Chapter 13

Structured Query Language (SQL)

Transparencies

Chapter - 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.**

Chapter - Objectives

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

- How to update database using INSERT, UPDATE, and DELETE.
- Data types supported by SQL-92.
- How to create and delete tables.

Objectives of SQL

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

Objectives of SQL

- **SQL is a transform-oriented language with 2 major components:**
  - **A DDL for defining the database structure.**
  - **A DML for retrieving and updating data.**
- **SQL does not contain flow control commands. These must be implemented using a programming or job-control language, or interactively by the decisions of the user.**

Objectives

- **Can be used by a range of users including DBAs, management, application programmers, and other types of end users.**
- **An ISO standard now exists for SQL, making it both the formal and *de facto* standard language for relational databases.**

History of SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' or SEQUEL.
- A revised version SEQUEL/2 was defined in 1976 but name was subsequently changed to SQL for legal reasons.

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.

Importance of SQL

- **SQL has become part of application architectures such as IBM's Systems Application Architecture (SAA).**
- **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 Access Group trying to define enhancements that will support interoperability across disparate systems.**
- **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.**

Writing SQL Commands

- **Use extended form of BNF notation:**
  - **Upper case letters represent reserved words.**
  - **Lower case letters represent user-defined words.**
  - **| indicates a *choice* among alternatives.**
  - **Curly braces indicate a *required element.***
  - **Square brackets indicate an *optional element.***
  - **… indicates *optional repetition* (0 or more).**

SELECT Statement

```
SELECT      [DISTINCT | ALL]
  {* | [column_expression [AS new_name]] [,...] }
FROM        table_name [alias] [, ...]
[WHERE      condition]
[GROUP      BY    column_list]
  [HAVING condition]
[ORDER BY   column_list]
```

SELECT Statement

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.**

List full details of all staff.

```
SELECT sno, fname, lname, address, tel_no,
       position, sex, dob, salary, nin, bno
FROM staff;
```

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

Example 13.2 Specific Columns, All Rows

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

```
SELECT sno, fname, lname, salary
FROM staff;
```


Example 13.3 Use of DISTINCT

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

```
SELECT pno
FROM viewing;
```

Table 13.3(a) Result table for Example 13.3 with duplicates.

pno
PA14
PG4
PG4
PA14
PG36

(5 rows)

Example 13.3 Use of DISTINCT

- **Use DISTINCT to eliminate duplicates:**

```
SELECT DISTINCT pno
FROM viewing;
```

Table 13.3(b) Result table for Example 13.3 with duplicates eliminated.

pno
PA14
PG4
PG36

(3 rows)

Example 13.4 Calculated Fields

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

```
SELECT sno, fname, lname, salary/12
FROM staff;
```

Example 13.4 Calculated Fields

Table 13.4 Result table for Example 13.4.

sno	*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

(6 rows)

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

```
SELECT sno, fname, lname, salary/12
                AS monthly_salary
FROM staff;
```


Example 13.5 Comparison Search Condition

List all staff with a salary greater than 10,000.

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > 10000;
```

Example 13.5 Comparison Search Condition

Table 13.5 Result table for Example 13.5.

sno	*fname*	*lname*	*position*	*salary*
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Snr Asst	12000.00
SG14	David	Ford	Deputy	18000.00
SG5	Susan	Brand	Manager	24000.00

(4 rows)

Example 13.6 Compound Comparison Search Condition

List the addresses of all branch offices in London or Glasgow.

```
SELECT bno, street, area, city, pcode
FROM branch
WHERE city = 'London' OR city = 'Glasgow';
```

Example 13.6 Compound Comparison Search Condition

Table 13.6 Result table for Example 13.6.

bno	*street*	*area*	*city*	*pcode*
B5	22 Deer Rd	Sidcup	London	SW1 4EH
B3	163 Main St	Partick	Glasgow	G11 9QX
B2	56 Clover Dr		London	NW10 6EU

(3 rows)

Example 13.7 Range Search Condition

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

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary BETWEEN 20000 AND 30000;
```

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

Example 13.7 Range Search Condition

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

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary>=20000 AND salary <= 30000;
```

- Useful, though, for a range of values.

Example 13.8 Set Membership

List all Managers and Deputy Managers.

```
SELECT sno, fname, lname, position
FROM staff
WHERE position IN ('Manager', 'Deputy');
```

Example 13.8 Set Membership

Table 13.8 Result table for Example 13.8.

sno	*fname*	*lname*	*position*
SL21	John	White	Manager
SG14	David	Ford	Deputy
SG5	Susan	Brand	Manager

(3 rows)

Example 13.8 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 sno, fname, lname, position
FROM staff
WHERE position='Manager' OR position='Deputy';
```

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

Example 13.9 Pattern Matching

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

```
SELECT sno, fname, lname, address, salary
FROM staff
WHERE address LIKE '%Glasgow%';
```


Example 13.9 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 13.10 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 rno, date
FROM viewing
WHERE pno = 'PG4' AND comment IS NULL;
```


Example 13.11 Single Column Ordering

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

```
SELECT sno, fname, lname, salary
FROM staff
ORDER BY salary DESC;
```

Example 13.11 Single Column Ordering

Table 13.11 Result table for Example 13.11.

sno	*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

(6 rows)

Example 13.12 Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

```
SELECT pno, type, rooms, rent
FROM property_for_rent
ORDER BY type;
```

Example 13.12 Multiple Column Ordering

Table 13.12(a) Result table for Example 13.12 with one sort key.

pno	*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

(6 rows)

Example 13.12 Multiple Column Ordering

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

```
SELECT pno, type, rooms, rent
FROM property_for_rent
ORDER BY type, rent DESC;
```

SELECT Statement - Aggregates

- ISO standard defines five aggregate functions:
  - COUNT, SUM, AVG, MIN, MAX
- Each operates on a single column of a table and return single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG only work on numeric fields.
- Apart from COUNT(*), each function 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.
- Aggregate functions can be used only in SELECT list and in HAVING clause.

SELECT Statement - Aggregates

- **If SELECT list includes an aggregate function and there is no GROUP BY clause, then SELECT list cannot reference a column outwith an aggregate function. For example, following is illegal:**

```
SELECT sno, COUNT(salary)
FROM staff;
```


Example 13.13 Use of COUNT(*)

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

```
SELECT COUNT(*) AS count
FROM property_for_rent
WHERE rent > 350;
```

Table 13.13 Result table for Example 13.13.

count
5

(1 row)

Example 13.14 Use of COUNT(DISTINCT)

How many different properties viewed in May '98?

```
SELECT COUNT(DISTINCT pno) AS count
FROM viewing
WHERE date BETWEEN DATE'1998-05-01'
 AND DATE'1998-05-31';
```

Table 13.14 Result table for Example 13.14.

count
2

(1 row)

Example 13.15 Use of COUNT and SUM

Find number of Managers and sum of their salaries.

```
SELECT COUNT(sno) AS count,
       SUM(salary) AS sum
FROM staff
WHERE position = 'Manager';
```

Table 13.15 Result table for Example 13.15.

count	*sum*
2	54000.00

(1 row)

Example 13.16 Use of MIN, MAX, AVG

Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS min,
  MAX(salary) AS max,
  AVG(salary) AS avg
FROM staff;
```

Table 13.15 Result table for Example 13.15.

count	*sum*
2	54000.00

(1 row)

SELECT Statement - Grouping

- **Use GROUP BY clause to get sub-totals.**
- **SELECT and GROUP BY closely integrated: each item in SELECT list must be *single-valued per group*, and SELECT clause may only contain:**
  - **Column names.**
  - **Aggregate functions.**
  - **Constants.**
  - **An expression involving combinations of the above.**

SELECT Statement - Grouping

- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.

Example 13.17 Use of GROUP BY

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

```
SELECT    bno, COUNT(sno) AS count,
          SUM(salary) AS sum
FROM staff
GROUP BY bno
ORDER BY bno;
```

Example 13.17 Use of GROUP BY

Table 13.17 Result table for Example 13.17.

bno	*count*	*sum*
B3	3	54000.00
B5	2	39000.00
B7	1	9000.00

(3 rows)

Restricted Grouping

- **HAVING clause is designed for use with GROUP BY clause 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 13.18 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 bno, COUNT(sno) AS count,
       SUM(salary) AS sum
FROM staff
GROUP BY bno
HAVING COUNT(sno) > 1
ORDER BY bno;
```


Example 13.18 Use of HAVING

Table 13.18 Result table for Example 13.18.

bno	*count*	*sum*
B3	3	54000.00
B5	2	39000.00

(2 rows)

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 DELETEs.**

Example 13.19 Subquery with Equality

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

```
SELECT sno, fname, lname, position
FROM staff
WHERE bno =
      (SELECT bno
       FROM branch
       WHERE street = '163 Main St');
```

Example 13.19 Subquery with Equality

- **Inner SELECT finds branch number corresponding to branch at '163 Main St' ('B3').**
- **Outer SELECT then retrieves details of all staff who work at this branch.**
- **Outer SELECT then becomes:**

```
SELECT sno, fname, lname, position
FROM staff
WHERE bno = 'B3';
```


Example 13.19 Subquery with Equality

Table 13.19 Result table for Example 13.19.

sno	*fname*	*lname*	*position*
SG37	Ann	Beech	Snr Asst
SG14	David	Ford	Deputy
SG5	Susan	Brand	Manager

(3 rows)

Example 13.20 Subquery with Aggregate

List all staff whose salary is greater than the average salary.

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary >
        (SELECT avg(salary)
         FROM staff);
```


Example 13.20 Subquery with Aggregate

- Cannot write 'WHERE salary > avg(salary)'.
- Instead, use subquery to find average salary (17000), and then use outer SELECT to find those staff with salary greater than this:

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > 17000;
```

Example 13.20 Subquery with Aggregate

Table 13.20 Result table for Example 13.20.

sno	*fname*	*lname*	*position*	*sal_diff*
SL21	John	White	Manager	13000.00
SG14	David	Ford	Deputy	1000.00
SG5	Susan	Brand	Manager	7000.00

(3 rows)

Subquery Rules

- **ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).**
- **Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.**
- **By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an *alias*.**

Subquery Rules

- **When subquery is an operand in a comparison, subquery must appear on right-hand side.**
- **A subquery may not be used as an operand in an expression.**

Example 13.21 Nested subquery: use of IN

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

```
SELECT pno, street, area, city, pcode, type, rooms, rent
FROM property_for_rent
WHERE sno IN
        (SELECT sno
         FROM staff
         WHERE bno =
                (SELECT bno
                 FROM branch
                 WHERE street = '163 Main St'));
```

Example 13.21 Nested subquery: use of IN

Table 13.21 Result table for Example 13.21.

pno	*street*	*area*	*city*	*pcode*	*type*	*rooms*	*rent*
PG4	6 Lawrence St	Partick	Glasgow	G11 9QX	Flat	3	350
PG16	5 Novar Dr	Hyndland	Glasgow	G12 9AX	Flat	4	450
PG36	2 Manor Rd		Glasgow	G32 4QX	Flat	3	375
PG21	18 Dale Rd	Hyndland	Glasgow	G12	House	5	600

(4 rows)

ANY and ALL

- **ANY and ALL may be used with subqueries that produce a single column of numbers and returns a boolean value if ANY or ALL of the results satifgy the condition.**
- **If subquery is empty, ALL returns true, ANY returns false (recall: Predicate Logic).**
- **ISO standard allows SOME to be used in place of ANY.**

Example 13.22 Use of ANY/SOME

Find staff whose salary is larger than salary of at least 1 member of staff at branch B3.

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > SOME
            (SELECT salary
             FROM staff
             WHERE bno = 'B3');
```


Example 13.22 Use of ANY/SOME

- **Inner query produces set {12000, 18000, 24000} and outer query selects those staff whose salaries are greater than any of the values in this set.**

Table 13.22 Result table for Example 13.22.

sno	*fname*	*lname*	*position*	*salary*
SL21	John	White	Manager	30000.00
SG14	David	Ford	Deputy	18000.00
SG5	Susan	Brand	Manager	24000.00

(3 rows)

Example 13.23 Use of ALL

Find staff whose salary is larger than salary of every member of staff at branch B3.

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > ALL
                (SELECT salary
                 FROM staff
                 WHERE bno = 'B3');
```


Example 13.23 Use of ALL

Table 13.23 Result table for Example 13.23.

sno	*fname*	*lname*	*position*	*salary*
SL21	John	White	Manager	30000.00

(1 row)

Multi-Table Queries

- **Can use subqueries provided result columns come from same table.**
- **If result columns come from more than one table must use a join.**
- **To perform join, include more than one table in FROM clause.**
- **Use comma as separator and typically include WHERE clause to specify join column(s).**

Multi-Table Queries

- **Also possible to use an alias for a table named in FROM clause.**
- **Alias is separated from table name with a space.**
- **Alias can be used to qualify column names when there is ambiguity.**

Example 13.24 Simple Join

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

```
SELECT r.rno, fname, lname, pno, comment
FROM renter r, viewing v
WHERE r.rno = v.rno;
```


Example 13.24 Simple Join

- **To obtain correct rows, include only those rows from both tables that have identical values in the Rno columns: r.rno = v.rno.**
- **These two columns are the matching columns for two tables.**
- **Equivalent to equi-join in relational algebra.**

Example 13.24 Simple Join

Table 13.24 Result table for Example 13.24.

rno	*fname*	*lname*	*pno*	*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

(5 rows)

Alternative JOIN Constructs

- **SQL2 provides alternative ways to specify joins:**

  FROM renter r JOIN viewing v ON r.rno = v.rno
  FROM renter JOIN viewing USING rno
  FROM renter NATURAL JOIN viewing

- **In each case, FROM replaces original FROM and WHERE. However, first produces table with two identical Rno columns, remaining two produce table with single Rno column.**

Example 13.25 Sorting a join

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

```
SELECT s.bno, s.sno, fname, lname, pno
FROM staff s, property_for_rent p
WHERE s.sno = p.sno
ORDER BY s.bno, s.sno, pno;
```


Example 13.25 Sorting a join

Table 13.25(b) Result table for Example 13.25 sorted on bno, sno, pno.

bno	*sno*	*fname*	*lname*	*pno*
B3	SG14	David	Ford	PG16
B3	SG14	David	Ford	PG4
B3	SG37	Ann	Beech	PG21
B3	SG37	Ann	Beech	PG36
B5	SL41	Julie	Lee	PL94
B7	SA9	Mary	Howe	PA14

(6 rows)

Example 13.26 Three Table Join

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

```
SELECT b.bno, b.city, s.sno, fname, lname, pno
FROM branch b, staff s, property_for_rent p
WHERE b.bno = s.bno AND s.sno = p.sno
ORDER BY b.bno, s.sno, pno;
```


Example 13.26 Three Table Join

Table 13.26 Result table for Example 13.26.

bno	*city*	*sno*	*fname*	*lname*	*pno*
B3	Glasgow	SG14	David	Ford	PG16
B3	Glasgow	SG14	David	Ford	PG4
B3	Glasgow	SG37	Ann	Beech	PG21
B3	Glasgow	SG37	Ann	Beech	PG36
B5	London	SL41	Julie	Lee	PL94
B7	Aberdeen	SA9	Mary	Howe	PA14

(6 rows)

Example 13.26 Three Table Join

- SQL2 provides alternative formulations for FROM and WHERE:

```
FROM (branch b JOIN staff s USING bno) AS
bs JOIN property_for_rent p USING sno
```


Example 13.27 Multiple Grouping Columns

Find number of properties handled by each staff member in each branch.

```
SELECT s.bno, s.sno, COUNT(*) AS count
FROM staff s, property_for_rent p
WHERE s.sno = p.sno
GROUP BY s.bno, s.sno
ORDER BY s.bno, s.sno;
```

Example 13.27 Multiple Grouping Columns

Table 13.27(a) Result table for Example 13.27.

bno	*sno*	*count*
B3	SG14	2
B3	SG37	2
B5	SL41	1
B7	SA9	1

(4 rows)

Computing a Join

Procedure for generating results of a SELECT with 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 the value of each item in the SELECT list to produce a single row in the result table.**

Computing a Join

4. If SELECT DISTINCT has been specified, eliminate any duplicate rows from the result table.
5. If there is an ORDER BY clause, sort the result table as required.

Computing a Join

- SQL92 provides special format of SELECT for Cartesian product:

```
SELECT [DISTINCT | ALL]      {* | column_list}
  FROM  table1 CROSS JOIN table2
```

Outer Joins

- **With a join, if one row of a table is unmatched, row is omitted from result table.**
- **The outer join operations retain rows that do not satisfy the join condition.**
- **Consider following two simplified tables:**

BRANCH1

bno	city
B3	Glasgow
B4	Bristol
B2	London

PROPERTY_FOR_RENT1

pno	pcity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Outer Joins

- **The (inner) join of these two tables:**

```
SELECT b.*, p.*
FROM branch1 b, property_for_rent1 p
WHERE b.bcity = p.pcity;
```

Table 13.27(b) Result table for inner join of simplified Branch and Property_for_Rent.

bno	*bcity*	*pno*	*pcity*
B3	Glasgow	PG4	Glasgow
B2	London	PL94	London

(2 rows)

Outer Joins

- Result table has two rows where the cities are the same.
- There are no rows corresponding to branches in Bristol and Aberdeen.
- To include unmatched rows in result table, use an outer join.

Example 13.28 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
     property_for_rent1 p ON b.bcity = p.pcity;
```

Example 13.28 Left Outer Join

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

Table 13.28 Result table for Example 13.28.

bno	*bcity*	*pno*	*pcity*
B3	Glasgow	PG4	Glasgow
B4	Bristol	NULL	NULL
B2	London	PL94	London

(3 rows)

Example 13.29 Right Outer Join

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

```
SELECT b.*, p.*
FROM branch1 b RIGHT JOIN
  property_for_rent1 p ON b.bcity = p.pcity;
```

Example 13.29 Right Outer Join

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

Table 13.29 Result table for Example 13.29.

bno	*bcity*	*pno*	*pcity*
NULL	NULL	PA14	Aberdeen
B3	Glasgow	PG4	Glasgow
B2	London	PL94	London

(3 rows)

Example 13.30 Full Outer Join

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

```
SELECT b.*, p.*
FROM branch1 b FULL JOIN
  property_for_rent1 p ON b.bcity = p.pcity;
```

Example 13.30 Full Outer Join

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

Table 13.30 Result table for Example 13.30.

bno	*bcity*	*pno*	*pcity*
NULL	NULL	PA14	Aberdeen
B3	Glasgow	PG4	Glasgow
B4	Bristol	NULL	NULL
B2	London	PL94	London

(4 rows)

EXISTS and NOT EXISTS

- **EXISTS and NOT EXISTS are for use only with subqueries.**
- **They produce a simple true/false result.**
- **EXISTS is true if and only if there exists at least one row in result table returned by subquery.**
- **It is false if subquery returns an empty result table.**
- **NOT EXISTS is the opposite of EXISTS.**

EXISTS and NOT EXISTS

- Since EXISTS and 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 13.31 Query using EXISTS

- Find all staff who work in a London branch.

```
SELECT sno, fname, lname, position
FROM staff s
WHERE EXISTS
 (SELECT *
  FROM branch b
  WHERE s.bno = b.bno AND city = 'London');
```

Example 13.31 Query using EXISTS

Table 13.31 Result table for Example 13.31.

sno	*fname*	*lname*	*position*
SL21	John	White	Manager
SL41	Julie	Lee	Assistant

(2 rows)

Example 13.31 Query using EXISTS

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

  SELECT * FROM branch WHERE city='London'

- **would be always be true and query would be:**

  SELECT sno, fname, lname, position FROM staff WHERE true;

Example 13.31 Query using EXISTS

- which is equivalent to:

```
SELECT sno, fname, lname, position FROM staff;
```

- Could also have written this query using join construct:

```
SELECT sno, fname, lname, position
FROM staff s, branch b
WHERE s.bno = b.bno AND city = 'London';
```


Union, Intersect, and Difference (Except)

- Can use normal set operations of union, intersection, and difference to combine results of two or more queries into a single result table.
- Union of two tables, A and B, is table containing all rows in either A or B or both.
- Intersection is table containing all rows common to both A and B.
- Difference is table containing all rows in A but not in B.
- Two tables must be *union compatible*.

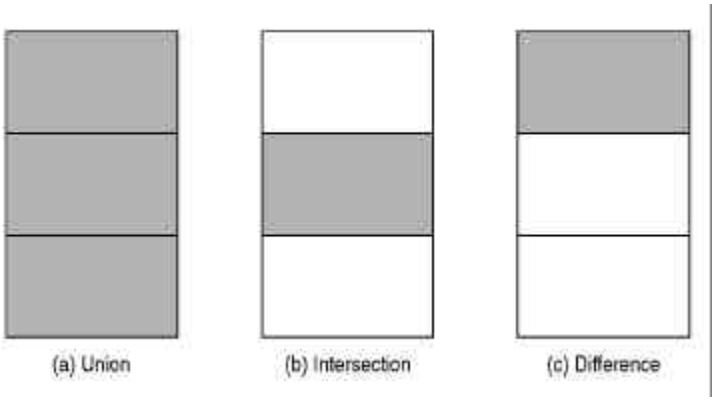
Union, Intersect, and Difference (Except)

- **Format of set operator clause in each case is:**

op **[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.**

Union, Intersect, and Difference (Except)



Example 13.32 Use of UNION

List all areas where there is either a branch or rental property.

```
(SELECT area
FROM branch
WHERE area IS NOT NULL) UNION
(SELECT area
FROM property_for_rent
WHERE area IS NOT NULL);
```


Example 13.32 Use of UNION

- **Or**

```
(SELECT *
  FROM branch
  WHERE area IS NOT NULL)
  UNION CORRESPONDING BY area
  (SELECT *
  FROM property_for_rent
  WHERE area IS NOT NULL);
```

Example 13.32 Use of UNION

- **Produces result tables from both queries and merges both tables together.**

Table 13.32 Result table for Example 13.32.

area
Sidcup
Dyce
Partick
Leigh
Dee
Kilburn
Hyndland

(7 rows)

Example 13.33 Use of INTERSECT

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

```
(SELECT city FROM branch) INTERSECT
(SELECT city FROM property_for_rent);
```

- **Or**

```
(SELECT * FROM branch)
INTERSECT CORRESPONDING BY city
(SELECT * FROM property_for_rent);
```

Example 13.33 Use of INTERSECT

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

Table 13.33 Result table for Example 13.33.

city
Aberdeen
Glasgow
London

(3 rows)

Example 13.33 Use of INTERSECT

- **Could rewrite this query without INTERSECT operator:**

```
SELECT city
FROM branch b property_for_rent p
WHERE b.city = p.city;
```

Example 13.33 Use of INTERSECT

- Or

```
SELECT distinct city
FROM branch b
WHERE EXISTS
 (SELECT *
 FROM property_for_rent p
 WHERE p.city = b.city);
```

- Ability to write a query in several equivalent forms is one of the disadvantages of SQL.

Example 13.34 Use of EXCEPT

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

```
(SELECT city FROM branch)  EXCEPT
(SELECT city FROM property_for_rent);
```

- Or

```
(SELECT * FROM branch)
EXCEPT CORRESPONDING BY city
(SELECT * FROM property_for_rent);
```

Example 13.34 Use of EXCEPT

- **Produces result tables from both queries and then creates single result table consisting of those rows appearing in first result table but not in second.**

Table 13.34

city
Bristol

(1 row)

Example 13.34 Use of EXCEPT

- **Could rewrite this query without EXCEPT:**

```
SELECT distinct city
FROM branch
WHERE city NOT IN
        (SELECT city
        FROM property_for_rent);
```

Example 13.34 Use of EXCEPT

- **Or**

```
SELECT distinct city
FROM branch b
WHERE NOT EXISTS
 (SELECT *
 FROM property_for_rent p
 WHERE p.city = b.city);
```


INSERT

```
INSERT INTO table_name [ (column_list) ]
VALUES (data_value_list)
```

- ***column_list*** **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

- ***data_value_list*** **must match** ***column_list*** **as follows:**
- **Number of items in each list must be the same.**
- **Must be direct correspondence in position of items in two lists.**
- **Data type of each item in** ***data_value_list*** **must be compatible with data type of corresponding column.**

Example 13.35 INSERT … VALUES

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

```
INSERT INTO staff
VALUES ('SG16', 'Alan', 'Brown',
    '67 Endrick Rd, Glasgow G32 8QX',
    '0141-211-3001', 'Assistant', 'M', '25-May-57',
    8300, 'WN848391H', 'B3');
```

Example 13.36 INSERT using Defaults

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

```
INSERT INTO staff (sno, fname, lname, position,
                   salary, bno)
  VALUES ('SG44', 'Anne', 'Jones', 'Assistant',
          8100, 'B3');
```


Example 13.36 INSERT using Defaults

- **Or**

```
INSERT INTO staff
VALUES ('SG44', 'Anne', 'Jones', NULL, NULL,
  'Assistant', NULL, NULL, 8100, NULL, 'B3');
```

INSERT … SELECT

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

INSERT INTO table_name [(column_list)]
SELECT ...

Example 13.37 INSERT … SELECT

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

Staff_Prop_Count(sno, fname, lname, prop_cnt)

Populate Staff_Prop_Count using Staff and Property_for_Rent.

Example 13.37 INSERT … SELECT

```
INSERT INTO staff_prop_count
  (SELECT s.sno, fname, lname, COUNT(*)
  FROM staff s, property_for_rent p
  WHERE s.sno = p.sno
  GROUP BY s.sno, fname, lname)
  UNION
  (SELECT sno, fname, lname, 0
  FROM staff
  WHERE sno NOT IN
    (SELECT DISTINCT sno
    FROM property_for_rent));
```


Example 13.37 INSERT … SELECT

Table 13.35 Result table for Example 13.37.

sno	*fname*	*lname*	*prop_count*
SG14	David	Ford	2
SL21	John	White	0
SG37	Ann	Beech	2
SA9	Mary	Howe	1
SG5	Susan	Brand	0
SL41	Julie	Lee	1

(6 rows)

- **If second part of UNION is omitted, excludes those staff who currently do not manage any properties.**

UPDATE

UPDATE table_name
SET column_name1 = data_value1
[, column_name2 = data_value2...]
[WHERE search_condition]

- ***table_name*** **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 *search_condition* are updated.**
- **New *data_value(s)* must be compatible with data type for corresponding column.**

Example 13.38 UPDATE 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';
```

- WHERE clause finds rows that contain data for Managers. Update is applied only to these particular rows.

Example 13.40 UPDATE Multiple Columns

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

```
UPDATE staff
SET position = 'Manager', salary = 18000
WHERE sno = 'SG14';
```

DELETE

DELETE FROM table_name
[WHERE search_condition]

- ***table_name*** **can be name of a base table or an updatable view.**
- ***search_condition*** **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 13.41 DELETE Rows/Tables

Delete all viewings that relate to property PG4.

```
DELETE FROM viewing
WHERE pno = 'PG4';
```

Delete all records from the Viewing table.

```
DELETE FROM viewing;
```

ISO SQL Data Types

Table 13.36 ISO SQL data types.

Data type	*Declarations*			
character	CHAR,	VARCHAR		
bit	BIT,	BIT VARYING		
exact numeric	NUMERIC,	DECIMAL,	INTEGER,	SMALLINT
approximate numeric	FLOAT,	REAL,	DOUBLE PRECISION	
datetime	DATE,	TIME,	TIMESTAMP	
interval	INTERVAL			

Data Definition

- **In SQL92, relations and other database objects exist in an *environment.***
- **Each environment contains one or more *catalogs*, and each catalog consists of set of schemas.**
- **Schema is a named collection of related database objects.**
- **Objects in a schema can be tables, views, domains, assertions, collations, translations, and character sets. All have same owner.**

Data Definition

CREATE SCHEMA [name |
AUTHORIZATION creator_id]
DROP SCHEMA name [RESTRICT | CASCADE]

- **With RESTRICT (default), schema must be empty or operation fails.**
- **With CASCADE, operation cascades to drop all objects associated with schema in the order defined above. If any of these operations fail, DROP SCHEMA fails.**

CREATE TABLE (Basic)

CREATE TABLE table_name
(col_name data_type [NULL | NOT NULL] [,...])

- **Creates a table with one or more columns of the specified *data_type*.**
- **NULL (default) indicates whether column can contain *nulls*.**
- **With NOT NULL, system rejects any attempt to insert a null in the column.**

CREATE TABLE (Basic)

- **Primary keys should always be specified as NOT NULL.**
- **Foreign keys are often (but not always) candidates for NOT NULL.**

Example 13.43 CREATE TABLE

```
CREATE TABLE staff(
        sno         VARCHAR(5)      NOT NULL,
        fname       VARCHAR(15)     NOT NULL,
        lname       VARCHAR(15)     NOT NULL,
        address     VARCHAR(50),
        tel_no      VARCHAR(13),
        position    VARCHAR(10)     NOT NULL,
        sex         CHAR,
        dob         DATETIME,
        salary      DECIMAL(7,2)    NOT NULL,
        nin         CHAR(9),
        bno         VARCHAR(3)      NOT NULL);
```

Example 13.43 CREATE TABLE

```
CREATE TABLE property_for_rent(
        pno         VARCHAR(5)      NOT NULL,
        street      VARCHAR(25)     NOT NULL,
        area        VARCHAR(15),
        city        VARCHAR(15)     NOT NULL,
        pcode       VARCHAR(8),
        type        CHAR(1)         NOT NULL,
        rooms       SMALLINT        NOT NULL,
        rent        DECIMAL(6,2)    NOT NULL,
        ono         VARCHAR(5)      NOT NULL,
        sno         VARCHAR(5),
        bno         VARCHAR(3)      NOT NULL);
```


DROP TABLE

DROP TABLE tbl_name [RESTRICT | CASCADE]

e.g. DROP TABLE property_for_rent;

- **Removes named table and all rows within it.**
- **With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.**
- **With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).**