

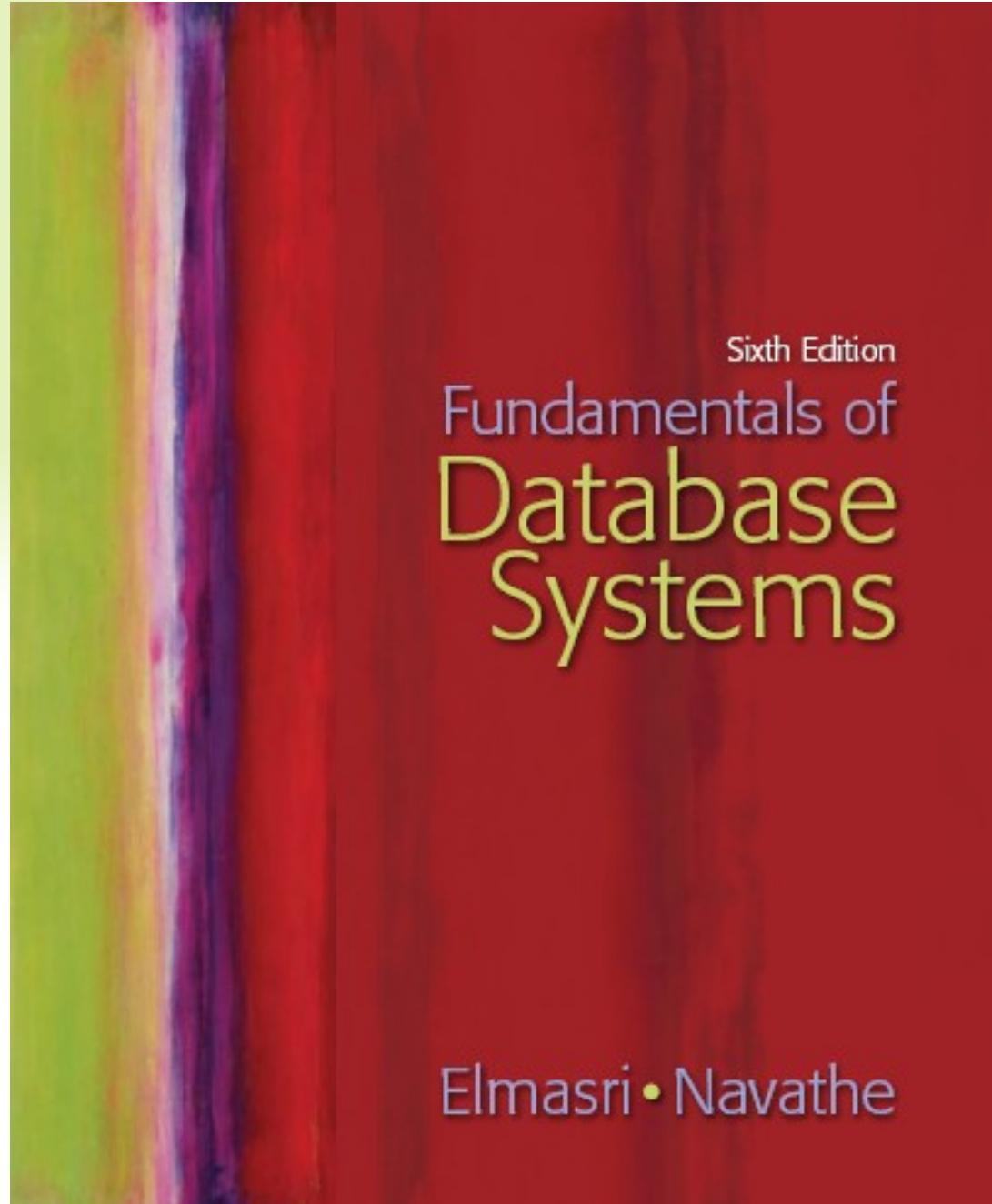
# **Chapter 5**

## **More SQL: Complex Queries, Triggers, Views, and Schema Modification**

Addison-Wesley  
is an imprint of



Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



# Outline

- More Complex SQL Retrieval Queries
- Specifying Constraints as Assertions and Action Triggers
- Views (Virtual Tables) in SQL
- Schema Change Statements in SQL

# More Complex SQL Retrieval Queries

- Additional features allow users to specify more complex retrievals from database:
  - Nested queries, joined tables, outer joins, aggregate functions, and grouping

# Comparisons Involving NULL and Three-Valued Logic

- Meanings of NULL
  - **Unknown value**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN

# Comparisons Involving NULL and Three-Valued Logic (cont'd.)

**Table 5.1** Logical Connectives in Three-Valued Logic

(a)	AND	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	FALSE	UNKNOWN
	FALSE	FALSE	FALSE	FALSE
	UNKNOWN	UNKNOWN	FALSE	UNKNOWN
(b)	OR	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE	UNKNOWN
	UNKNOWN	TRUE	UNKNOWN	UNKNOWN
(c)	NOT			
	TRUE	FALSE		
	FALSE	TRUE		
	UNKNOWN	UNKNOWN		

# Comparisons Involving NULL and Three-Valued Logic (cont'd.)

- SQL allows queries that check whether an attribute value is NULL
  - IS or IS NOT NULL

**Query 18.** Retrieve the names of all employees who do not have supervisors.

```
Q18:  SELECT      Fname, Lname  
        FROM       EMPLOYEE  
        WHERE      Super_ssn IS NULL;
```

# Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries**

- Complete select-from-where blocks within WHERE clause of another query
  - **Outer query**

- Comparison operator IN

- Compares value  $v$  with a set (or multiset) of values  $V$
  - Evaluates to TRUE if  $v$  is one of the elements in  $V$

# Nested Queries (cont'd.)

```
Q4A:  SELECT      DISTINCT Pnumber
      FROM       PROJECT
      WHERE      Pnumber IN
                  ( SELECT      Pnumber
                      FROM       PROJECT, DEPARTMENT, EMPLOYEE
                      WHERE      Dnum=Dnumber AND
                                 Mgr_ssn=Ssn AND Lname='Smith' )
                  OR
                  Pnumber IN
                  ( SELECT      Pno
                      FROM       WORKS_ON, EMPLOYEE
                      WHERE      Essn=Ssn AND Lname='Smith' );
```

# Nested Queries (cont'd.)

- Use tuples of values in comparisons
  - Place them within parentheses

```
SELECT      DISTINCT Essn
FROM        WORKS_ON
WHERE       (Pno, Hours) IN ( SELECT      Pno, Hours
                           FROM        WORKS_ON
                           WHERE       Essn='123456789' );
```

# Nested Queries (cont'd.)

- Use other comparison operators to compare a single value  $v$ 
  - $= ANY$  (or  $= SOME$ ) operator
    - Returns TRUE if the value  $v$  is equal to some value in the set  $V$  and is hence equivalent to IN
  - Other operators that can be combined with ANY (or SOME):  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ , and  $\neq$

```
SELECT      Lname, Fname
FROM        EMPLOYEE
WHERE       Salary > ALL    ( SELECT      Salary
                           FROM        EMPLOYEE
                           WHERE       Dno=5 );
```

# Nested Queries (cont'd.)

- Avoid potential errors and ambiguities
  - Create tuple variables (aliases) for all tables referenced in SQL query

**Query 16.** Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:   SELECT      E.Fname, E.Lname  
          FROM       EMPLOYEE AS E  
          WHERE      E.Ssn IN  ( SELECT      Essn  
                                FROM       DEPENDENT AS D  
                                WHERE      E.Fname=D.Dependent_name  
                                          AND E.Sex=D.Sex );
```

# Correlated Nested Queries

- **Correlated nested query**
  - Evaluated once for each tuple in the outer query

# The EXISTS and UNIQUE Functions in SQL

- EXISTS function
  - Check whether the result of a correlated nested query is empty or not
- EXISTS and NOT EXISTS
  - Typically used in conjunction with a correlated nested query
- SQL function UNIQUE(Q)
  - Returns TRUE if there are no duplicate tuple's in the result of query Q

## ■ **EXISTS**

- Used to check if sub-query returns any rows
- Ex. List the names of managers who have at least one dependent.

Select Fname, Lname

From Employee

Where **EXISTS** (Select \*

From Dependent

Where Ssn = Essn)

**AND**

**EXISTS** (Select \*

From Department

Where Ssn = Mgr\_ssn);  
15

# Explicit Sets and Renaming of Attributes in SQL

- Can use explicit set of values in WHERE clause
- Use qualifier AS followed by desired new name
  - Rename any attribute that appears in the result of a query

Q8A:    **SELECT**      E.Lname **AS** Employee\_name, S.Lname **AS** Supervisor\_name  
          **FROM**        EMPLOYEE **AS** E, EMPLOYEE **AS** S  
          **WHERE**        E.Super\_ssn=S.Ssn;

# Joined Tables in SQL and Outer Joins

- **Joined table**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query
- The FROM clause in Q1A
  - Contains a single joined table

**Q1A:**    **SELECT**      Fname, Lname, Address  
             **FROM**        (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)  
             **WHERE**      Dname='Research';

# Joined Tables in SQL and Outer Joins (cont'd.)

- Specify different types of join
  - NATURAL JOIN
  - Various types of OUTER JOIN
- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Implicit EQUIJOIN condition for each pair of attributes with same name from R and S

# Joined Tables in SQL and Outer Joins (cont'd.)

- **Inner join**
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation
- **LEFT OUTER JOIN**
  - Every tuple in left table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of right table

# Joined Tables in SQL and Outer Joins (cont'd.)

- **RIGHT OUTER JOIN**
  - Every tuple in right table must appear in result
  - If no matching tuple
    - Padded with NULL values for the attributes of left table
- **FULL OUTER JOIN**
- Can nest join specifications

# Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- **Grouping**
  - Create subgroups of tuples before summarizing
- Built-in aggregate functions
  - **COUNT, SUM, MAX, MIN, and AVG**
- Functions can be used in the SELECT clause or in a HAVING clause

# Aggregate Functions in SQL (cont'd.)

- NULL values discarded when aggregate functions are applied to a particular column

**Query 20.** Find the sum of the salaries of all employees of the ‘Research’ department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
| SELECT      SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
| FROM        (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
| WHERE       Dname='Research';
```

**Queries 21 and 22.** Retrieve the total number of employees in the company (Q21) and the number of employees in the ‘Research’ department (Q22).

Q21: **SELECT** COUNT (\*)  
**FROM** EMPLOYEE;

Q22: **SELECT** COUNT (\*)  
**FROM** EMPLOYEE, DEPARTMENT  
**WHERE** DNO=DNUMBER AND DNAME='Research';

# Grouping: The GROUP BY and HAVING Clauses

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- **GROUP BY** clause
  - Specifies grouping attributes
- If NULLs exist in grouping attribute
  - Separate group created for all tuples with a NULL value in grouping attribute

# Grouping: The GROUP BY and HAVING Clauses (cont'd.)

## ■ **HAVING clause**

- Provides a condition on the summary information

**Query 28.** For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than \$40,000.

**Q28:**

```
SELECT      Dnumber, COUNT (*)
FROM        DEPARTMENT, EMPLOYEE
WHERE       Dnumber=Dno AND Salary>40000 AND
           ( SELECT      Dno
             FROM        EMPLOYEE
             GROUP BY Dno
             HAVING      COUNT (*) > 5)
```

# Discussion and Summary of SQL Queries

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

# Specifying Constraints as Assertions and Actions Triggers

## ■ **CREATE ASSERTION**

- Specify additional types of constraints outside scope of built-in relational model constraints

## ■ **CREATE TRIGGER**

- Specify automatic actions that database system will perform when certain events and conditions occur

# Specifying General Constraints as Assertions in SQL

- CREATE ASSERTION
  - Specify a query that selects any tuples that violate the desired condition
  - Use only in cases where it is not possible to use CHECK on attributes and domains

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS ( SELECT      *
                      FROM        EMPLOYEE E, EMPLOYEE M,
                                  DEPARTMENT D
                     WHERE      E.Salary>M.Salary
                               AND E.Dno=D.Dnumber
                               AND D.Mgr_ssn=M.Ssn ) );
```

# Introduction to Triggers in SQL

- **CREATE TRIGGER** statement
  - Used to monitor the database
- Typical trigger has three components:
  - **Event(s)**
  - **Condition**
  - **Action**

# Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables
  - Considered to be a virtual table

# Specification of Views in SQL

## ■ CREATE VIEW command

- Give table name, list of attribute names, and a query to specify the contents of the view

```
V1:   CREATE VIEW WORKS_ON1
      AS SELECT Fname, Lname, Pname, Hours
            FROM EMPLOYEE, PROJECT, WORKS_ON
           WHERE Ssn=Essn AND Pno=Pnumber;

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;
```

# Specification of Views in SQL (cont'd.)

- Specify SQL queries on a view
- View always up-to-date
  - Responsibility of the DBMS and not the user
- **DROP VIEW** command
  - Dispose of a view

# View Implementation, View Update, and Inline Views

- Complex problem of efficiently implementing a view for querying
- **Query modification approach**
  - Modify view query into a query on underlying base tables
  - Disadvantage: inefficient for views defined via complex queries that are time-consuming to execute

# View Implementation

- **View materialization approach**
  - Physically create a temporary view table when the view is first queried
  - Keep that table on the assumption that other queries on the view will follow
  - Requires efficient strategy for automatically updating the view table when the base tables are updated

# View Implementation (cont'd.)

- **Incremental update strategies**
  - DBMS determines what new tuples must be inserted, deleted, or modified in a materialized view table

# View Update and Inline Views

- Update on a view defined on a single table without any aggregate functions
  - Can be mapped to an update on underlying base table
- View involving joins
  - Often not possible for DBMS to determine which of the updates is intended

# View Update and Inline Views (cont'd.)

- Clause **WITH CHECK OPTION**
  - Must be added at the end of the view definition if a view is to be updated
- **In-line view**
  - Defined in the FROM clause of an SQL query

# Schema Change Statements in SQL

## ■ Schema evolution commands

- Can be done while the database is operational
- Does not require recompilation of the database schema

# The DROP Command

- **DROP command**
  - Used to drop named schema elements, such as tables, domains, or constraint
- **Drop behavior options:**
  - CASCADE and RESTRICT
- **Example:**
  - `DROP SCHEMA COMPANY CASCADE;`

# The ALTER Command

- **Alter table actions** include:
  - Adding or dropping a column (attribute)
  - Changing a column definition
  - Adding or dropping table constraints
- **Example:**
  - `ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);`
- **To drop a column**
  - Choose either CASCADE or RESTRICT

# The ALTER Command (cont'd.)

- Change constraints specified on a table
  - Add or drop a named constraint

```
ALTER TABLE COMPANY.EMPLOYEE  
DROP CONSTRAINT EMPSUPERFK CASCADE;
```

# Summary

- Complex SQL:
  - Nested queries, joined tables, outer joins, aggregate functions, grouping
- CREATE ASSERTION and CREATE TRIGGER
- Views
  - Virtual or derived tables