SQL

- 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.
- SQL (Structured Query Language) comes with 2 major components:
  - A DDL (Data definition language) for defining database structure.
  - A DML (Data manipulation language) for retrieving and updating data.

Objectives of SQL (cont'd)

- SQL is non-procedural - you specify *what* information you require, rather than *how* to get it
- SQL is defined by the ANSI/ISO SQL Standard. The SQL standard has been evolving since 1986 and several versions exist: "SQL-92", "SQL:1999", "SQL:2003"
- 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;
```

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

- 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).

Writing SQL Commands (cont'd)

- 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 (cont'd)

- 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).

SELECT Statement

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

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

SELECT Statement (cont'd)

SELECT	Specifies which columns are to appear in output.
FROM	Specifies table(s) to be used.
WHERE	Filters rows.
GROUP BY	Forms groups of rows with same column value.
HAVING	Filters groups subject to some condition.
ORDER BY	Specifies the order of the output.

Relational schema of DreamHome rental database

- Branch (branchNo, street, city, postcode)
- Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)
- PropertyForRent (propNo, street, city, postcode, type, noOfRooms, rent, ownerNo, staffNo, branchNo)
- PrivateOwner (ownerNo, fName, lName, address, telephone, email, password)
- Client (clientNo, fName, lName, telephone, email, prefType, maxRent)
- Viewing (clientNo, propNo, viewDate, comment)
- Registration (clientNo, propNo, staffNo, dateJoined)

Instances of DreamHome rental database

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Instances of DreamHome rental database (cont'd)

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Client

clientNo	fName	lName	telNo	prefType	maxRent	eMail
CR76	John	Kay	0207-774-5632	Flat	425	john.kay@gmail.com
CR56	Aline	Stewart	0141-848-1825	Flat	350	astewart@hotmail.com
CR74	Mike	Ritchie	01475-392178	House	750	mritchie01@yahoo.co.uk
CR62	Mary	Tregear	01224-196720	Flat	600	maryt@hotmail.co.uk

Instances of DreamHome rental database (cont'd)

PrivateOwner

ownerNo	fName	lName	address	telNo	eMail	password
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212	jkeogh@lhh.com	********
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419	cfarrel@gmail.com	********
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728	tinam@hotmail.com	********
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025	tony.shaw@ark.com	********

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-08	too small
CR76	PG4	20-Apr-08	too remote
CR56	PG4	26-May-08	
CR62	PA14	14-May-08	no dining room
CR56	PG36	28-Apr-08	

Registration

clientNo	branchNo	staffNo	dateJoined
CR76	B005	SL41	2-Jan-08
CR56	B003	SG37	11-Apr-07
CR74	B003	SG37	16-Nov-06
CR62	B007	SA9	7-Mar-07

Example 6.1 All Columns, All Rows

List full details of all staff.

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

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

```
SELECT *
FROM Staff;
```

Note that the Where clause is unnecessary here.

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;
```

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Table 6.2 Result table for Example 6.2.

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

Example 6.3 Use of Distinct

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

SELECT propertyNo
FROM Viewing;

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-08	too small
CR76	PG4	20-Apr-08	too remote
CR56	PG4	26-May-08	
CR62	PA14	14-May-08	no dining room
CR56	PG36	28-Apr-08	

propertyNo
PA14
PG4
PG4
PA14
PG36

Result table with duplicates

Example 6.3 Use of Distinct (cont'd)

Use DISTINCT to eliminate duplicates:

SELECT DISTINCT propertyNo
FROM Viewing;

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-08	too small
CR76	PG4	20-Apr-08	too remote
CR56	PG4	26-May-08	
CR62	PA14	14-May-08	no dining room
CR56	PG36	28-Apr-08	

propertyNo
PA14
PG4
PG36

Result table with out duplicates

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;
```

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Result table

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

Example 6.4 Calculated Fields (cont'd)

- To name column, use AS clause:

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

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

staffNo	fName	lName	monthlySalary
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

Row Selection (WHERE clause)

- The previous examples retrieve all rows from a table.
- To restrict the rows that are retrieved, use the WHERE clause.
- Consists of the keyword WHERE followed by a search condition that specifies the rows to be retrieved.

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;
```

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Result table

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

Comparison Operators of SQL

- In SQL, the following simple comparison operators are available:
  - = equals
  - < > is not equal to
  - != is not equal to (allowed in some dialects)
  - < is less than
  - > is greater than
  - <= is less than or equal to
  - >= is greater than or equal to
- More complex predicates can be generated using the logical operators AND, OR, NOT, with parentheses (if needed or desired) to show the order of evaluation.
- When more than one logical operator is used in a statement, NOT is evaluated first, then AND, and finally OR

Example 6.6 Compound comparison search condition

List addresses of all branch offices in London or Glasgow.

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

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Result table

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

Example 6.7 Range search condition (BETWEEN /NOT BETWEEN)

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.
- Condition can be written as

  WHERE salary >= 20000 AND salary <=30000;

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

Example 6.8 Set membership search condition (IN / NOT IN)

List the staff who are managers or supervisors.

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

- Condition can be written as

  WHERE position = 'Manager' OR position = 'Supervisor';

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

Example 6.9 Pattern match search condition (LIKE / NOT LIKE)

Find all owners with the string 'Glasgow' in their address.

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

PrivateOwner

ownerNo	fName	lName	address	telNo	eMail	password
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212	jkeogh@lhh.com	********
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419	cfarrel@gmail.com	********
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728	tinam@hotmail.com	********
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025	tony.shaw@ark.com	********

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

27

Pattern Matching

- SQL has two special pattern matching symbols:
  - %: sequence of zero or more characters;
  - _ (underscore): any single character.
  - Note that Microsoft Access uses * and ? Instead of % and _
- LIKE '%Glasgow%' means a sequence of characters of any length containing '*Glasgow*'.

28

Example 6.10 NULL search condition (IS NULL / IS NOT NULL)

List details of all viewings on property PG4 where a comment has not been supplied.

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

- Note that the following condition does not work

  WHERE propertyNo = 'PG4' AND comment =' ';

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-08	too small
CR76	PG4	20-Apr-08	too remote
CR56	PG4	26-May-08	
CR62	PA14	14-May-08	no dining room
CR56	PG36	28-Apr-08	

clientNo	viewDate
CR56	26-May 08

Sorting Results (ORDER BY clause)

- ORDER BY clause consists of a list of column identifiers that the result is to be sorted on, separated by commas.
- A column identifier may be either a column name or a column number that identifies an element of the SELECT list by its position within the list, 1 being the first.
- The ORDER BY clause allows the retrieved rows to be ordered in ascending (ASC) or descending (DESC) order on any column or combination of columns, regardless of whether that column appears in the result.
- The ISO standard specifies that NULLs in a column sorted with ORDER BY should be treated as either less than all non-null values or greater than all non-null values. The choice is left to the DBMS to implement.
  - For Oracle, you can set ORDER BY XXX NULLS FIRST

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;
```

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

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

Example 6.12 Multiple column ordering

Produce abbreviated list of properties in order of property type.

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

- Four flats in this list - as no minor sort key specified, system arranges these rows in any order it chooses.

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

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

Example 6.12 Multiple column ordering (cont'd)

- To arrange in order of rent, specify minor order:

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

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

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

SELECT statement – Aggregate Functions

- Besides retrieving rows and columns from the database, we can perform summation or aggregation of data, similar to the totals at the bottom of a report.
- 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 – Aggregate Functions (cont'd)

- Each operates on a single column of a table and returns a single value.
- **COUNT**, **MIN**, and **MAX** apply to numeric and non-numeric fields, but **SUM** and **AVG** may be used on numeric fields only.
- Apart from **COUNT**(*), each function eliminates nulls first and operates only on remaining non-null values.
- **COUNT**(*) counts all rows of a table, regardless of whether nulls or duplicate values occur whereas **COUNT**(*fieldname*) ignores nulls.

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;
```

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

myCount
5

Example 6.14 Use of COUNT(DISTINCT)

How many different properties viewed in May 2008?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN '1-May-08' AND '31-May-08';
```

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-08	too small
CR76	PG4	20-Apr-08	too remote
CR56	PG4	26-May-08	
CR62	PA14	14-May-08	no dining room
CR56	PG36	28-Apr-08	

myCount
2

Note that count(distinct) is not supported in MS Access, where you have to write sub-query to handle this.

Example 6.15 Use of COUNT and SUM

Find the total number of Managers and sum of their salaries.

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

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

myCount	mySum
2	54000.00

Example 6.16 Use of MIN, MAX, AVG

Find minimum, maximum, and average staff salary.

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

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

myMin	myMax	myAvg
9000.00	30000.00	17000.00

Grouping Results – GROUP BY clause

- Use **GROUP BY** clause to get sub-totals for each group.
- If **WHERE** is used with **GROUP BY**, **WHERE** is applied first, then groups are formed from remaining rows satisfying the search condition.

Example 6.17 Use of GROUP BY

Find the number of staff in each branch and the sum of their salaries.

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

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

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Grouping Results – GROUP BY clause

- If **SELECT** list includes an aggregate function and there is no **GROUP BY** clause, no item in the **SELECT** list can include any reference to a column unless that column is the argument to an aggregate function. For example, the following is **illegal**:

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

- There may be column names in the **GROUP BY** clause that do not appear in the **SELECT** list.

Restricting groupings – HAVING clause

- **HAVING** clause is designed for use with **GROUP BY** to restrict groups that appear in final result table.
- **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. E.g.

```
SELECT type, AVG (price)
FROM Titles
GROUP BY type
HAVING type LIKE 'p%';
```

Column names in **HAVING** clause appears in the **GROUP BY** list.

Example 6.18 Use of HAVING

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

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

Column names in **HAVING** clause contained within an aggregate function.

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Subqueries

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

Example 6.19 Using a 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');
```

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

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

Example 6.19 Using a subquery with equality (cont'd)

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

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


Example 6.20 Using a subquery with an aggregate function

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

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

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

staffNo	fName	lName	position	salDiff
SL21	John	White	Manager	13000.00
SG14	David	Ford	Supervisor	1000.00
SG5	Susan	Brand	Manager	7000.00

Example 6.20 Using a subquery with an aggregate function

- Cannot write
  SELECT *, salary – **AVG**(salary) **As** SalDiff ✘
- Because as discussed before,
  - If **SELECT** list includes an aggregate function and there is no **GROUP BY** clause, no item in the **SELECT** list can include any reference to a column unless that column is the argument to an aggregate function.

Example 6.20 Using a subquery with an aggregate function

- Aggregate functions can be used only in **SELECT** list and in **HAVING** clause.
- Hence, 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;
```

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.
- When subquery is an operand in a comparison, subquery must appear on right-hand side. Hence, the following will be incorrect:

```
SELECT staffNo, fName, lName, position,
FROM Staff
WHERE (SELECT AVG(salary) FROM Staff) < salary;
```

Multi-table Queries

- Our examples so far only have result table with columns from a single table.
- To combine columns from several tables into a result table, use **join** operation.
- To perform join, include more than one table names in **FROM** clause.
  - Use comma as separator and typically include **WHERE** clause to specify join column(s).
  - Can 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 regarding the source of column name.

Example 6.21 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.
- These two columns are called the **matching columns** for the two tables.
- This cannot be written using subquery because it needs columns from several tables into a result table.

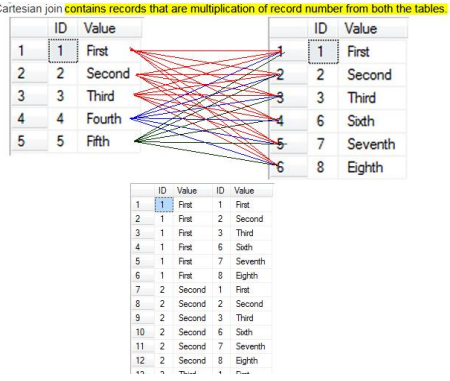
Example 6.21 Simple Join (cont'd)

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

Use "DemoOnSimpleJoin" to demonstrate the result in MS Access.

What happens when a join is performed

Cartesian join contains records that are multiplication of record number from both the tables.



	ID	Value
1	1	First
2	2	Second
3	3	Third
4	4	Fourth
5	5	Fifth

	ID	Value
1	1	First
2	2	Second
3	3	Third
4	6	Sixth
5	7	Seventh
6	8	Eighth

	ID	Value	ID	Value
1	1	First	1	First
2	1	First	2	Second
3	1	First	3	Third
4	1	First	6	Sixth
5	1	First	7	Seventh
6	1	First	8	Eighth
7	2	Second	1	First
8	2	Second	2	Second
9	2	Second	3	Third
10	2	Second	6	Sixth
11	2	Second	7	Seventh
12	2	Second	8	Eighth

Example 6.22 Sorting a Join

For each branch, list staff 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;
```

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

Join vs Subquery

- From your observation, do you notice the difference between join and subquery?
- *Under what circumstances would you not be able to use a subquery?*
  - With a subquery, the columns specified in the SELECT list are restricted to one table. Thus, cannot use a subquery if the SELECT list contains columns from more than one table.

Join vs Subquery

- It is a good practice to avoid multiple levels of nested subqueries, since they are not easily readable and do not have good performance.
- In general, it is better to write a query with JOINs.

Example 6.23 Three-table Join

For each branch, list staff numbers and names of 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;
```

- The Branch and Staff details are joined to link each branch to the staff working there.
- The Staff and PropertyForRent details are joined to link staff to the properties they manage.

Example 6.23 Three-table Join (cont'd)

Table 6.23 Result table for Example 6.23

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

Inner Joins

- The join operations discussed so far are inner joins.
- Data from 2 tables are combined by forming pairs of related rows where the matching columns in each table have the same value.
- If a row is unmatched, it is omitted from the result table.
- Consider the following tables:

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Inner Joins (cont'd)

The inner join of these two tables

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

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

- The result table has two rows where cities are the same.
- There is no row corresponding to branches in Bristol and there is no row corresponding to property in Aberdeen.
- To include unmatched rows in result table, use an Outer join.

Outer Joins

- Outer join retains rows that do not satisfy the join condition.
- There are 3 types of outer joins:
  - Left:
    - Includes all of the records from the first (left) of two tables, even if there are no matching values for records in the second (right) table.
  - Right
    - Includes all of the records from the second (right) of two tables, even if there are no matching values for records in the first (left) table.
  - Full
    - Not implemented in MS Access
    - http://www.databasejournal.com/features/msaccess/article.php/3516561/Implementing-the-Equivalent-of-a-FULL-OUTER-JOIN-in-Microsoft-Access.htm

Example 6.24 Left Outer Join

List all branch offices and any properties that are in the same city.

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

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

- The result table includes not only those rows that have the same city, but also those rows of the first (left) table that are unmatched with rows from the second (right) table.
- The columns from the second table are filled with NULLs.

Example 6.25 Right Outer Join

List all properties and any branch offices that are in the same city.

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

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

- The result table includes not only those rows that have the same city, but also those rows of the second (right) table that are unmatched with rows from the first (left) table.
- The columns from the first table are filled with NULLs.

Example 6.26 Full Outer Join

List the branch offices and properties that are in the same city, along with any unmatched branches or properties.

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

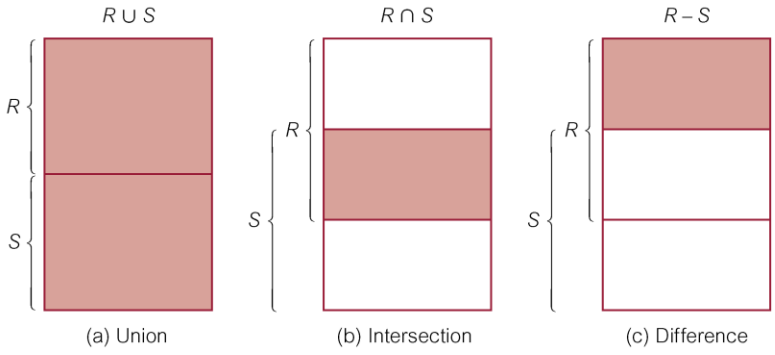
branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

- The result table includes not only those rows that have the same city, but also unmatched rows in both tables.
- The unmatched columns are filled with NULLs.

Combining Result Tables

- 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 a table containing all rows in either A or B or both.
- **Intersection** of two tables, A and B, is a table containing all rows common to both A and B.
- **Difference** of two tables, A and B, is a table containing all rows in A but not in B.
- Two tables must be *union compatible* (having same structure in terms of number of columns and corresponding data types and lengths) in order to carry out these operations.

Union, Intersection, and Difference



Union, Intersect, Except (Difference)

- The 3 set operators are called **UNION**, **INTERSECT**, **EXCEPT**.
- Format of set operator clause in each case is:
  Operator [**ALL**] [**CORRESPONDING** [**BY** {column1 [, ...]}]]
- If **CORRESPONDING BY** is specified, set operation is performed on the named column(s).
- If **CORRESPONDING** is specified but not **BY** clause, operation is performed on common columns of both tables.
- If **ALL** is specified, result can include duplicate rows.

Example 6.27 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);
```

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

- Produces result tables from both queries and merges both tables together, with duplicate rows removed.

city
London
Glasgow
Aberdeen
Bristol

Example 6.28 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);
```

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

- Produces result tables from both queries and then creates a single result table consisting of those rows that are common to both result tables.

city
Glasgow
London

Example 6.29 Use of INTERSECT (cont'd)

- Could rewrite the query without INTERSECT operator:

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

- Or:

```
SELECT DISTINCT city
FROM Branch b
WHERE city IN (SELECT city
                FROM PropertyForRent p);
```

Example 6.30 Use of EXCEPT

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

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

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

- Produces result tables from both queries and then creates a single result table consisting of those rows that appear in the first result table but not in the second one.

city
Bristol

Example 6.31 Use of EXCEPT (cont'd)

- Could rewrite the query without EXCEPT operator:

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

Database Updates

- The 3 available SQL statements to modify the contents of the tables in the database:
  - **INSERT**: adds new rows of data to a table
  - **UPDATE**: modifies existing data in a table
  - **DELETE**: removes rows of data from a table

INSERT

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

- *TableName* can be the name of a base table or an updatable view.
- *columnList* is optional;
  - if omitted, SQL assumes a list of all columns in their original **CREATE TABLE** order.
  - if specified, any columns omitted must have been declared as NULL columns when table was created, unless DEFAULT was specified when creating column.

INSERT (cont'd)

- *dataValueList* must match *columnList* as follows:
  - number of items in each list must be the same;
  - there 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 6.32 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');
```

- As we are inserting data into each column in the order the table was created, there is no need to specify a column list.

Example 6.33 INSERT using defaults

Insert a new row into Staff table supplying data for all mandatory columns: staffNo, fName, lName, position, salary, branchNo

```
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');
```

- Since we are inserting data into only certain columns, we must specify the names of the columns that we are inserting data into.

INSERT…SELECT

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

```
INSERT INTO TableName [ (columnList) ]
    SELECT ...
```

- The rows inserted into the named table are identical to the result table produced by the subselect.

UPDATE

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

- *TableName* can be the name of a base table or an updatable view.
- **SET** clause specifies names of one or more columns that are to be updated.
- **WHERE** is optional;
  - if omitted, the named columns are updated for *all* rows in the table.
  - if present, only those rows that satisfy the *searchCondition* are updated.
- New *dataValue(s)* must be compatible with data type for corresponding column(s).

Example 6.34 UPDATE all rows

Give all staff a 3% pay increase.

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

- As the update applies to all rows, the **WHERE** clause is omitted.

Example 6.35 UPDATE specific rows

Give all Managers a 5% pay increase.

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


Example 6.36 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 the 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 itself.
    - To delete the table contents and the table definition the **DROP TABLE** statement must be used.
  - If specified, only those rows that satisfy the condition are deleted

Example 6.37 DELETE specific rows

Delete all viewings that relate to property PG4.

DELETE FROM Viewing
WHERE propertyNo = 'PG4';

Example 6.38 DELETE all rows

Delete all records from the Viewing table.

```
DELETE FROM Viewing;
```

- The **WHERE** clause is omitted, hence, all rows are deleted from the table, leaving only the table definition, so that we can still insert data into the table at a later stage.

REMINDERS

SELECT	If **SELECT** list includes an aggregate function and there is no **GROUP BY** clause, no item in the **SELECT** list can include any reference to a column unless that column is the argument to an aggregate function.
WHERE	Aggregate functions can be used only in SELECT list and in HAVING clause, but NOT in WHERE clause.
ORDER BY	**Can**not be used in a subquery.