

Chapter 6

SQL – Data Manipulation

Chapter 6 - Objectives

- **Th purpose and importance of SQL.**
- **The history and development of SQL.**
- **How to write an SQL command.**
- **How to retrieve data from database using SELECT.**
- **How to update database using INSERT, UPDATE, and DELETE.**

Chapter 6 - Objectives

- **How to build SQL statements that:**
 - use the WHERE clause to retrieve rows that satisfy various conditions;
 - sort query results using ORDER BY;
 - use the aggregate functions of SQL;
 - group data using GROUP BY;
 - join tables together;
 - perform set operations (UNION, INTERSECT, EXCEPT).

Over the last few years, SQL has become the standard relational database language.

Introduction to 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
- Command structure/syntax must be easy to learn.
- It must be portable.
- SQL is intended to satisfy these requirements.

Introduction to SQL

- **SQL is a transform-oriented language (using relation to transform input to output) with 2 major components:**
 - A DDL for defining database structure.
 - A DML for retrieving and updating data.
- **Until SQL:1999 (SQL3), 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.**

SQL is Relatively Easy to Learn

- It is non-procedural – you specify *what* information you require, rather than *how* to get it;
- Like most modern languages, it is essentially free-format – position of the text doesn't matter;

SQL is Relatively Easy to Learn

- The command structure consists of standard English words:

```
1) CREATE TABLE Staff (  staffNo VARCHAR(5),  
                           IName VARCHAR(15),  
                           salary DECIMAL(7,2) );  
  
2) INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);  
  
3) SELECT staffNo, IName, salary  
   FROM Staff  
   WHERE salary > 10000;
```

SQL is Relatively Easy to Learn

- Can be used by a range of users including DBAs, management personnel, 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 a 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.
- In summer 2008, SQL:2008 was released.
- In late 2011, SQL:2011 was released.

Importance of SQL

- SQL is the first and, so far, only standard database language to gain wide acceptance.
- 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:**
 - **ISO's Information Resource Directory System (IRDS) Standard**
 - **Remote Data Access (RDA) Standard.**

SQL Terminology

- The ISO SQL standard does not use the formal terms of relations, attributes, and tuples, instead using the terms tables, columns, and rows.
- SQL does not adhere strictly to the definition of the relational model.
 - It allows the table produced as the result of the SELECT statement to contain duplicate rows.
 - It imposes an ordering on the columns, and it allows the user to order the rows of a result table.

Writing SQL Commands

- SQL statement consists of *reserved words* and *user-defined words*.
 - Reserved words are a fixed part of SQL and have a fixed meaning. They 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.
- The words in a statement are built according to a set of syntax rules. SQL ends the statement usually with the semicolon “;”.

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 to show the relationship.

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. (a|b|c)
 - Curly braces indicate a *required element*. {a}
 - Square brackets indicate an *optional element*. [a]
 - ... indicates *optional repetition* (0 or more).
{ a|b } [, c...] :
means either a or b followed by zero or more repetitions of c separated by commas.

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).

• E.g.

```
INSERT INTO PropertyForRent ( propertyNo, street, city,  
                             postcode, type, rooms, rent, ownerNo,  
                             staffNo, branchNo)
```

```
VALUES ( 'PA14', '16 Holhead', 'Aberdeen',  
        'AB7 5SU', 'House', 6, 650.00, 'CO46', 'SA9',  
        'B007');
```

SELECT Statement

The purpose is to retrieve and display data from one or more database tables. It is capable of performing the equivalent of the relational algebra's Selection, Projection, and Join operations in a single statement.

SELECT [DISTINCT | ALL]

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

FROM TableName [alias] [, ...]

[WHERE condition]

[GROUP BY columnList] [HAVING condition]

[ORDER BY columnList]

Sequence of Processing in SELECT

- 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.

SELECT Statement

- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.
- The SELECT operation is closed: the result of a query on a table is another table

Example 6.1 All Columns, All Rows

List full details of all staff.

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

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

```
SELECT *  
FROM Staff;
```

Example 6.1 All Columns, All Rows

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

Example 6.2 Specific Columns, All Rows

Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

```
SELECT staffNo, fName, lName, salary  
FROM Staff;
```


Example 6.2 Specific Columns, All Rows

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;
```

Unlike Projection operation,
SELECT doesn't eliminate
the duplicates.

propertyNo
PA14
PG4
PG4
PA14
PG36

Example 6.3 Use of DISTINCT

- Use DISTINCT to eliminate duplicates:

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

propertyNo
PA14
PG4
PG36

Example 6.4 Calculated Fields

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

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

The column name for the expression is the column # in the 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

- To name column, use AS clause:

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

Row selection (WHERE clause)

- The previous examples show the use of the **SELECT** statement to retrieve all rows from a table.
- We often need to restrict the rows that are retrieved. This can be achieved with the **WHERE** clause, which consists of the keyword **WHERE** followed by a search condition that specifies the rows to be retrieved.
- There are five basic search conditions.

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;
```

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

Example 6.6 Compound Comparison Search Condition

List addresses of all branch offices in London or Glasgow.

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

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

Example 6.7 Range Search Condition

List all staff with a salary between 20,000 and 30,000.

```
SELECT staffNo, fName, lName, position, salary  
FROM Staff  
WHERE salary BETWEEN 20000 AND 30000;
```

- **BETWEEN** test includes the endpoints of range.

Example 6.7 Range Search Condition

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

Example 6.7 Range Search Condition

- There is 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;
```

- Simpler, though, for testing a range of values.

Example 6.8 Set Membership

List all managers and supervisors.

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

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

Example 6.8 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';
```

- But IN is more efficient when set contains many values.

Example 6.9 Pattern Matching

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

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

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

Example 6.9 Pattern Matching

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

Example 6.10 NULL Search Condition

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

- There are 2 viewings for property PG4, one with and one without a comment.
- Have to test for null explicitly using special keyword IS NULL:

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


Example 6.10 NULL Search Condition

clientNo	viewDate
CR56	26-May-04

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

Sorting Results (ORDER BY Clause)

- In general, the rows of an SQL query result table are not arranged in any particular order.
- However, we can ensure the results of a query are sorted using the ORDER BY clause.
- The ORDER BY clause consists of a list of column ids that the result is to be sorted on, separated by commas.
- A column id may be either a column name or a column number according to its position in the SELECT list.
- The sort can be ascending (ASC) or descending (DESC).
- The ORDER BY clause must always be the last clause of the SELECT statement.

Example 6.11 Single Column Ordering

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

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

Example 6.11 Single Column Ordering

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

Example 6.12 Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

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

Example 6.12 Multiple Column Ordering

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

Example 6.12 Multiple Column Ordering

- Four flats in this list - as no minor sort key is specified, system arranges these rows in any order it chooses.
- To arrange in order of rent, specify minor order:

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

Example 6.12 Multiple Column Ordering

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

SELECT Statement - Aggregates

- **ISO standard defines five aggregate functions:**
 - ☐ **COUNT** returns number of values in specified column.
 - ☐ **SUM** returns sum of values in specified column.
 - ☐ **AVG** returns average of values in specified column.
 - ☐ **MIN** returns smallest value in specified column.
 - ☐ **MAX** returns largest value in specified column.
- **They are similar to the totals at the bottom of a report.**

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**.

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 outside an aggregate function. For example, the following is illegal:

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

Example 6.13 Use of COUNT(*)

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

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

myCount
5

Example 6.14 Use of COUNT(DISTINCT)

How many different properties were viewed in May 2013?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount  
FROM Viewing  
WHERE viewDate BETWEEN '1-May-13'  
AND '31-May-13';
```

myCount
2

Example 6.15 Use of COUNT and SUM

Find number of Managers and sum of their salaries.

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

myCount	mySum
2	54000.00

Example 6.16 Use of MIN, MAX, AVG

Find minimum, maximum, and average staff salary.

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

myMin	myMax	myAvg
9000.00	30000.00	17000.00

SELECT Statement - Grouping

- The previous summary queries condense all the detailed data into a single row of data.
- 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 the search condition.
- ISO considers two nulls to be equal for purposes of GROUP BY.

Example 6.17 Use of GROUP BY

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

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

Example 6.17 Use of GROUP BY

branchNo	staffNo	salary		COUNT(staffNo)	SUM(salary)
B003	SG37	12000.00	}	3	54000.00
B003	SG14	18000.00			
B003	SG5	24000.00			
B005	SL21	30000.00	}	2	39000.00
B005	SL41	9000.00			
B007	SA9	9000.00	}	1	9000.00

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 clause, but WHERE clause filters individual rows whereas HAVING clause filters groups.**
- **Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.**
- **HAVING clause is not a necessary part of SQL.**

Example 6.18 Use of HAVING

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

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

Example 6.18 Use of HAVING

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Subqueries

- Some SQL statements can have 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.


Three Types of Subquery

- A ***scalar subquery*** returns a single column and a single row, that is, a single value. In principle, a scalar subquery can be used whenever a single value is needed.
- A ***row subquery*** returns multiple columns, but only a single row. A row subquery can be used whenever a row value constructor is needed.
- A ***table subquery*** returns one or more columns and multiple rows. A table subquery can be used whenever a table is needed.

Example 6.19 Subquery with Equality

List staff who work in branch at '163 Main St'.

```
SELECT staffNo, fName, lName, position  
FROM Staff  
WHERE branchNo =  
    (SELECT branchNo  
     FROM Branch  
     WHERE street = '163 Main St');
```



There will be only one such branch number, so this is an example of a scalar subquery.

Example 6.19 Subquery with Equality

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

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

Example 6.19 Subquery with Equality

- Result table:

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

Example 6.20 Subquery with Aggregate

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

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

Example 6.20 Subquery with Aggregate

- Cannot write 'WHERE salary > AVG(salary)' because aggregate functions cannot be used in the WHERE clause.
- Instead, use subquery to find average salary (17000), and then use outer SELECT to find those staff with salary greater than this:

```
SELECT staffNo, fName, lName, position,  
       salary – 17000 As salDiff  
FROM Staff  
WHERE salary > 17000;
```

Example 6.20 Subquery with Aggregate

- Result table:

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

Subquery Rules

- **ORDER BY** clause may not be used in 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. It is possible to refer to a table in a **FROM** clause of an outer query by qualifying the column name using alias (see example 6.24).

Subquery Rules


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

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

Example 6.21 Nested subquery: use of IN

List properties handled by staff at '163 Main St'.

```
SELECT propertyNo, street, city, postcode, type, rooms, rent
FROM PropertyForRent
WHERE staffNo IN
    (SELECT staffNo
     FROM Staff
     WHERE branchNo =
        (SELECT branchNo
         FROM Branch
         WHERE street = '163 Main St'));
```



- Since there may be more than one staffNo found, we cannot use the equality condition (=) in the outermost query.

Example 6.21 Nested subquery: use of IN

● Result table:

propertyNo	street	city	postcode	type	rooms	rent
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375
PG21	18 Dale Rd	Glasgow	G12	House	5	600

ANY and ALL

- ANY and ALL may be used with subqueries that produce a single column of numbers.
- With ALL, condition will only be true if it is satisfied by *all* values produced by subquery.
- With ANY, condition will be true if it is satisfied by *any* values produced by subquery.
- If subquery is empty, ALL returns true, ANY returns false.
- SOME may be used in place of ANY.

Example 6.22 Use of ANY/SOME

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

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

Example 6.22 Use of ANY/SOME

- Inner query produces set {12000, 18000, 24000} and outer query selects those staff whose salaries are greater than any of the values in this set (that is, greater than the minimum value, 12000) .
- Result table:

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

Example 6.23 Use of ALL

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

```
SELECT staffNo, fName, lName, position, salary  
FROM Staff
```

```
WHERE salary > ALL
```

```
(SELECT salary  
FROM Staff
```

```
WHERE branchNo = 'B003');
```

Example 6.23 Use of ALL

- Result table:

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00

Multi-Table Queries

- To obtain information from more than one table, either use a subquery or a join.
- Can use subqueries if result columns come from same table.
- If result columns come from more than one table, a join must be used.
- To perform join, include more than one table in FROM clause, using a comma as a separator and typically including a WHERE clause to specify join column(s).

Multi-Table Queries

- It is 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 6.24 Simple Join

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

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

Example 6.24 Simple Join

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

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

Alternative JOIN Constructs

- SQL provides alternative ways to specify joins:

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

FROM Client JOIN Viewing USING clientNo

FROM Client NATURAL JOIN Viewing

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

Example 6.25 Sorting a join

For each branch, list numbers and names of staff who manage properties, and properties they manage.

```
SELECT s.branchNo, s.staffNo, fName, lName,  
       propertyNo  
FROM Staff s, PropertyForRent p  
WHERE s.staffNo = p.staffNo  
ORDER BY s.branchNo, s.staffNo, propertyNo;
```

Example 6.25 Sorting a join

● Result table:

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

Example 6.26 Three Table Join

For each branch, list staff who manage properties, including city in which branch is located and properties they manage.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName,  
       propertyNo  
FROM Branch b, Staff s, PropertyForRent p  
WHERE b.branchNo = s.branchNo AND  
       s.staffNo = p.staffNo  
ORDER BY b.branchNo, s.staffNo, propertyNo;
```


Example 6.26 Three Table Join

- Result Table:

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

- Alternative formulation for FROM and WHERE:

**FROM (Branch b JOIN Staff s USING branchNo) AS
bs JOIN PropertyForRent p USING staffNo**

Example 6.27 Multiple Grouping Columns

Find number of properties handled by each staff member.

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS myCount  
FROM Staff s, PropertyForRent p  
WHERE s.staffNo = p.staffNo  
GROUP BY s.branchNo, s.staffNo  
ORDER BY s.branchNo, s.staffNo;
```

Example 6.27 Multiple Grouping Columns

● Result Table:

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

Computing a Join

- A join is a subset of a more general combination of two tables known as the Cartesian product
- The Cartesian product of two tables is another table consisting of all possible pairs of rows from the two tables.
- SQL provides special format of SELECT for Cartesian product:

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

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 containing the selected columns 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.

Outer Joins

- For a join, 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 simplified Branch and PropertyForRent tables:

Branch1		PropertyForRent1	
branchNo	bCity	propertyNo	pCity
B003	Glasgow	PA14	Aberdeen
B004	Bristol	PL94	London
B002	London	PG4	Glasgow

Outer Joins

- The (inner) join of these two tables:

```
SELECT b.*, p.*
```

```
FROM Branch1 b, PropertyForRent1 p
```

```
WHERE b.bCity = p.pCity;
```

- Result Table:

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

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.
- There are three types of Outer join: Left, Right, and Full Outer joins.

Example 6.28 Left Outer Join

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

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

Example 6.28 Left Outer Join

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

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

Example 6.29 Right Outer Join

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

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

Example 6.29 Right Outer Join

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

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

Example 6.30 Full Outer Join

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

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

Example 6.30 Full Outer Join

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

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

EXISTS and NOT EXISTS

- **EXISTS and NOT EXISTS 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 6.31 Query using EXISTS

Find all staff who work in a London branch.

```
SELECT staffNo, fName, lName, position  
FROM Staff s  
WHERE EXISTS  
    (SELECT *  
     FROM Branch b  
     WHERE s.branchNo = b.branchNo AND  
           city = 'London');
```

Example 6.31 Query using EXISTS

- **Result Table:**

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

Example 6.31 Query using EXISTS

- Note, search condition `s.branchNo = b.branchNo` is necessary to consider correct branch record for each member of staff.
- If omitted, would get all staff records listed out because subquery:

```
SELECT * FROM Branch WHERE city='London'
```

would always be true and query would be:

```
SELECT staffNo, fName, lName, position FROM Staff  
WHERE true;
```

Example 6.31 Query using EXISTS

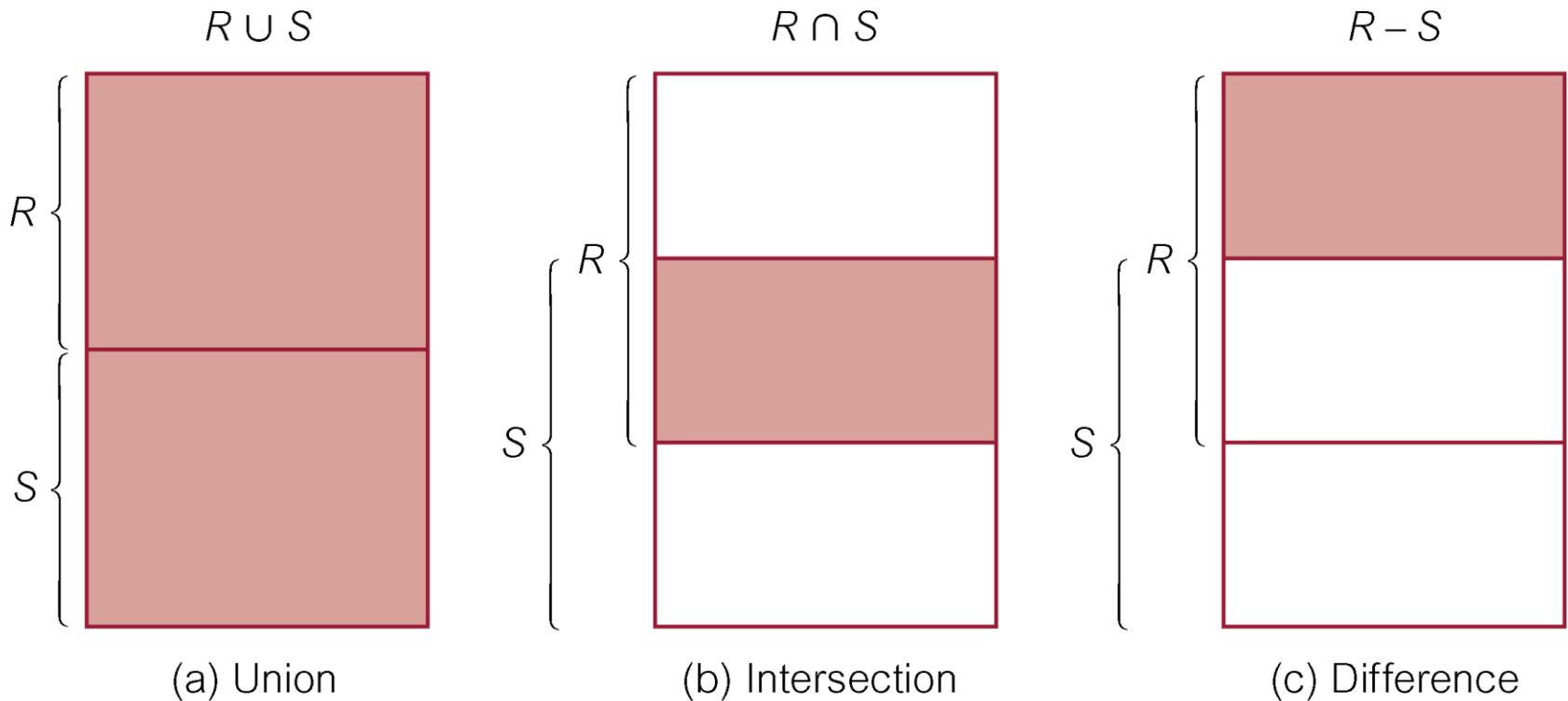
- Could also write this query using join construct:

```
SELECT staffNo, fName, lName, position  
FROM Staff s, Branch b  
WHERE s.branchNo = b.branchNo AND  
      city = 'London';
```

Union, Intersect, and Difference (Except)

- Can use normal set operations of Union, Intersection, and Difference to combine results of two or more queries into 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 A and B is a table containing all rows common to both A and B.
- Difference of A and B is a table containing all rows in A but not in B.
- Table A and B must be *union compatible* – *having the same structure*.

Union, Intersect, and Difference (Except)



Union, Intersect, and Difference (Except)

- Format of set operator clause in each case is:

op [ALL] [CORRESPONDING [BY {column1 [, ...]}]]

- If CORRESPONDING BY specified, set operation performed on the named column(s).
- If CORRESPONDING specified but not BY clause, operation performed on common columns.
- If ALL specified, result can include duplicate rows.

Example 6.32 Use of UNION

List all cities where there is either a branch office or a property.

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

Example 6.32 Use of UNION

- Or

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

- The ability to write a query in several equivalent forms illustrates one of the disadvantages of the SQL language.

Example 6.32 Use of UNION

- Produces result tables from both queries and merges both tables together.
- Duplicate rows are removed.
- Result Table:

city
London
Glasgow
Aberdeen
Bristol

Example 6.33 Use of INTERSECT

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

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

Example 6.33 Use of INTERSECT

• Or

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

city
Aberdeen
Glasgow
London

Example 6.33 Use of INTERSECT

- Could rewrite this query without INTERSECT operator:

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

- Or:

```
SELECT DISTINCT city FROM Branch b  
WHERE EXISTS  
    (SELECT * FROM PropertyForRent p  
     WHERE p.city = b.city);
```

Example 6.34 Use of EXCEPT

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

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

- Or

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

city
Bristol

Example 6.34 Use of EXCEPT

- Could rewrite this query without EXCEPT:

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

- Or

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


SQL Statements to Modify Data

- The commands for modifying the database are not as complex as the SELECT statement.
 - 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** has two forms. The first allows a single row to be inserted into a named table and has the following format:

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

- *TableName* may be either 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.
- 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:
 - The number of items in each list must be the same;
 - There must be a direct correspondence in the position of items in the two lists;
 - The data type of each item in *dataValueList* must be compatible with the data type of corresponding column.

Example 6.35 INSERT ... VALUES

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

```
INSERT INTO Staff
```

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

Example 6.36 INSERT using Defaults

Insert a new row into Staff table supplying data for all mandatory columns (excluding sex and DOB 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 ...
```

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

Example 6.37 INSERT ... SELECT

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

StaffPropCount(staffNo, fName, lName, propCnt)

Populate StaffPropCount using Staff and PropertyForRent tables.

Example 6.37 INSERT ... SELECT

```
INSERT INTO StaffPropCount
  (SELECT s.staffNo, fName, lName, COUNT(*))
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.staffNo, fName, lName)
UNION
(SELECT staffNo, fName, lName, 0
FROM Staff
WHERE staffNo NOT IN
  (SELECT DISTINCT staffNo
   FROM PropertyForRent));
```


Example 6.37 INSERT ... SELECT

staffNo	fName	lName	propCount
SG14	David	Ford	1
SL21	John	White	0
SG37	Ann	Beech	2
SA9	Mary	Howe	1
SG5	Susan	Brand	0
SL41	Julie	Lee	1

- If second part of UNION is omitted, those staff who currently do not manage any properties will be excluded.

UPDATE

- **UPDATE** allows the contents of existing rows in a named table to be changed.

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 *searchCondition* are updated.
- **New *dataValue(s)* must be compatible with data type(s) for corresponding column(s).**

Example 6.38/39 UPDATE All/Specific Rows

Give all staff a 3% pay increase.

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

Give all Managers a 5% pay increase.

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

Example 6.40 UPDATE Multiple Columns

Promote David Ford (staffNo='SG14') to Manager and change his salary to £18,000.

UPDATE Staff

SET position = 'Manager', salary = 18000

WHERE staffNo = 'SG14';

DELETE

- The DELETE statement allows rows to be deleted from a named table.

```
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 doesn't delete table. If *search_condition* is specified, only those rows that satisfy condition are deleted.

Example 6.41/42 DELETE Specific/All 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;
```