

The background of the book cover is a photograph of a desert landscape at sunset or sunrise. The sky is a mix of light blue, purple, and orange, with soft, wispy clouds. The foreground shows the undulating, golden-brown dunes of a desert, with the sun's glow creating a bright, horizontal band of light across the middle of the image.

# Fundamentals of DATABASE SYSTEMS

FOURTH EDITION

ELMASRI  NAVATHE

# Chapter 8

## SQL-99: Schema Definition, Basic Constraints, and Queries



# Data Definition, Constraints, and Schema Changes

- Used to CREATE, DROP, and ALTER the descriptions of the tables (relations) of a database

# CREATE TABLE

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n))
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE DEPARTMENT  
(          DNAME          VARCHAR(10) NOT NULL,  
          DNUMBER      INTEGER      NOT NULL,  
          MGRSSN         CHAR(9),  
          MGRSTARTDATE    CHAR(9) );
```

# CREATE TABLE

- In SQL2, can use the CREATE TABLE command for specifying the primary key attributes, secondary keys, and referential integrity constraints (foreign keys).
- Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases

```
CREATE TABLE DEPT  
( DNAME          VARCHAR(10) NOT NULL,  
  DNUMBER        INTEGER      NOT NULL,  
  MGRSSN         CHAR(9),  
  MGRSTARTDATE   CHAR(9),  
  PRIMARY KEY (DNUMBER),  
  UNIQUE (DNAME),  
  FOREIGN KEY (MGRSSN) REFERENCES EMP );
```

# DROP TABLE

- Used to remove a relation (base table) *and its definition*
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

**DROP TABLE DEPENDENT;**

# ALTER TABLE

- Used to add an attribute to one of the base relations
- The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is *not allowed* for such an attribute
- Example:

```
ALTER TABLE EMPLOYEE ADD JOB  
VARCHAR(12);
```

- The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.

# Features Added in SQL2 and SQL-99

- **CREATE SCHEMA**
- **REFERENTIAL INTEGRITY OPTIONS**



# CREATE SCHEMA

- Specifies a new database schema by giving it a name

# REFERENTIAL INTEGRITY OPTIONS

- We can specify RESTRICT, CASCADE, SET NULL or SET DEFAULT on referential integrity constraints (foreign keys)

```
CREATE TABLE DEPT  
( DNAME VARCHAR(10) NOT NULL,  
  DNUMBER INTEGER NOT NULL,  
  MGRSSN CHAR(9),  
  MGRSTARTDATE CHAR(9),  
  PRIMARY KEY (DNUMBER),  
  UNIQUE (DNAME),  
  FOREIGN KEY (MGRSSN) REFERENCES EMP  
ON DELETE SET DEFAULT ON UPDATE  
CASCADE );
```

# REFERENTIAL INTEGRITY OPTIONS (continued)

```
CREATE TABLE EMP  
(      ENAME      VARCHAR(30) NOT NULL,  
      ESSN CHAR(9),  
      BDATE DATE,  
      DNO  INTEGER DEFAULT 1,  
      SUPERSSN CHAR(9),  
      PRIMARY KEY (ESSN),  
      FOREIGN KEY (DNO) REFERENCES DEPT  
ON DELETE SET DEFAULT ON UPDATE CASCADE,  
      FOREIGN KEY (SUPERSSN) REFERENCES EMP  
ON DELETE SET NULL ON UPDATE CASCADE );
```

# Additional Data Types in SQL2 and SQL-99

Has DATE, TIME, and TIMESTAMP data types

- **DATE:**

- Made up of year-month-day in the format yyyy-mm-dd

- **TIME:**

- Made up of hour:minute:second in the format hh:mm:ss

- **TIME(i):**

- Made up of hour:minute:second plus i additional digits specifying fractions of a second
- format is hh:mm:ss:ii...i

- **TIMESTAMP:**

- Has both DATE and TIME components

# Additional Data Types in SQL2 and SQL-99 (cont.)

## ● **INTERVAL:**

- Specifies a relative value rather than an absolute value
- Can be DAY/TIME intervals or YEAR/MONTH intervals
- Can be positive or negative when added to or subtracted from an absolute value, the result is an absolute value

# Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the SELECT statement
- This is *not the same as* the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model; SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
- Hence, an SQL relation (table) is a *multi-set* (sometimes called a bag) of tuples; it *is not* a set of tuples
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query

# Retrieval Queries in SQL (cont.)

- Basic form of the SQL SELECT statement is called a *mapping* or a *SELECT-FROM-WHERE block*

**SELECT**    <attribute list>  
**FROM**     <table list>  
**WHERE**    <condition>

- <attribute list> is a list of attribute names whose values are to be retrieved by the query
- <table list> is a list of the relation names required to process the query
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

## Relational Database Schema--Figure 5.5

### EMPLOYEE

|       |       |       |            |       |         |     |        |          |     |
|-------|-------|-------|------------|-------|---------|-----|--------|----------|-----|
| FNAME | MINIT | LNAME | <u>SSN</u> | BDATE | ADDRESS | SEX | SALARY | SUPERSSN | DNO |
|-------|-------|-------|------------|-------|---------|-----|--------|----------|-----|

### DEPARTMENT

|       |                |        |              |
|-------|----------------|--------|--------------|
| DNAME | <u>DNUMBER</u> | MGRSSN | MGRSTARTDATE |
|-------|----------------|--------|--------------|

### DEPT\_LOCATIONS

|                |                  |
|----------------|------------------|
| <u>DNUMBER</u> | <u>DLOCATION</u> |
|----------------|------------------|

### PROJECT

|       |                |           |      |
|-------|----------------|-----------|------|
| PNAME | <u>PNUMBER</u> | PLOCATION | DNUM |
|-------|----------------|-----------|------|

### WORKS\_ON

|             |            |       |
|-------------|------------|-------|
| <u>ESSN</u> | <u>PNO</u> | HOURS |
|-------------|------------|-------|

### DEPENDENT

|             |                       |     |       |              |
|-------------|-----------------------|-----|-------|--------------|
| <u>ESSN</u> | <u>DEPENDENT_NAME</u> | SEX | BDATE | RELATIONSHIP |
|-------------|-----------------------|-----|-------|--------------|



# Populated Database--Fig.5.6

| EMPLOYEE | FNAME    | MINIT | LNAME   | SSN       | BDATE      | ADDRESS                  | SEX | SALARY | SUPERSSN  | DNO |
|----------|----------|-------|---------|-----------|------------|--------------------------|-----|--------|-----------|-----|
|          | John     | B     | Smith   | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | M   | 30000  | 333445555 | 5   |
|          | Franklin | T     | Wong    | 333445555 | 1955-12-08 | 638 Voss, Houston, TX    | M   | 40000  | 888665555 | 5   |
|          | Alicia   | J     | Zelaya  | 999887777 | 1968-07-19 | 3321 Castle, Spring, TX  | F   | 25000  | 987654321 | 4   |
|          | Jennifer | S     | Wallace | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX  | F   | 43000  | 888665555 | 4   |
|          | Ramesh   | K     | Narayan | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX | M   | 38000  | 333445555 | 5   |
|          | Joyce    | A     | English | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX   | F   | 25000  | 333445555 | 5   |
|          | Ahmad    | V     | Jabbar  | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX  | M   | 25000  | 987654321 | 4   |
|          | James    | E     | Borg    | 888665555 | 1937-11-10 | 450 Stone, Houston, TX   | M   | 55000  | null      | 1   |
|          |          |       |         |           |            |                          |     |        |           |     |

| DEPARTMENT | DNAME          | DNUMBER | MGRSSN    | MGRSTARTDATE |
|------------|----------------|---------|-----------|--------------|
|            | Research       | 5       | 333445555 | 1988-05-22   |
|            | Administration | 4       | 987654321 | 1995-01-01   |
|            | Headquarters   | 1       | 888665555 | 1981-06-19   |

| DEPT_LOCATIONS | DNUMBER | DLOCATION |
|----------------|---------|-----------|
|                | 1       | Houston   |
|                | 4       | Stafford  |
|                | 5       | Bellaire  |
|                | 5       | Sugarland |
|                | 5       | Houston   |

| WORKS_ON | ESSN      | PNO | HOURS |
|----------|-----------|-----|-------|
|          | 123456789 | 1   | 32.5  |
|          | 123456789 | 2   | 7.5   |
|          | 666884444 | 3   | 40.0  |
|          | 453453453 | 1   | 20.0  |
|          | 453453453 | 2   | 20.0  |
|          | 333445555 | 2   | 10.0  |
|          | 333445555 | 3   | 10.0  |
|          | 333445555 | 10  | 10.0  |
|          | 333445555 | 20  | 10.0  |
|          | 999887777 | 30  | 30.0  |
|          | 999887777 | 10  | 10.0  |
|          | 987987987 | 10  | 35.0  |
|          | 987987987 | 30  | 5.0   |
|          | 987654321 | 30  | 20.0  |
|          | 987654321 | 20  | 15.0  |
|          | 888665555 | 20  | null  |

| PROJECT | PNAME           | PNUMBER | PLOCATION | DNUM |
|---------|-----------------|---------|-----------|------|
|         | ProductX        | 1       | Bellaire  | 5    |
|         | ProductY        | 2       | Sugarland | 5    |
|         | ProductZ        | 3       | Houston   | 5    |
|         | Computerization | 10      | Stafford  | 4    |
|         | Reorganization  | 20      | Houston   | 1    |
|         | Newbenefits     | 30      | Stafford  | 4    |

| DEPENDENT | ESSN      | DEPENDENT_NAME | SEX | BDATE      | RELATIONSHIP |
|-----------|-----------|----------------|-----|------------|--------------|
|           | 333445555 | Alice          | F   | 1986-04-05 | DAUGHTER     |
|           | 333445555 | Theodore       | M   | 1983-10-25 | SON          |
|           | 333445555 | Joy            | F   | 1958-05-03 | SPOUSE       |
|           | 987654321 | Abner          | M   | 1942-02-28 | SPOUSE       |
|           | 123456789 | Michael        | M   | 1988-01-04 | SON          |
|           | 123456789 | Alice          | F   | 1988-12-30 | DAUGHTER     |
|           | 123456789 | Elizabeth      | F   | 1967-05-05 | SPOUSE       |

# Simple SQL Queries

- Basic SQL queries correspond to using the SELECT, PROJECT, and JOIN operations of the relational algebra
- All subsequent examples use the COMPANY database
- Example of a simple query on *one* relation
- Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

**Q0: SELECT                BDATE, ADDRESS  
      FROM               EMPLOYEE  
      WHERE FNAME='John' AND MINIT='B'  
      AND                LNAME='Smith'**

- Similar to a SELECT-PROJECT pair of relational algebra operations; the SELECT-clause specifies the *projection attributes* and the WHERE-clause specifies the *selection condition*
- However, the result of the query *may contain* duplicate tuples

# Simple SQL Queries (cont.)

- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

**Q1: SELECT            FNAME, LNAME, ADDRESS  
         FROM EMPLOYEE, DEPARTMENT  
         WHERE        DNAME='Research' AND  
         DNUMBER=DNO**

- Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
- (DNAME='Research') is a *selection condition* (corresponds to a SELECT operation in relational algebra)
- (DNUMBER=DNO) is a *join condition* (corresponds to a JOIN operation in relational algebra)

# Simple SQL Queries (cont.)

- Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

|                   |   |
|-------------------|---|
| <b>Q2: SELECT</b> | <b>PNUMBER, DNUM, LNAME, BDATE, ADDRESS</b> |
| <b>FROM</b>       | <b>PROJECT, DEPARTMENT, EMPLOYEE</b>        |
| <b>WHERE</b>      | <b>DNUM=DNUMBER AND MGRSSN=SSN</b>          |
| <b>AND</b>        | <b>PLOCATION='Stafford'</b>                 |

- In Q2, there are *two* join conditions
- The join condition DNUM=DNUMBER relates a project to its controlling department
- The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

# Aliases, \* and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*  
A query that refers to two or more attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name

Example:

- EMPLOYEE.LNAME, DEPARTMENT.DNAME

# ALIASES

- Some queries need to refer to the same relation twice
- In this case, *aliases* are given to the relation name
- Query 8: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
Q8:  SELECT      E.FNAME, E.LNAME, S.FNAME,  
                S.LNAME  
      FROM      EMPLOYEE E S  
      WHERE     E.SUPERSSN=S.SSN
```

- In Q8, the alternate relation names E and S are called *aliases* or *tuple variables* for the EMPLOYEE relation
- We can think of E and S as two *different copies* of EMPLOYEE; E represents employees in role of *supervisees* and S represents employees in role of *supervisors*

# ALIASES (cont.)

- Aliasing can also be used in any SQL query for convenience  
Can also use the AS keyword to specify aliases

**Q8:**        **SELECT            E.FNAME, E.LNAME, S.FNAME,**  
                 **S.LNAME**  
**FROM            EMPLOYEE AS E, EMPLOYEE AS S**  
**WHERE           E.SUPERSSN=S.SSN**

# UNSPECIFIED WHERE-clause

- A *missing WHERE-clause* indicates no condition; hence, *all tuples* of the relations in the FROM-clause are selected
- This is equivalent to the condition WHERE TRUE
- Query 9: Retrieve the SSN values for all employees.

**Q9:**            **SELECT        SSN**  
                 **FROM         EMPLOYEE**

- If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected



# UNSPECIFIED WHERE-clause (cont.)

- Example:

**Q10:            SELECT        SSN, DNAME  
                 FROM        EMPLOYEE, DEPARTMENT**

- It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large relations may result

# USE OF \*

- To retrieve all the attribute values of the selected tuples, a \* is used, which stands for *all the attributes*

Examples:

**Q1C:**        **SELECT        \***  
              **FROM         EMPLOYEE**  
              **WHERE       DNO=5**

**Q1D:**        **SELECT        \***  
              **FROM         EMPLOYEE, DEPARTMENT**  
              **WHERE       DNAME='Research' AND**  
                      **DNO=DNUMBER**

# USE OF DISTINCT

- SQL does not treat a relation as a set; *duplicate tuples can appear*
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

Q11:        **SELECT        SALARY**  
             **FROM         EMPLOYEE**

Q11A:      **SELECT      DISTINCT SALARY**  
             **FROM         EMPLOYEE**

# SET OPERATIONS

- SQL has directly incorporated some set operations
- There is a union operation (**UNION**), and in *some versions* of SQL there are set difference (**MINUS**) and intersection (**INTERSECT**) operations
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- The set operations apply only to *union compatible relations* ; the two relations must have the same attributes and the attributes must appear in the same order

# SET OPERATIONS (cont.)

- Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

**Q4: (SELECT PNAME  
FROM PROJECT, DEPARTMENT, EMPLOYEE  
WHERE DNUM=DNUMBER AND MGRSSN=SSN  
AND LNAME='Smith')  
UNION  
(SELECT PNAME  
FROM PROJECT, WORKS\_ON, EMPLOYEE  
WHERE PNUMBER=PNO AND ESSN=SSN AND  
LNAME='Smith')**

# NESTING OF QUERIES

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT      FNAME, LNAME, ADDRESS
      FROM        EMPLOYEE
      WHERE       DNO IN (SELECT DNUMBER
      FROM        DEPARTMENT
      WHERE       DNAME='Research' )
```

# NESTING OF QUERIES (cont.)

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator **IN** compares a value *v* with a set (or multi-set) of values *V*, and evaluates to **TRUE** if *v* is one of the elements in *V*
- In general, we can have several levels of nested queries
- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*
- In this example, the nested query is *not correlated* with the outer query

# CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be *correlated*
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12: SELECT      E.FNAME, E.LNAME
      FROM        EMPLOYEE AS E
      WHERE       E.SSN IN (SELECT      ESSN
                           FROM DEPENDENT
                           WHERE          ESSN=E.SSN AND
                                           E.FNAME=DEPENDENT_NAME)
```



# CORRELATED NESTED QUERIES (cont.)

- In Q12, the nested query has a different result *for each tuple* in the outer query
- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can *always* be expressed as a single block query. For example, Q12 may be written as in Q12A

**Q12A:**

|               |                                 |
|---------------|---------------------------------|
| <b>SELECT</b> | <b>E.FNAME, E.LNAME</b>         |
| <b>FROM</b>   | <b>EMPLOYEE E, DEPENDENT D</b>  |
| <b>WHERE</b>  | <b>E.SSN=D.ESSN AND</b>         |
|               | <b>E.FNAME=D.DEPENDENT_NAME</b> |

- The original SQL as specified for SYSTEM R also had a **CONTAINS** comparison operator, which is used in conjunction with nested correlated queries
- This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently

# CORRELATED NESTED QUERIES (cont.)

- Most implementations of SQL *do not* have this operator
- The CONTAINS operator compares two *sets of values*, and returns TRUE if one set contains all values in the other set (reminiscent of the *division* operation of algebra).
  - Query 3: Retrieve the name of each employee who works on *all* the projects controlled by department number 5.

**Q3:**     **SELECT FNAME, LNAME**  
          **FROM   EMPLOYEE**  
          **WHERE ( (SELECT           PNO**  
                    **FROM WORKS\_ON**  
                    **WHERE           SSN=ESSN)**  
                    **CONTAINS**  
                    **(SELECT           PNUMBER**  
                    **FROM PROJECT**  
                    **WHERE           DNUM=5) )**

# CORRELATED NESTED QUERIES (cont.)

- In Q3, the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5
- The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different *for each employee tuple* because of the correlation

# THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below

# THE EXISTS FUNCTION (cont.)

- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

**Q12B:     SELECT    FNAME, LNAME  
              FROM     EMPLOYEE  
              WHERE    EXISTS   (SELECT   \*  
                                  FROM     DEPENDENT  
                                  WHERE    SSN=ESSN AND  
                                  FNAME=DEPENDENT\_NAME)**

# THE EXISTS FUNCTION (cont.)

- Query 6: Retrieve the names of employees who have no dependents.

**Q6:**

|               |  |
|---------------|--|
| <b>SELECT</b> | <b>FNAME, LNAME</b>  |
| <b>FROM</b>   | <b>EMPLOYEE</b>  |
| <b>WHERE</b>  | <b>NOT EXISTS (SELECT *<br/>FROM DEPENDENT<br/>WHERE SSN=ESSN)</b> |

- In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist*, the EMPLOYEE tuple is selected
- EXISTS is necessary for the expressive power of SQL

# EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

**Q13:**        **SELECT        DISTINCT ESSN**  
              **FROM         WORKS\_ON**  
              **WHERE       PNO IN (1, 2, 3)**

# NULLS IN SQL QUERIES

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)
- SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .
- Query 14: Retrieve the names of all employees who do not have supervisors.

**Q14:            SELECT            FNAME, LNAME  
                 FROM            EMPLOYEE  
                 WHERE          SUPERSSN IS NULL**

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result



# Joined Relations Feature in SQL2

- Can specify a "joined relation" in the FROM-clause
- Looks like any other relation but is the result of a join
- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

# Joined Relations Feature in SQL2 (cont.)

- Examples:

**Q8: SELECT            E.FNAME, E.LNAME, S.FNAME, S.LNAME  
      FROM            EMPLOYEE E S  
      WHERE           E.SUPERSSN=S.SSN**

can be written as:

**Q8: SELECT            E.FNAME, E.LNAME, S.FNAME, S.LNAME  
      FROM (EMPLOYEE E LEFT OUTER JOIN EMPLOYEES  
            ON E.SUPERSSN=S.SSN)**

**Q1: SELECT            FNAME, LNAME, ADDRESS  
      FROM EMPLOYEE, DEPARTMENT  
      WHERE           DNAME='Research' AND DNUMBER=DNO**

# Joined Relations Feature in SQL2 (cont.)

- could be written as:

**Q1: SELECT            FNAME, LNAME, ADDRESS  
      FROM (EMPLOYEE JOIN DEPARTMENT  
              ON DNUMBER=DNO)  
      WHERE            DNAME='Research'**

or as:

**Q1: SELECT            FNAME, LNAME, ADDRESS  
      FROM (EMPLOYEE NATURAL JOIN DEPARTMENT  
              AS DEPT(DNAME, DNO, MSSN, MSDATE)  
      WHERE            DNAME='Research'**

# Joined Relations Feature in SQL2 (cont.)

- Another Example;
  - Q2 could be written as follows; this illustrates multiple joins in the joined tables

```
Q2:      SELECT      PNUMBER, DNUM,  
                      LNAME,  
                      BDATE, ADDRESS  
          FROM      (PROJECT JOIN  
                      DEPARTMENT ON  
                      DNUM=DNUMBER) JOIN  
                      EMPLOYEE ON  
                      MGRSSN=SSN) )  
          WHERE      PLOCATION='Stafford'
```

# AGGREGATE FUNCTIONS

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

**Q15:**        **SELECT        MAX(SALARY),**  
                 **MIN(SALARY), AVG(SALARY)**  
                 **FROM        EMPLOYEE**

- Some SQL implementations *may not allow more than one function* in the SELECT-clause

# AGGREGATE FUNCTIONS

## (cont.)

- Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16: SELECT    MAX(SALARY), MIN(SALARY),  
                AVG(SALARY)  
FROM EMPLOYEE, DEPARTMENT  
WHERE DNO=DNUMBER AND  
      DNAME='Research'
```

# AGGREGATE FUNCTIONS

## (cont.)

- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

**Q17:**            **SELECT            COUNT (\*)**  
                  **FROM             EMPLOYEE**

**Q18:**            **SELECT            COUNT (\*)**  
                  **FROM             EMPLOYEE,**  
                                 **DEPARTMENT**  
                  **WHERE           DNO=DNUMBER AND**  
                                 **DNAME='Research'**

# GROUPING

- In many cases, we want to apply the aggregate functions *to subgroups of tuples in a relation*
- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*



# GROUPING (cont.)

- Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

**Q20: SELECT            DNO, COUNT (\*), AVG (SALARY)  
         FROM EMPLOYEE  
         GROUP BY     DNO**

- In Q20, the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO
- The COUNT and AVG functions are applied to each such group of tuples separately
- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
- A join condition can be used in conjunction with grouping

# GROUPING (cont.)

- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

**Q21:**        **SELECT        PNUMBER, PNAME, COUNT (\*)**  
              **FROM         PROJECT, WORKS\_ON**  
              **WHERE        PNUMBER=PNO**  
              **GROUP BY   PNUMBER, PNAME**

- In this case, the grouping and functions are applied *after* the joining of the two relations

# THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*
- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

# THE HAVING-CLAUSE (cont.)

- Query 22: For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

**Q22:**

|                 |                              |
|-----------------|------------------------------|
| <b>SELECT</b>   | <b>PNUMBER, PNAME, COUNT</b> |
| <b>(*)</b>      |                              |
| <b>FROM</b>     | <b>PROJECT, WORKS_ON</b>     |
| <b>WHERE</b>    | <b>PNUMBER=PNO</b>           |
| <b>GROUP BY</b> | <b>PNUMBER, PNAME</b>        |
| <b>HAVING</b>   | <b>COUNT (*) &gt; 2</b>      |

# SUBSTRING COMPARISON

- The **LIKE** comparison operator is used to compare partial strings
- Two reserved characters are used: '%' (or '\*' in some implementations) replaces an arbitrary number of characters, and '\_' replaces a single arbitrary character

# SUBSTRING COMPARISON (cont.)

- Query 25: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX'.

**Q25:**            **SELECT            FNAME, LNAME**  
                  **FROM                EMPLOYEE**  
                  **WHERE            ADDRESS LIKE**  
                         **'%Houston,TX%'**

# SUBSTRING COMPARISON (cont.)

- Query 26: Retrieve all employees who were born during the 1950s. Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '\_\_\_\_\_5\_', with each underscore as a place holder for a single arbitrary character.

**Q26:**            **SELECT            FNAME, LNAME**  
                 **FROM             EMPLOYEE**  
                 **WHERE          BDATE LIKE        '\_\_\_\_\_5\_'**

- The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible; hence, in SQL, character string attribute values are not atomic

# ARITHMETIC OPERATIONS

- The standard arithmetic operators '+', '-', '\*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result
- Query 27: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q27: SELECT      FNAME, LNAME, 1.1*SALARY
                FROM EMPLOYEE, WORKS_ON, PROJECT
                WHERE SSN=ESSN AND PNO=PNUMBER AND
                      PNAME='ProductX'
```



# ORDER BY

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

|      |                 |                                   |
|------|-----------------|-----------------------------------|
| Q28: | <b>SELECT</b>   | <b>DNAME, LNAME, FNAME, PNAME</b> |
|      | <b>FROM</b>     | <b>DEPARTMENT, EMPLOYEE,</b>      |
|      |                 | <b>WORKS_ON, PROJECT</b>          |
|      | <b>WHERE</b>    | <b>DNUMBER=DNO AND SSN=ESSN</b>   |
|      | <b>AND</b>      | <b>PNO=PNUMBER</b>                |
|      | <b>ORDER BY</b> | <b>DNAME, LNAME</b>               |

# ORDER BY (cont.)

- The default order is in ascending order of values
- We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default

# Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

**SELECT** <attribute list>  
**FROM** <table list>  
**[WHERE** <condition>  
**[GROUP BY** <grouping attribute(s)>  
**[HAVING** <group condition>  
**[ORDER BY** <attribute list>]

# Summary of SQL Queries (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

# Specifying Updates in SQL

- There are three SQL commands to modify the database; INSERT, DELETE, and UPDATE

# INSERT

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

# INSERT (cont.)

- Example:

**U1: INSERT INTO EMPLOYEE  
VALUES ('Richard','K','Marini', '653298653', '30-DEC-52',  
'98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4 )**

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
- Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

**U1A: INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)  
VALUES ('Richard', 'Marini', '653298653')**

# INSERT (cont.)

- Important Note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database
- Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation



# INSERT (cont.)

- Example: Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department. A table DEPTS\_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

**U3A:        CREATE TABLE DEPTS\_INFO  
                                 (DEPT\_NAME    VARCHAR(10),  
                                 NO\_OF\_EMPS    INTEGER,  
                                 TOTAL\_SAL     INTEGER);**

**U3B:        INSERT INTO    DEPTS\_INFO (DEPT\_NAME,  
                                 NO\_OF\_EMPS, TOTAL\_SAL)  
              SELECT        DNAME, COUNT (\*), SUM (SALARY)  
              FROM         DEPARTMENT, EMPLOYEE  
              WHERE        DNUMBER=DNO  
              GROUP BY     DNAME ;**

# INSERT (cont.)

- Note: The DEPTS\_INFO table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations *after* issuing U3B. We have to create a view (see later) to keep such a table up to date.

# DELETE

- Removes tuples from a relation
- Includes a WHERE-clause to select the tuples to be deleted
- Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
- A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
- The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause
- Referential integrity should be enforced

# DELETE (cont.)

- Examples:

|             |   |   |
|-------------|---|---|
| <b>U4A:</b> | <b>DELETE FROM<br/>WHERE</b>                                | <b>EMPLOYEE<br/>LNAME='Brown'</b>   |
| <b>U4B:</b> | <b>DELETE FROM<br/>WHERE</b>                                | <b>EMPLOYEE<br/>SSN='123456789'</b>   |
| <b>U4C:</b> | <b>DELETE FROM<br/>WHERE<br/>(SELECT<br/>FROM<br/>WHERE</b> | <b>EMPLOYEE<br/>DNO IN<br/>DNUMBER<br/>DEPARTMENT<br/>DNAME='Research')</b> |
| <b>U4D:</b> | <b>DELETE FROM</b>  | <b>EMPLOYEE</b>   |

# UPDATE

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

# UPDATE (cont.)

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U5: UPDATE    PROJECT  
    SET       PLOCATION = 'Bellaire', DNUM = 5  
    WHERE     PNUMBER=10
```

# UPDATE (cont.)

- Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6: UPDATE      EMPLOYEE  
SET            SALARY = SALARY *1.1  
WHERE          DNO IN (SELECT DNUMBER  
                  FROM      DEPARTMENT  
                  WHERE     DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
- The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
- The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

# Additional Features in SQL

- Section 8.7 (Ed. 5) : Specification of more general constraints via assertions
- Section 8.8 (Ed. 5) : SQL facilities for defining views (virtual tables) and implementation of views
- Chapter 9 discusses various techniques for accessing and manipulating a database via programs in general-purpose languages (e.g., Java)



# Constraints as Assertions

## (Section 8.7, Ed. 5)

- General constraints: constraints that do not fit in the basic SQL categories described in Section 8.2— such as referential integrity defined in the CREATE TABLE statement.
- Mechanism: CREATE ASSERTION
  - components include: a constraint name, followed by CHECK, followed by a condition that includes the result of an SQL query

# Assertions: An Example

- “The salary of an employee must not be greater than the salary of the manager of the department that the employee works for”

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT *
FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT
D
WHERE E.SALARY > M.SALARY AND
E.DNO=D.NUMBER AND
D.MGRSSN=M.SSN ) )
```

# Using General Assertions

- The assertion is stated by specifying a query that violates the condition; it is included inside a NOT EXISTS clause
- Query result must be empty
  - if the query result is not empty, the assertion has been violated
- The constraint is *satisfied* by a database state if no combination of tuples in that database state violates the constraint.
- The CHECK clause by itself is used to specify constraints on domains, attributes and tuples and is checked each time a tuple is inserted or updated.

# CHECK examples

- Attribute constraint:

**Dnumber INT NOT NULL CHECK**

**(Dnumber > 0 AND Dnumber < 21)**

- Use of CREATE DOMAIN:

**CREATE DOMAIN D\_NUM AS INTEGER**  
**CHECK (D\_NUM > 0 AND D\_NUM < 21)**

See page 249.

```

CREATE TABLE EMPLOYEE
( ...,
  Dno          INT          NOT NULL          DEFAULT 1,
  CONSTRAINT EMPPK
    PRIMARY KEY(Ssn),
  CONSTRAINT EMPSUPERFK
    FOREIGN KEY(Super_ssn) REFERENCES EMPLOYEE(Ssn)
      ON DELETE SET NULL      ON UPDATE CASCADE,
  CONSTRAINT EMPDEPTFK
    FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
      ON DELETE SET DEFAULT  ON UPDATE CASCADE );

CREATE TABLE DEPARTMENT
( ...,
  Mgr_ssn      CHAR(9)     NOT NULL          DEFAULT '888665555',
  ...,
  CONSTRAINT DEPTPK
    PRIMARY KEY(Dnumber),
  CONSTRAINT DEPTSK
    UNIQUE(Dname),
  CONSTRAINT DEPTMGRFK
    FOREIGN KEY(Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
      ON DELETE SET DEFAULT  ON UPDATE CASCADE );

CREATE TABLE DEPT_LOCATIONS
( ...,
  PRIMARY KEY(Dnumber, Dlocation),
  FOREIGN KEY(Dnumber) REFERENCES DEPARTMENT(Dnumber)
    ON DELETE CASCADE      ON UPDATE CASCADE );

```

**Figure 8.2**

Example illustrating how default attribute values and referential integrity triggered actions are specified in SQL.

# SQL Triggers

- Objective: to monitor a database and take action when a condition occurs
- Triggers are expressed in a syntax similar to assertions and include the following:
  - event (e.g., an update operation)
  - condition
  - action (to be taken when the condition is satisfied)

# SQL Triggers: An Example

- A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
    SALARY, SUPERVISOR_SSN ON EMPLOYEE
FOR EACH ROW
    WHEN
        (NEW.SALARY > (SELECT SALARY FROM EMPLOYEE
                        WHERE SSN=NEW.SUPERVISOR_SSN) )
    INFORM_SUPERVISOR (NEW.SUPERVISOR_SSN,NEW.SSN) ;
```

# Views in SQL (Section 8.8, Ed. 5)

- A view is a “virtual” table that is derived from other tables
- Allows for limited update operations (since the table may not physically be stored)
- Allows full query operations
- A convenience for expressing certain operations



# Specification of Views

- SQL command: CREATE VIEW
  - a table (view) name
  - a possible list of attribute names (for example, when arithmetic operations are specified or when we want the names to be different from the attributes in the base relations)
  - a query to specify the table contents

# SQL Views: An Example

- Specify a different WORKS\_ON table

```
CREATE VIEW WORKS_ON1 AS  
SELECT FNAME, LNAME, PNAME, HOURS  
FROM EMPLOYEE, PROJECT, WORKS_ON  
WHERE SSN=ESSN AND PNO=PNUMBER;
```

# Using a Virtual Table

- We can specify SQL queries on a newly create table (view):

```
QV1: SELECT FNAME, LNAME  
      FROM WORKS_ON_NEW  
      WHERE PNAME='ProjectX';
```

- When no longer needed, a view can be dropped:

```
DROP VIEW WORKS_ON1;
```

# View Using aggregation

- Views may involve SQL queries with aggregation:

```
V2: CREATE VIEW DEPT_INFO  
(Dept_name, No_of_emps, Total_sal)  
AS  
  
  SELECT Dname, COUNT (*), SUM  
(Salary)  
  
  FROM DEPARTMENT, EMPLOYEE,  
  
  WHERE Dnumber = Dno  
  
  GROUP BY Dname;
```

# Efficient View Implementation

- Query modification: present the view query in terms of a query on the underlying base tables
  - disadvantage: inefficient for views defined via complex queries (especially if additional queries are to be applied to the view within a short time period)

# Efficient View Implementation

- View materialization: involves physically creating and keeping a temporary table
  - assumption: other queries on the view will follow
  - concerns: maintaining correspondence between the base table and the view when the base table is updated
  - strategy: incremental update

# View Update

- Update on a single view without aggregate operations: update may map to an update on the underlying base table
- Views involving joins: an update *may* map to an update on the underlying base relations
  - not always possible

# Un-updatable Views

- Views defined using groups and aggregate functions are not updateable
- EXAMPLE:

```
UV2: UPDATE DEPT_INFO  
      SET Total_sal = 10000  
      WHERE Dname = 'Research' ;
```

The above view cannot be mapped to the base tables: EMPLOYEE AND DEPARTMENT



# Un-updatable Views (contd.)

- Views defined on multiple tables using joins are generally not updateable
- WITH CHECK OPTION: must be added to the definition of a view if the view is to be updated
  - to allow check for updatability and to plan for an execution strategy