STRUCTURED QUERY LANGUAGE (SQL)

EGCO321 DATABASE SYSTEMS

KANAT POOLSAWASD
DEPARTMENT OF COMPUTER ENGINEERING
MAHIDOL UNIVERSITY

SQL TIMELINE

Year	Event
1972	System R project at IBM Research Labs
1974	SQUARE language developed
1975	Language revision and name change to SEQUEL
1976	Language revision and name change to SEQUEL 2
1977	Name change to SQL
1978	First commercial implementation by Oracle Corporation
1981	IBM product SQL/DS featuring SQL
1986	SQL-86 (SQL1) standard approved
1989	SQL-89 standard approved (revision to SQL-86)
1992	SQL-92 (SQL2) standard approved
1999	SQL:1999 (SQL3) standard approved
2003	SQL:2003 approved

SCOPE OF SQL

Selected SQL Statements

Statement Type	Statements	Purpose
Database definition	CREATE SCHEMA, TABLE, VIEW	Define a new database, table, and view
	ALTER TABLE	Modify table definition
Database manipulation	SELECT	Retrieve contents of tables
	UPDATE, DELETE, INSERT	Modify, remove, and add rows
Database control	COMMIT, ROLLBACK	Complete, undo transaction
	GRANT, REVOKE	Add and remove access rights
	CREATE ASSERTION	Define integrity constraint
	CREATE TRIGGER	Define database rule

THE ISO SQL DATA TYPES

- SQL identifiers are used to identify objects in the database, such as table names, view names, and columns.
- The characters that can be used in a user-defined SQL identifier must appear in a character set.

SOME COMMON SQL DATA TYPE

DATA TYPE	FORMAT	COMMENTS
Numeric	NUMBER(L,D)	The declaration NUMBER(7,2) indicates numbers that will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place. Examples: 12.32, −134.99.
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER, but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a *minimum* specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as Smith and Katzenjammer are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. The designation VARCHAR2(25) will let you store characters up to 25 characters long. However, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.
Date	DATE	Stores dates in the Julian date format.

DATA DEFINITION LANGUAGE (DDL)

- Modification anomaly is an unexpected side effect that occurs when changing the data in a table with excessive redundancies.
- To understand more precisely the impact of modification anomalies, let us consider a poorly design database. Imagine that a university database consist of the single table show in this table.

CREATING THE DATABASE AND TABLE

- The database is a shared, integrated computer structure that houses a collection of end-user data and metadata.
- The end-user data and metadata are stored in tables.

```
CREATE SCHEMA 'database-name';

CREATE TABLE 'table-name' (
    'col-name-1' CHAR(10) NOT NULL,
    'col-name-2' INT,
    PRIMARY KEY ('col-name-1'));
```

CHANGING A TABLE DEFINITION (1)

- The ISO standard provides an ALTER TABLE statement for changing the structure of a table once it has been created. The definition of the ALTER TABLE statement in the ISO standard consists of six options to:
  - Add a new column to a table.
  - Drop a column from a table.
  - Add a new table constraint.
  - Drop a table constraint.
  - Set a default for a column.
  - Drop a default for a column.

CHANGING A TABLE DEFINITION (2)

```
ALTER TABLE TableName
[ADD [COLUMN] columnName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption] [CHECK (searchCondition)]]
[DROP [COLUMN] columnName [RESTRICT | CASCADE]]
[ADD [CONSTRAINT [ConstraintName]] tableConstraintDefinition]
[DROP CONSTRAINT ConstraintName [RESTRICT | CASCADE]]
[ALTER [COLUMN] SET DEFAULT defaultOption]
[ALTER [COLUMN] DROP DEFAULT]
```

(a) *Change the* Staff *table by removing the default of 'Assistant' for the* position *column and setting the default for the* sex *column to female ('F').*

```
ALTER TABLE Staff
    ALTER position DROP DEFAULT;
ALTER TABLE Staff
    ALTER sex SET DEFAULT 'F';
```

(b) *Change the* PropertyForRent *table by removing the constraint that staff are not allowed to handle more than 100 properties at a time. Change the* Client *table by adding a new column representing the preferred number of rooms.*

```
ALTER TABLE PropertyForRent
    DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client
    ADD prefNoRooms PropertyRooms;
```

REMOVING A TABLE

- We can remove a redundant table from the database using the DROP TABLE statement, which has the format:
  - DROP TABLE TableName [RESTRICT | CASCADE]
- For Example:
  - DROP TABLE PropertyForRent

CREATING & REMOVING AN INDEX

- An index is a structure that provides accelerated access to the rows of a table based on the values of one or more columns
- The creation of indexes is not standard SQL. However, most dialects support at least the following capabilities:
  - CREATE [UNIQUE] INDEX IndexName ON TableName (columnmName [ASC | DESC] [,…])
  - DROP INDEX IndexName

DATA MANIPULATION LANGUAGE (DML)

- This section looks at the SQL DML statements, namely:
  - SELECT – to query data in the database.
  - INSERT – to insert data into a table.
  - UPDATE – to update data in a table.
  - DELETE – to delete data from a table.

SIMPLE QUERY

- The purpose of the SELECT statement is to retrieve and display data from one or more database tables.
- It is an extremely powerful command capable of performing the equivalent of the relational algebra's *Selection*, *Projection*, and *Join* operations in a single statement
- SELECT is the most frequently used SQL command and has the following general form:

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

EXAMPLE 1

- Retrieve all columns, all rows

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

- A quick way of expressing 'all columns' in SQL, using an asterisk (*) in place of the column names.

```
SELECT * FROM Staff;
```

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 2

- Retrieve specific columns, all rows

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

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 3

- Use of DISTINCT

```
SELECT propertyNo FROM Viewing;

SELECT DISTINCT propertyNo FROM Viewing;
```

Without Distinct

propertyNo
PA14
PG4
PG4
PA14
PG36

With Distinct

propertyNo
PA14
PG4
PG36

EXAMPLE 4 (1)

- Calculated fields (1)
- *"Produce a list of monthly salaries for all staff, showing the staff number, the first and last names, and the salary details."*

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

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 4 (2)

- Calculated fields (2)

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

ROW SELECTION (WHERE)

- However, 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.
- The five basic search conditions are as follows:
  - Comparison
  - Range
  - Set Membership
  - Pattern Match
  - Null

EXAMPLE 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

COMPARISON OPERATORS

- In SQL, the following simple comparison operators are available:

=	equals		
<>	is not equal to (ISO standard)	!=	is not equal to (allowed in some dialects)
<	is less than	<=	is less than or equal to
>	is greater than	>=	is greater than or equal to

- More complex predicates can be generated using the logical operators AND, OR, and NOT

EXAMPLE 6

- Compound comparison search condition
  "List the 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 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;
```

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

EXAMPLE 8

- Set membership search condition (IN/NOT IN)

  "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 9 (1)

- Pattern match search condition (LIKE/NOT LIKE)
  "Find all owners with the string 'Glasgow' in their address."

  - % percent character represents any sequence of zero or more characters (*wildcard*).
  - _ underscore character represents any single character.

  All other characters in the pattern represent themselves. For example:

  - address LIKE 'H%' means the first character must be *H*, but the rest of the string can be anything.
  - address LIKE 'H_ _ _' means that there must be exactly four characters in the string, the first of which must be an *H*.
  - address LIKE '%e' means any sequence of characters, of length at least 1, with the last character an *e*.
  - address LIKE '%Glasgow%' means a sequence of characters of any length containing *Glasgow*.
  - address NOT LIKE 'H%' means the first character cannot be an *H*.

EXAMPLE 9 (2)

"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 10

- NULL search condition (IS NULL/IS NOT NULL)
  "List the 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;
```

clientNo	viewDate
CR56	26-May-04

SORTING RESULTS (ORDER BY)

- 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 in the SELECT statement.
- The ORDER BY clause consists of a list of column identifiers that the result is to be sorted on, separated by commas.

EXAMPLE 11

- Single-column ordering
  "Produce a list of salaries for all staff, arranged in descending order of salary. "

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

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 12 (1)

- Multiple column ordering
  "Produce an abbreviated list of properties arranged in order of property type. "

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

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 12 (2)

- Multiple column ordering
  "Produce an abbreviated list of properties arranged in order of property type and rent."

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

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

SQL AGGREGATE FUNCTIONS

- As well as retrieving rows and columns from the database, we often want to perform some form of summation or aggregation of data, similar to the totals at the bottom of a report.
- The standard defines five aggregate functions:
  - COUNT
  - SUM
  - AVG
  - MIN
  - MAX

EXAMPLE 13

- Use of COUNT(*)

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

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

EXAMPLE 14

- Use of COUNT(DISTINCT)

  "How many different properties were viewed in May 2004?"

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
```

EXAMPLE 15

- Use of COUNT and SUM

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

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

myCount	mySum
2	54000.00

EXAMPLE 16

- Use of MIN, MAX, AVG

  "Find the 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

GROUPING RESULTS (GROUP BY)

- A query that includes the GROUP BY clause is called a *grouped query*, because it groups the data from the SELECT table(s) and produces a single summary row for each group.
- The columns named in the GROUP BY clause are called the *grouping columns*.
- When GROUP BY is used, each item in the SELECT list must be *single-valued per group*.

EXAMPLE 17

- Use of GROUP BY

  "Find the number of staff working 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	staffNo	salary
B003	SG37	12000.00
B003	SG14	18000.00
B003	SG5	24000.00
B005	SL21	30000.00
B005	SL41	9000.00
B007	SA9	9000.00

COUNT(staffNo)	SUM(salary)
3	54000.00
2	39000.00
1	9000.00

RESTRICTING GROUPINGS (HAVING)

- The HAVING clause is designed for use with the GROUP BY clause to restrict the groups that appear in the final result table.
- The WHERE clause filters individual rows going into the final result table, whereas HAVING filters groups going into the final result table.

EXAMPLE 18

- Use of HAVING

  "For each branch office with more than one member of staff, find the number of staff working in each branch and the sum of their salaries."

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

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

SUBQUERIES

- In this section we examine the use of a complete SELECT statement embedded within another SELECT statement.
- The results of this inner SELECT statement (or subselect) are used in the outer statement to help determine the contents of the final result.
- A sub- select can be used in the WHERE and HAVING clauses of an outer SELECT statement, where it is called a *subquery* or *nested query*.

EXAMPLE 19

- Using a subquery with equality

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

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

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

EXAMPLE 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)
```

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

EXAMPLE 21

- Nested subqueries: use of IN

  "List the properties that are handled by staff who work in the branch 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'));
```

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

- The words ANY and ALL may be used with subqueries that produce a single column of numbers.
- If the subquery is preceded by the keyword ALL, the condition will only be true if it is satisfied by all values produced by the subquery.
- If the subquery is preceded by the keyword ANY, the condition will be true if it is satisfied by any (one or more) values produced by the subquery.
- If the subquery is empty, the ALL condition returns true, the ANY condition returns false.
- The standard also allows the qualifier SOME to be used in place of ANY.

EXAMPLE 22

- Use of ANY/SOME

  "Find all staff whose salary is larger than the 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');
```

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

EXAMPLE 23

- Use of ALL

  "Find all staff whose salary is larger than the 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');
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00

MULTI-TABLE QUERIES

- All the examples we have considered so far have a major limitation: the columns that are to appear in the result table must all come from a single table.
- In many cases, this is not sufficient. To combine columns from several tables into a result table we need to use a *join* operation.

EXAMPLE 24

- Simple join

  "List the 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;
```

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

EXAMPLE 25

- Sorting a join

  "For each branch office, list the numbers and names of staff who manage properties and the properties that 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

EXAMPLE 26

- Three-table join

  "For each branch, list the numbers and names of staff who manage properties, including the city in which the branch is located and the properties that the staff 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;
```

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

EXAMPLE 27

- Multiple grouping columns

  "Find the 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;
```

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.
- The columns of the product table are all the columns of the first table followed by all the columns of the second table.
- If we specify a two-table query without a WHERE clause, SQL produces the Cartesian product of the two tables as the query result.

```
SELECT   [DISTINCT | ALL] {* | columnList}
FROM     TableName1 CROSS JOIN TableName2
```

EXAMPLE 28

- The (Inner) join of these two tables:

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

```
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

EXAMPLE 29

- 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

EXAMPLE 30

- 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

EXAMPLE 31

- 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;
```

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

EXISTS AND NOT EXISTS

- The keywords EXISTS and NOT EXISTS are designed 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 the result table returned by the subquery; it is false if the subquery returns an empty result table.
- NOT EXISTS is the opposite of EXISTS.
- Since EXISTS and NOT EXISTS check only for the existence or non-existence of rows in the subquery result table, the subquery can contain any number of columns.

EXAMPLE 32

- Query using EXISTS

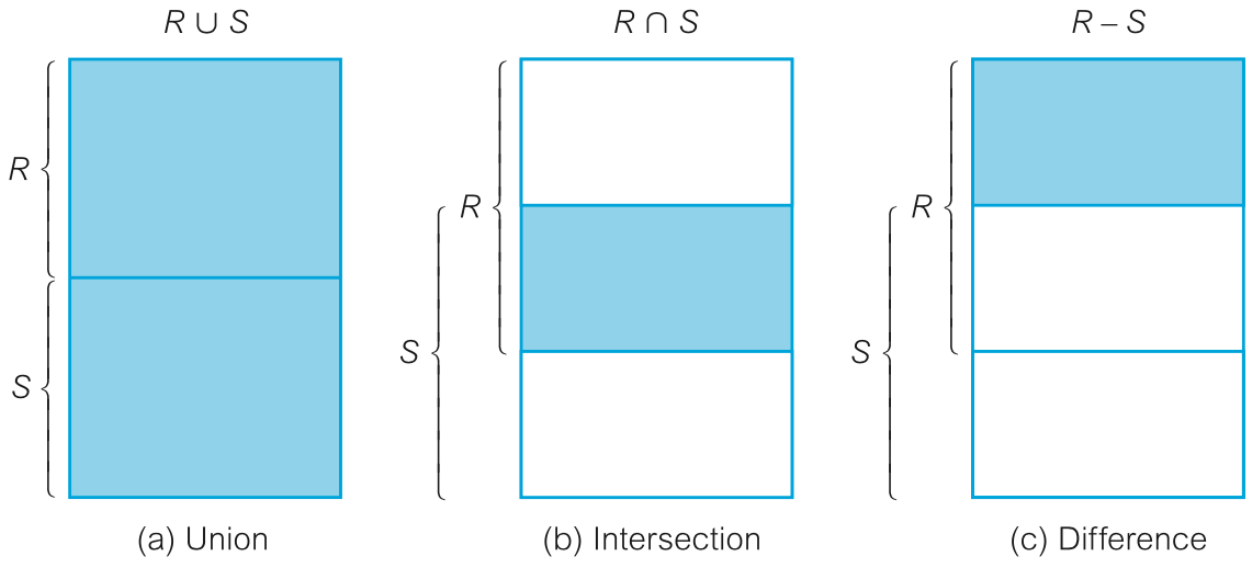
  "Find all staff who work in a London branch office."

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

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

COMBINING RESULT TABLES

- In SQL, we can use the normal set operations of Union, Intersection, and Difference to combine the results of two or more queries into a single result table:
  - UNION
  - INTERSECT
  - EXCEPT (Difference)



EXAMPLE 33

- Use of INTERSECT

  "Construct a list of all cities where there is both a branch office and a property."

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

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

city
Aberdeen
Glasgow
London

EXAMPLE 34

- Use of EXCEPT

  "Construct a list of all cities where there is a branch office but no properties."

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

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

city
Bristol

DATABASE UPDATES

- SQL is a complete data manipulation language that can be used for modifying the data in the database as well as querying the database.
- The commands for modifying the database are not as complex as the SELECT statement.
- In this section, we describe the three SQL statements that are available to modify the contents of the tables in the database:
  - INSERT
  - UPDATE
  - DELETE

ADDING DATA TO THE DATABASE

- There are two forms of the INSERT statement. The first allows a single row to be inserted into a named table and has the following format:

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

EXAMPLE 35

- INSERT … VALUES

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

```
INSERT INTO Staff
VALUES (‘SG16’, ‘Alan’, ‘Brown’, ‘Assistant’,
‘M’, DATE ‘1957-05-25’, 8300, ‘B003’ );
```

EXAMPLE 36

- INSERT using defaults
  "Insert a new row into the Staff table supplying data for all mandatory columns: staffNo, fName, lName, position, salary, and branchNo."

```
INSERT INTO Staff (staffNo, fName, lName,
position, salary, branchNo)
VALUES ('SG44', 'Anne', 'Jones', 'Assistant',
8100, 'B003');
```

EXAMPLE 37

- INSERT … SELECT

  "Populate the StaffPropCount table using details from the Staff and PropertyForRent tables."

```
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 s
 WHERE NOT EXISTS (SELECT *
                   FROM PropertyForRent p
                   WHERE p.staffNo = s.staffNo));
```

MODIFYING DATA IN THE DATABASE

- The UPDATE statement allows the contents of existing rows in a named table to be changed. The format of the command is:

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

EXAMPLE 38

- UPDATE all rows & UPDATE 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 39

- 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';
```

DELETING DATA FROM THE DATABASE

- The DELETE statement allows rows to be deleted from a named table. The format of the command is:

```
DELETE FROM TableName
[WHERE searchCondition]
```

EXAMPLE 40

- DELETE specific rows

  “Delete all viewings that relate to property PG4.“

```
DELETE FROM Viewing WHERE propertyNo = ‘PG4’;
```