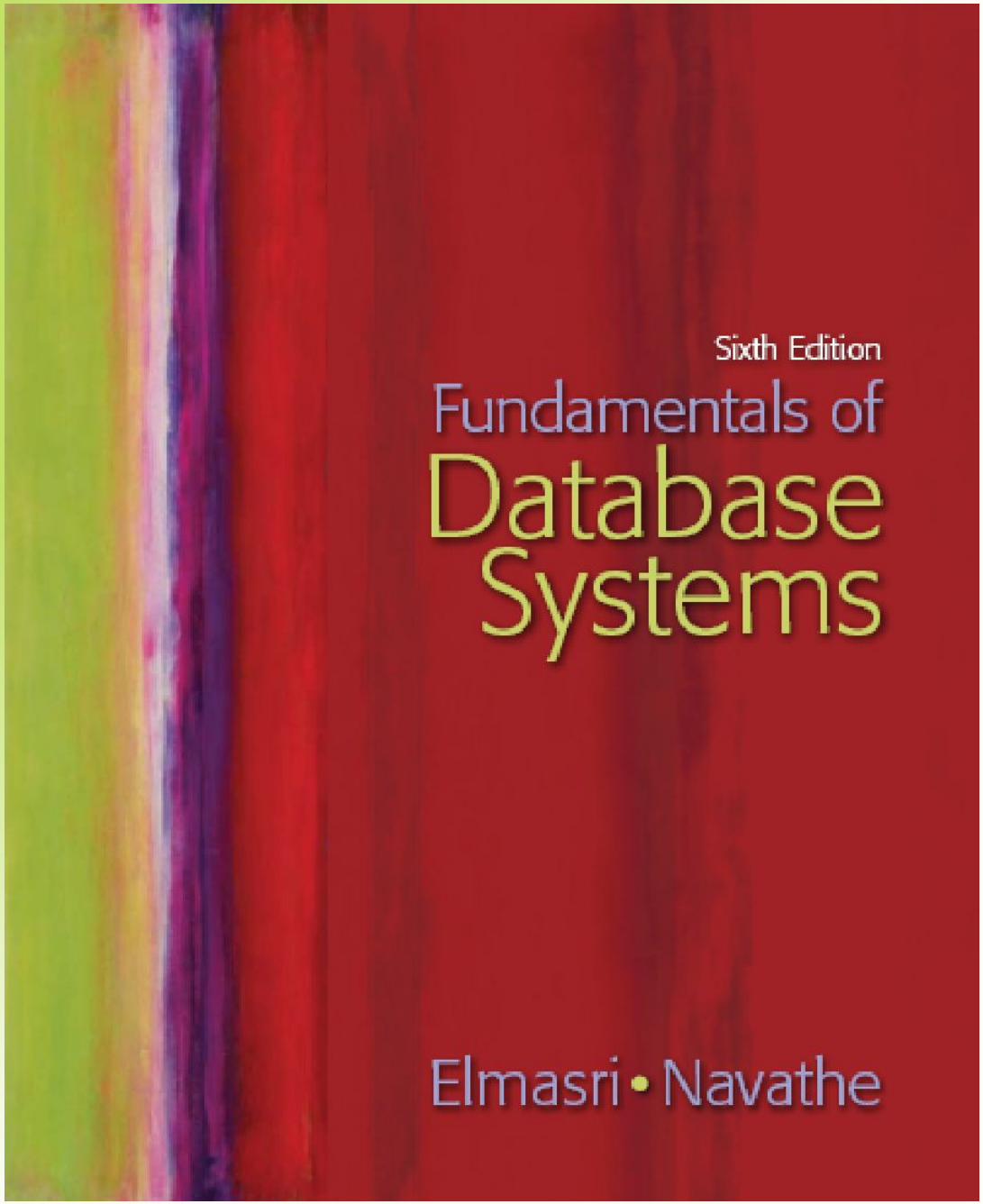
Chapter 5

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



Addison-Wesley
is an imprint of
PEARSON

Chapter 5

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

Addison-Wesley
is an imprint of
PEARSON

Chapter 5 Outline

- More Complex SQL Retrieval Queries
- Specifying Additional Constraints and Actions in SQL
  - CREATE ASSERTION
  - CREATE TRIGGER
- Views (virtual tables) in SQL
  - CREATE VIEW
- Schema Modification in SQL
  - ADD, DROP statements

Outline of Topics for More Complex SQL Retrieval Queries

- Handling NULLs, 3-valued Logic in SQL
- Nested Queries
  - Correlated vs. uncorrelated
  - EXISTS function
- Joined Tables, Inner Joins, and Outer Joins
- Aggregate Functions and Grouping in SQL
  - COUNT, AVG, SUM, MIN, MAX functions
  - GROUP BY, HAVING clauses

Handling NULLs in SQL

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

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

3-valued Logic in SQL

- Standard 2-valued logic assumes a condition can evaluate to either TRUE or FALSE
- With NULLs a condition can evaluate to UNKNOWN, leading to 3-valued logic
- Example: Consider a condition EMPLOYEE.DNO = 5; this evaluates for individual tuples in EMPLOYEE as follows:
  - TRUE for tuples with DNO=5
  - UNKNOWN for tuples where DNO is NULL
  - FALSE for other tuples in EMPLOYEE

3-valued Logic in SQL (cont.)

- Combining individual conditions using AND, OR, NOT logical connectives must consider UNKNOWN in addition to TRUE and FALSE
- Next slide (Table 5.1) shows the truth tables for 3-valued logic

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

Nesting of Queries in SQL

- A complete SELECT ... query, called a *nested query*, can be specified within the WHERE-clause of *another query*
  - The other query is 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.)

- In Q1, the *nested query* selects the DNUMBER of the 'Research' department
- The *outer query* select an EMPLOYEE tuple if its DNO value is in the result of the 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, 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   D.ESSN
               FROM      DEPENDENT AS D
               WHERE     E.FNAME=D.DEPENDENT_NAME) ;
```

Correlated Nested Queries (cont.)

- In Q12, the nested query has a different result *for each tuple* in the outer query (because it refers to E.FNAME)
- 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:   SELECT   E.FNAME, E.LNAME
        FROM     EMPLOYEE E, DEPENDENT D
        WHERE    E.SSN=D.ESSN AND
                 E.FNAME=D.DEPENDENT_NAME ;
```

Correlated Nested Queries (cont.)

- 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
  - 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 (see Chapter 6)

Correlated Nested Queries (cont.)

- Example of Using CONTAINS (not in current SQL)
- Query 3: Retrieve the name of each employee who works on *all the projects* controlled by department number 5.

```
Q3:  SELECT    E.FNAME, E.LNAME
     FROM      EMPLOYEE AS E
     WHERE  ( (SELECT   W.PNO
               FROM     WORKS_ON AS W
               WHERE    E.SSN=W.ESSN)
                  CONTAINS
               (SELECT  P.PNUMBER
               FROM     PROJECT AS P
               WHERE    P.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; this is *different for each employee tuple* because it references E.SSN

The EXISTS Function in SQL

- EXISTS is used to check whether the result of a query is empty (contains no tuples) or not (contains one or more tuples)
  - Applied to a query, but returns a boolean result (TRUE or FALSE)
  - Can be used in the WHERE-clause as a condition
  - EXISTS (Q) evaluates to TRUE if the result of Q has one or more tuple; evaluates to FALSE if the result of Q has no tuples

The EXISTS Function (cont.)

- Query 7: Retrieve the names of employees who are department managers *and* have at least one dependent.

```
Q7:   SELECT   M.FNAME, M.LNAME
      FROM     EMPLOYEE AS M
      WHERE    EXISTS  (SELECT    *
                        FROM      DEPENDENT
                        WHERE     M.SSN=ESSN)
               AND
               EXISTS (SELECT *
                        FROM      DEPARTMENT
                        WHERE     M.SSN=MGRSSN) ;
```

The EXISTS Function (cont.)

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

```
Q6:  SELECT   E.FNAME, E.LNAME
     FROM     EMPLOYEE E
     WHERE    NOT EXISTS   (SELECT  *
                            FROM     DEPENDENT D
                            WHERE    E.SSN=D.ESSN) ;
```

- 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 (Literal) Sets in SQL

- An **explicit (enumerated) set of values** is enclosed in parentheses
- 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) ;
```

Joined Tables (Relations) in SQL

- 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 (INNER JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc) – see the next slides
  - Each join type can specify a different query and produce a different result

Types of join – INNER JOIN

- This is the regular join operation
- Joined tuples must satisfy all join conditions
- Example: Query QJ1: Retrieve the employee names with the names of the department they work for

SELECT E.FNAME, E.LNAME, D.DNAME

FROM DEPARTMENT **AS** D, EMPLOYEE **AS** E

WHERE D.DNUMBER=E.DNO ;

This can be written using *joined tables* as follows:

SELECT E.FNAME, E.LNAME, D.DNAME

FROM (DEPARTMENT **AS** D **JOIN** EMPLOYEE **AS** E **ON** D.DNUMBER=E.DNO) ;

Types of join – OUTER JOIN

- In QJ1, an EMPLOYEE record is joined only if it has a matching DEPARTMENT with D.DNUMBER=E.DNO
- Hence, an EMPLOYEE with NULL for E.DNO will not appear in the query result
- Also, a DEPARTMENT that has no matching EMPLOYEE records (i.e. currently has no employees) does not appear in the query result
- OUTER JOINs gives the options to include every EMPLOYEE record or every DEPARTMENT record in the query results
- A record that does not have a matching joined record will be “padded” with an imaginary “NULL record” from the other table (all its attributes will be NULL)

Types of join – LEFT OUTER JOIN

- Example: Query QJ2: Retrieve the employee names with the names of the department they work for; *every department* must appear in the result even if it has no employees

This can be written using *joined tables* as follows:

SELECT E.FNAME, E.LNAME, D.DNAME

FROM (DEPARTMENT **AS** D **LEFT OUTER JOIN** EMPLOYEE **AS** E **ON** D.DNUMBER=E.DNO) ;

Note: An earlier left outer join syntax in ORACLE is as follows:

SELECT E.FNAME, E.LNAME, D.DNAME

FROM DEPARTMENT **AS** D, EMPLOYEE **AS** E

WHERE D.DNUMBER += E.DNO ;

Types of join – RIGHT OUTER JOIN

- Example: Query QJ3: Retrieve the employee names with the names of the department they work for; *every employee* must appear in the result even they are not currently assigned to a department

This can be written using *joined tables* as follows:

SELECT E.FNAME, E.LNAME, D.DNAME

FROM (DEPARTMENT **AS** D **RIGHT OUTER JOIN** EMPLOYEE **AS** E **ON** D.DNUMBER=E.DNO) ;

Note: An earlier left outer join syntax in ORACLE is as follows:

SELECT E.FNAME, E.LNAME, D.DNAME

FROM DEPARTMENT **AS** D, EMPLOYEE **AS** E

WHERE D.DNUMBER =+ E.DNO ;

Types of join – FULL OUTER JOIN

- Example: Query QJ4: Retrieve the employee names with the names of the department they work for; *every employee* and *every department* must appear in the result

This can be written using *joined tables* as follows:

SELECT E.FNAME, E.LNAME, D.DNAME

FROM (DEPARTMENT **AS** D **FULL OUTER JOIN** EMPLOYEE **AS** E **ON** D.DNUMBER=E.DNO) ;

Note: An earlier left outer join syntax in ORACLE is as follows:

SELECT E.FNAME, E.LNAME, D.DNAME

FROM DEPARTMENT **AS** D, EMPLOYEE **AS** E

WHERE D.DNUMBER +=+ E.DNO ;

Types of join – NATURAL JOIN

- If the join attributes in both tables *have the same name*, the join condition can be left out (it is automatically added by the system)
- NATURAL JOIN is a form of inner join
- Example: QJ5: We rename DNUMBER in DEPARTMENT to DNO to match the join attribute name (DNO) in EMPLOYEE (we also rename other attributes)
- Implicit join condition is E.DNO = D.DNO

SELECT E.FN, E.LN, E. ADR

FROM (DEPARTMENT **AS** D(DNM, DNO, MSSN, STRDATE)

NATURAL JOIN

EMPLOYEE **AS** E(FN,MI,LN,S,BD,ADR,SX,SAL,SU,DNO) ;

Joined Tables – Other Examples

- Query 8: Retrieve the employee names, and the names of their direct supervisor

```
Q8:SELECT   E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM (EMPLOYEE AS E INNER JOIN EMPLOYEE AS S
            ON E.SUPERSSN=S.SSN);
```

- In Q8, an *employee with no supervisor* will not appear in the result; if we want every employee to appear, we write:

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

Joined Tables – Other Examples

- Examples:

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

- 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 Tables – Other Examples

- 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**
- These can summarize information from multiple tuples into a single tuple
- Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q15:  SELECT  MAX(SALARY) AS HIGH_SAL,
              MIN(SALARY) AS LOW_SAL,
              AVG(SALARY) AS MEAN_SAL
      FROM    EMPLOYEE ;
```

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(E.SALARY), MIN(E.SALARY),
                    AVG(E.SALARY)
      FROM    EMPLOYEE E, DEPARTMENT D
      WHERE  E.DNO=D.DNUMBER AND
                    D.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). (Note: COUNT(*) counts the number of selected records)

```
Q17:  SELECT  COUNT (*)
      FROM    EMPLOYEE ;

Q18:  SELECT  COUNT (*)
      FROM    EMPLOYEE AS E, DEPARTMENT AS D
      WHERE   E.DNO=D.DNUMBER AND
              D.DNAME='Research' ;
```

Grouping (Partitioning Records into Subgroups)

- 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)* – for example, *employees who work in the same department* (have the same DNO)
- The aggregate functions are 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 has same value for the grouping attribute DNO
  - The COUNT and AVG functions are applied to each such group of tuples separately (see Figure 5.1(a), next slide)
  - The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples

Figure 5.1
Results of GROUP BY and HAVING. (a) Q24. (b) Q26.

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

(b)

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING

Continued next page...

Grouping (cont.)

- A join condition can be used with grouping
- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21:  SELECT    P.PNUMBER, P.PNAME, COUNT (*)
      FROM      PROJECT AS P, WORKS_ON AS W
      WHERE     P.PNUMBER=W.PNO
      GROUP BY  P.PNUMBER, P.PNAME ;
```

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

The HAVING-clause

- Sometimes we want to retrieve the values of these aggregate 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 (Figure 5.1(b) – next two slides).

```
Q22:  SELECT PNUMBER, PNAME, COUNT(*)
      FROM   PROJECT, WORKS_ON
      WHERE  PNUMBER=PNO
      GROUP BY PNUMBER, PNAME
      HAVING COUNT(*) > 2 ;
```

Figure 5.1
Results of GROUP BY and HAVING. (a) Q24. (b) Q26.

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

(b)

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING

Continued next page...

Pname	Pnumber	...	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	...	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

After applying the HAVING clause condition

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)

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, as well as joined tables
- 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 query result
  - Conceptually, a query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause and ORDER BY

Specifying General Constraints as Assertionsin SQL

- General constraints: constraints that do not fit in the basic SQL constraints (primary keys, UNIQUE, foreign keys, NOT NULL – see Chapter 4)
- Mechanism: **`CREAT ASSERTION`**
  - Components include:
    - a constraint name,
    - followed by `CHECK`,
    - followed by a condition that must be TRUE

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"

```
CREAT 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))
```

constraint name, CHECK, condition

Using General Assertions

1. Specify a query that *violates* the condition; include inside a `NOT EXISTS` clause
2. Query result must be empty; apply NOT EXISTS to it in the CHECK clause
3. If the query result is not empty, the assertion has been violated (CHECK will evaluate to FALSE)

SQL Triggers

- Used to monitor a database and initiate action when certain events and conditions occur (see Section 26.1 for details)
- Triggers are expressed in a syntax similar to assertions and include the following:

Event

- Such as an insert, deleted, or 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

- 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 defining complex operations once and reusing the definition
- Can also be used as a security mechanism

Specification of Views

SQL command: **`CREATE VIEW`**

- a virtual 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 view contents

SQL Views: An Example

- Specify a virtual DEPT_INFO table to summarize departmental information

- Makes it easier to query without having to specify the aggregate functions, GROUP BY, and HAVING

```
   CREATE VIEW DEPT_INFO(DNO, NO_EMPS,
TOTAL_SAL) AS
      SELECT DNO, COUNT(*), SUM(SALARY)
      FROM EMPLOYEE
      GROUP BY DNO;
```

Querying the View

- We can specify SQL retrieval queries on a view table, same as on a base table:

```
SELECT DNO
FROM DEPT_INFO
WHERE NO_OF_EMPS > 100;
```

- Can also specify joins and other retrieval operations on the view

SQL Views: Another Example

- Specify a virtual WORKS_ON table (called WORKS_ON_NEW), with EMPLOYEE and PROJECT names (instead of numbers)

- This makes it easier to query by names without having to specify the two join conditions

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

Querying a View (cont.)

We can specify SQL retrieval queries on a view table, same as on a base table:

```
SELECT FNAME, LNAME
FROM WORKS_ON_NEW
WHERE PNAME='Research';
```

When no longer needed, a view can be dropped:

```
DROP WORKS_ON_NEW;
```

View Implementation

- View implementation is hidden from the user
- Two main techniques

1. Query modification:

DBMS automatically modifies the view query into a query on the underlying base tables

Disadvantage:

Inefficient for views defined via complex queries

- Especially if many queries are to be applied to the view within a short time period

View Implementation (cont.)

2. View materialization:

- Involves physically creating and keeping a temporary table that holds the view query result
- Assumes that other queries on the view will follow

Concerns:

- Maintaining correspondence between the base tables and view when the base tables are updated

Strategy:

- Incremental update of the temporary view table

Updating of Views

- All views can be queried for retrievals, but many views cannot be updated

- Update on a view on a single table without aggregate operations:

If view includes key and NOT NULL attributes, view update may map to an update on the base table

- Views involving joins and aggregate functions are generally not updatable unless they can be mapped to unique updates on the base tables

Checking Views for Updatability

- When a user intends to update a view, must add the clause `WITH CHECK OPTION` at the end of the CREATE VIEW statement
- This allows the system to check for updatability
- If view is not updatable, and error will be generated
- If view is updatable, system will create a mapping strategy to process view updates

Schema modification in SQL

- There are two many commands for modifying schema constructs
- DROP statement can remove named schema constructs, such as tables, constraints, assertions, views, and even schemas
- ALTER statement can be used to change a table by adding or dropping of attributes and table constraints

Example: 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;
```

Example: DROP TABLE (cont.)

- If the table being dropped is referenced from other tables, it cannot be dropped and an error is generated
- By adding CASCADE, all references to the table are automatically removed
- Example:

```
DROP TABLE  DEPENDENT CASCADE;
```

Example: ALTER TABLE

- Can be used to add or drop an attribute from a base relation
  - Suppose we want to remove the attribute BDATE from the EMPLOYEE table
- Example:

```
ALTER TABLE EMPLOYEE DROP BDATE ;
```

- If the attribute is referenced from another table, an error is generated unless CASCADE is used

Example: ALTER TABLE (cont.)

- Suppose we want to add an attribute JOB
  - Will have NULLs (or some default) in all the tuples after command is executed; hence, NOT NULL not allowed for new JOB attribute
- Example:

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

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

Chapter 5 Summary

- More Complex SQL Retrieval Queries
- Specifying Additional Constraints and Actions in SQL
  - CREATE ASSERTION
  - CREATE TRIGGER
- Views (virtual tables) in SQL
  - CREATE VIEW
- Schema Modification in SQL
  - ADD, DROP statements