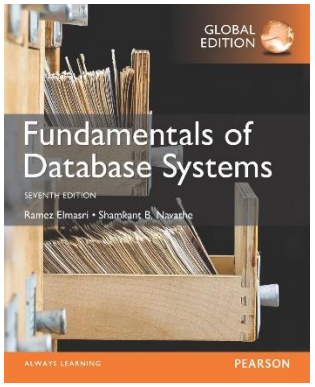
# Database Systems


Department of Computer Science
Chungbuk National University

Prof. Jong Yun LEE

# CHAPTER 7:

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



# Chapter 7 Outline

- **More Complex SQL Retrieval Queries**
- **Specifying Semantic Constraints as Assertions and Actions as Triggers**
- **Views (Virtual Tables) in SQL**
- **Schema Modification in SQL**

# More Complex SQL Retrieval Queries

# More Complex SQL Retrieval Queries

- **Additional features allow users to specify more complex retrievals from database:**
  - Nested queries, joined tables, and outer joins (in the FROM clause), 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 <u>three-valued logic</u>:**
  - `TRUE`, `FALSE`, and `UNKNOWN` (like Maybe)

- **<u>NULL = NULL comparison is avoided</u>**

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

**Table 7.1** 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;
```

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

```
mysql> desc current_dept_emp;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| emp_no    | int(11) | NO   |     | NULL    |       |
| dept_no   | char(4) | NO   |     | NULL    |       |
| from_date | date    | YES  |     | NULL    |       |
| to_date   | date    | YES  |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> SELECT emp_no from current_dept_emp WHERE from_date
IS NULL;
Empty set (0.33 sec)

mysql>
```

# Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries**
  - Complete select-from-where blocks within `WHERE` clause of another query
  - **Outer query and nested subqueries**

- **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.)

```
mysql> SELECT emp_no FROM dept_emp WHERE emp_no IN
( SELECT emp_no FROM current_dept_emp WHERE dept_no
= 'd001') limit 5;
+--------+
| emp_no |
+--------+
|  10017 |
|  10055 |
|  10058 |
|  10108 |
|  10108 |
+--------+
5 rows in set (0.29 sec)

mysql>
```

# 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' );
```

```
mysql> SELECT emp_no FROM dept_emp WHERE (emp_no, dept_no) IN
(SELECT emp_no, dept_no FROM dept_emp WHERE dept_no = 'd001')
limit 3;
+--------+
| emp_no |
+--------+
|  10017 |
|  10055 |
|  10058 |
+--------+
3 rows in set (0.00 sec)

mysql>
```

# 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`): `>, >=, <, <=,` and `<>`
  - `ALL`: value must exceed all values from nested query

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

# Nested Queries (cont'd.)

```
mysql> desc current_dept_emp;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| emp_no    | int(11) | NO   |     | NULL    |       |
| dept_no   | char(4) | NO   |     | NULL    |       |
| from_date | date    | YES  |     | NULL    |       |
| to_date   | date    | YES  |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> desc salaries;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| emp_no    | int(11) | NO   | PRI | NULL    |       |
| salary    | int(11) | NO   |     | NULL    |       |
| from_date | date    | NO   | PRI | NULL    |       |
| to_date   | date    | NO   |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> SELECT salary FROM salaries WHERE salary > ALL (SELECT S.salary from salaries S,
current_dept_emp C WHERE S.emp_no = C.emp_no AND C.dept_no = 'd001') limit 3;
+--------+
| salary |
+--------+
| 145732 |
| 145215 |
| 148820 |
+--------+
```

# 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  D.Essn
                          FROM    DEPENDENT AS D
                          WHERE   E.Fname = D.Dependent_name
                          AND E.Sex = D.Sex );
```

# Nested Queries (cont'd.)

```
mysql> SELECT salary FROM salaries WHERE salary > ALL (SELECT S.salary
from salaries AS S, current_dept_emp AS C WHERE S.emp_no = C.emp_no
AND C.dept_no = 'd001') limit 5;
+--------+
| salary |
+--------+
| 145732 |
| 145215 |
| 148820 |
| 145300 |
| 149440 |
+--------+
5 rows in set (0.84 sec)

mysql>
```

# Correlated Nested Queries

- **Queries that are nested using the = or IN comparison operator can be collapsed into one single block: E.g., Q16 can be written as:**

```
Q16:  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
                           AND E.Sex = D.Sex );
```

```
Q16A:  SELECT   E.Fname, E.Lname
       FROM     EMPLOYEE AS E, DEPENDENT AS D
       WHERE    E.Ssn=D.Essn AND E.Sex=D.Sex
                AND E.Fname=D.Dependent_name;
```

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

# The EXISTS and UNIQUE Functions in SQL for correlating queries

- **`EXISTS` function**
  - Check whether the result of a correlated nested query is empty or not. They are Boolean functions that return a `TRUE` or `FALSE` result.

- **`EXISTS` and `NOT EXISTS`**
  - Typically used in conjunction with a correlated nested query

- **SQL function `UNIQUE(Q)`**
  - Returns `TRUE` if there are no duplicate tuples in the result of query Q

# USE of EXISTS

**Q7: 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)
```

# USE of EXISTS

```
mysql> SELECT salary FROM salaries WHERE EXISTS (SELECT S.salary from
salaries AS S, current_dept_emp AS C WHERE S.emp_no = C.emp_no AND
C.dept_no = 'd001') limit 5;
+--------+
| salary |
+--------+
|  60117 |
|  62102 |
|  66074 |
|  66596 |
|  66961 |
+--------+
5 rows in set (0.30 sec)

mysql>
```

# USE OF NOT EXISTS

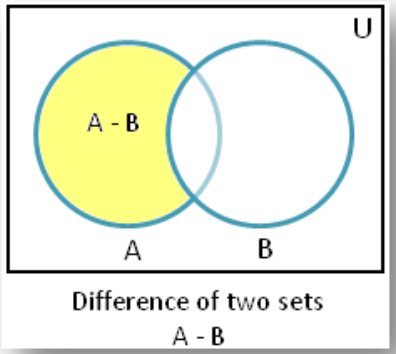
**To achieve the "for all" (universal quantifier- see Ch.8) effect, we use double negation this way in SQL:**

**Query: List first and last name of employees who work on ALL projects controlled by Dno=5.**

```
SELECT Fname, Lname
FROM Employee
WHERE NOT EXISTS ( (SELECT  Pnumber
                    FROM PROJECT
                    WHERE Dno=5)
```

**Empty Set**

```
EXCEPT (SELECT   Pno
        FROM WORKS_ON
        WHERE Ssn= ESsn));
```




The above is equivalent to double negation: List names of those employees for whom there does NOT exist a project managed by department no. 5 that they do NOT work on.

# USE OF NOT EXISTS

```
mysql> SELECT salary FROM salaries WHERE NOT EXISTS (SELECT S.salary
from salaries AS S, current_dept_emp AS C WHERE S.emp_no = C.emp_no
AND C.dept_no = 'd001')  limit 5;
Empty set (0.30 sec)

mysql>
```

```
mysql> SELECT salary FROM salaries WHERE salary NOT IN (SELECT
S.salary from salaries AS S, current_dept_emp AS C WHERE S.emp_no =
C.emp_no AND C.dept_no = 'd001')  limit 5;
+--------+
| salary |
+--------+
|  81025 |
|  85112 |
|  69366 |
|  40006 |
|  43636 |
+--------+
```

Note that MySQL does not support the EXCEPT, MINUS operator.

# Double Negation to accomplish “for all” in SQL

- **Q3B:**

```
SELECT   Lname, Fname
FROM     EMPLOYEE
WHERE    NOT EXISTS ( SELECT   *
                      FROM     WORKS_ON B
                      WHERE    ( B.Pno IN  (  SELECT Pnumber
                                              FROM PROJECT
                                              WHERE Dnum=5)
                                 AND
                                 NOT EXISTS ( SELECT  *
                                              FROM  WORKS_ON C
                                              WHERE  C.Essn=Ssn
                                                   AND      C.Pno=B.Pno )));
```

Select any `WORK_ON(B)` tuples whose `Pno` is a project controlled by department 5

if there is not a `WORK_ON(C)` tuple with the same `Pno` and the same `Ssn`.

**The above is a direct rendering of: List names of those employees for whom there does NOT exist a project managed by department no. 5 that they do NOT work on.**

# Explicit Sets and Renaming of Attributes in SQL

- **Can use explicit set of values in WHERE clause**

**Q17:**

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     Pno IN (1, 2, 3);
```

```
mysql> SELECT DISTINCT dept_no FROM current_dept_emp;
+---------+
| dept_no |
+---------+
| d005    |
| d007    |
| d004    |
| d003    |
| d008    |
| d006    |
| d009    |
| d001    |
| d002    |
+---------+
```

# Explicit Sets and Renaming of Attributes in SQL

- **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;
```

```
mysql> SELECT C.dept_no AS department_number FROM
current_dept_emp AS C, employees AS E where C.emp_no =
E.emp_no limit 5;
+-------------------+
| department_number |
+-------------------+
| d005              |
| d007              |
| d004              |
| d004              |
| d003              |
+-------------------+
5 rows in set (0.29 sec)
```

# Specifying Joined Tables in the FROM Clause of SQL

- **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. JOIN may also be called INNER JOIN

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

# Different Types of JOINed Tables in SQL

- **Specify different types of join**
  - NATURAL JOIN
  - Various types of OUTER JOIN (LEFT, RIGHT, FULL )

- **NATURAL JOIN on two relations R and S**
  - No join condition specified
  - Is equivalent to an implicit EQUIJOIN condition for each pair of attributes with same name from R and S

# NATURAL JOIN

- **Rename attributes of one relation so it can be joined with another using NATURAL JOIN:**

```
Q1B:    SELECT    Fname, Lname, Address
        FROM      (EMPLOYEE NATURAL JOIN
                  (DEPARTMENT AS DEPT (Dname, Dno, Mssn,
                        Msdate)))
        WHERE      Dname='Research';
```

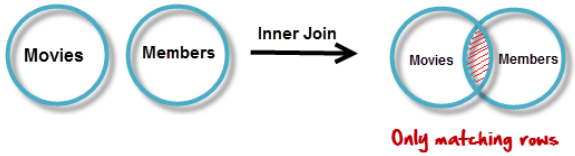
**The above works with EMPLOYEE.Dno = DEPT.Dno as an implicit join condition**

# NATURAL JOIN

```
mysql> SELECT C.dept_no AS department_number FROM current_dept_emp AS C
NATURAL JOIN employees AS E limit 5;
+-------------------+
| department_number |
+-------------------+
| d005              |
| d007              |
| d004              |
| d004              |
| d003              |
+-------------------+

mysql> SELECT C.dept_no AS department_number FROM current_dept_emp AS C
NATURAL JOIN employees AS E WHERE C.emp_no = E.emp_no limit 5;
+-------------------+
| department_number |
+-------------------+
| d005              |
| d007              |
| d004              |
| d004              |
| d003              |
+-------------------+
```

# INNER and OUTER Joins




- **INNER JOIN (versus OUTER 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


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

# Example: LEFT OUTER JOIN

```
SELECT E.Lname AS Employee_Name
       S.Lname AS Supervisor_Name
FROM Employee AS E LEFT OUTER JOIN EMPLOYEE AS S
                                ON E.Super_ssn = S.Ssn)
```

**ALTERNATE SYNTAX:**

```
SELECT E.Lname , S.Lname
FROM  EMPLOYEE E, EMPLOYEE S
WHERE E.Super_ssn += S.Ssn
```

# Example: LEFT OUTER JOIN

```
mysql> SELECT C.* FROM current_dept_emp AS C LEFT JOIN
employees AS E ON  C.dept_no = 'd001' limit 5;
+--------+---------+------------+------------+
| emp_no | dept_no | from_date  | to_date    |
+--------+---------+------------+------------+
|  10017 | d001    | 1993-08-03 | 9999-01-01 |
|  10055 | d001    | 1992-04-27 | 1995-07-22 |
|  10058 | d001    | 1988-04-25 | 9999-01-01 |
|  10108 | d001    | 1999-12-06 | 2001-10-20 |
|  10140 | d001    | 1991-03-14 | 9999-01-01 |
+--------+---------+------------+------------+
5 rows in set (0.33 sec)

mysql>
```

# Example: RIGHT OUTER JOIN

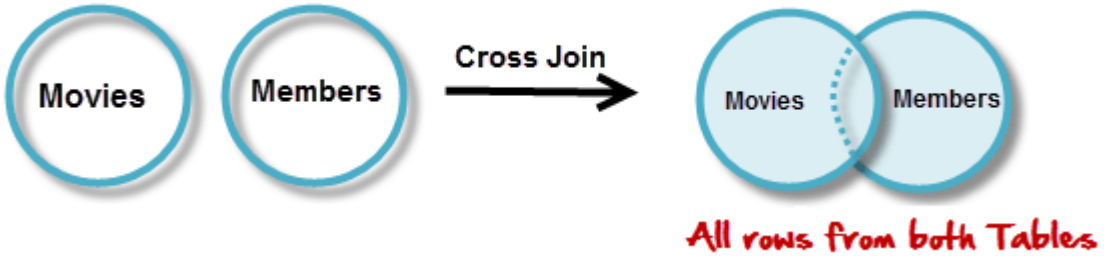
```
mysql> SELECT E.* FROM current_dept_emp AS C RIGHT JOIN employees AS E
ON  C.dept_no = 'd001' limit 5;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
+--------+------------+------------+-----------+--------+------------+
5 rows in set (0.29 sec)

mysql>
```

# Example: CROSS JOIN (CARTESIAN PRODUCT)

```
mysql> SELECT E.* FROM current_dept_emp AS C CROSS JOIN employees AS E
limit 5;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
|  10002 | 1964-06-02 | Bezalel    | Simmel    | F      | 1985-11-21 |
|  10003 | 1959-12-03 | Parto      | Bamford   | M      | 1986-08-28 |
|  10004 | 1954-05-01 | Chirstian  | Koblick   | M      | 1986-12-01 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak  | M      | 1989-09-12 |
+--------+------------+------------+-----------+--------+------------+
5 rows in set (0.30 sec)

mysql>
```



# Multiway JOIN in the FROM clause

- **FULL OUTER JOIN – combines result if LEFT and RIGHT OUTER JOIN**

- **Can nest JOIN specifications for a multiway join:**

```
Q2A:  SELECT   Pnumber, Dnum, Lname, Address, Bdate
      FROM     ((PROJECT JOIN DEPARTMENT ON
                 Dnum=Dnumber)  JOIN EMPLOYEE ON
                 Mgr_ssn=Ssn)
      WHERE    Plocation='Stafford';
```

# Aggregate Functions in SQL

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

# Renaming Results of Aggregation

- Following query returns a single row of computed values from `EMPLOYEE` table:

```
Q19:    SELECT   SUM (Salary), MAX (Salary), MIN (Salary),
                 AVG (Salary)
        FROM     EMPLOYEE;
```

- The result can be presented with new names:

```
Q19A:   SELECT   SUM (Salary) AS Total_Sal, MAX (Salary) AS
                 Highest_Sal, MIN (Salary) AS Lowest_Sal, AVG
                 (Salary) AS Average_Sal
        FROM     EMPLOYEE;
```

# Renaming Results of Aggregation

```
mysql> SELECT SUM(salary), AVG(salary), MIN(salary), MAX(salary)
from salaries;
+--------------+-------------+-------------+-------------+
| SUM(salary)  | AVG(salary) | MIN(salary) | MAX(salary) |
+--------------+-------------+-------------+-------------+
| 181480757419 |  63810.7448 |       38623 |      158220 |
+--------------+-------------+-------------+-------------+
1 row in set (0.72 sec)

mysql>
```

# Aggregate Functions in SQL (cont'd.)

- **NULL values are 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.

```
Q20:    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';
```

# Aggregate Functions in SQL (cont'd.)

```
mysql> SELECT COUNT(*) from employees;
+----------+
| COUNT(*) |
+----------+
|   300024 |
+----------+
1 row in set (0.03 sec)

mysql>
```

# Aggregate Functions on Booleans

- **SOME and ALL may be applied as functions on Boolean Values.**

- **SOME returns true if at least one element in the collection is TRUE (similar to OR)**

- **ALL returns true if all of the elements in the collection are TRUE (similar to AND)**

# Grouping: The `GROUP BY` Clause

- **Partition relation into subsets of tuples**
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently

- **`GROUP BY` clause**
  - Specifies grouping attributes

- **`COUNT (*)` counts the number of rows in the group**

# Examples of GROUP BY

- The grouping attribute must appear in the SELECT clause:

  Q24: SELECT Dno, COUNT (*), AVG (Salary)
  FROM EMPLOYEE
  GROUP BY Dno;

(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

```
mysql> SELECT dept_no, COUNT(*)
FROM current_dept_emp GROUP BY
dept_no;
+---------+----------+
| dept_no | COUNT(*) |
+---------+----------+
| d001    |    18426 |
| d002    |    15579 |
| d003    |    16071 |
| d004    |    66675 |
| d005    |    76958 |
| d006    |    18295 |
| d007    |    46922 |
| d008    |    19285 |
| d009    |    21813 |
+---------+----------+
```

# Examples of GROUP BY

- If the grouping attribute has NULL as a possible value, then a separate group is created for the null value (e.g., null Dno in the above query)

- GROUP BY may be applied to the result of a JOIN:

```
Q25:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber=Pno
        GROUP BY    Pnumber, Pname;
```

# Examples of GROUP BY

```
mysql> SELECT dept_no, gender, COUNT(*) FROM current_dept_emp C,
employees E WHERE C.emp_no = E.emp_no  GROUP BY dept_no, E.gender;
+---------+--------+----------+
| dept_no | gender | COUNT(*) |
+---------+--------+----------+
| d001    | M      |    11111 |
| d001    | F      |     7315 |
| d002    | M      |     9273 |
| d002    | F      |     6306 |
| d003    | M      |     9701 |
| d003    | F      |     6370 |
| d004    | M      |    39885 |
| d004    | F      |    26790 |
| d005    | M      |    46218 |
| d005    | F      |    30740 |
| d006    | M      |    10921 |
| d006    | F      |     7374 |
| d007    | M      |    28176 |
| d007    | F      |    18746 |
| d008    | M      |    11587 |
| d008    | F      |     7698 |
| d009    | M      |    13101 |
| d009    | F      |     8712 |
+---------+--------+----------+
```

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

- **HAVING clause**
  - Provides a condition to select or reject an entire group:

- **Query 26. For each project *on which more than two employees work,* retrieve the project number, the project name, and the number of employees who work on the project.**

```
Q26:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber=Pno
        GROUP BY    Pnumber, Pname
        HAVING      COUNT (*) > 2;
```

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

(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

| 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)

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

```
mysql> SELECT dept_no, gender, COUNT(*) FROM current_dept_emp C,
employees E WHERE C.emp_no = E.emp_no  GROUP BY dept_no, E.gender
HAVING COUNT(*) > 10000;
+---------+--------+----------+
| dept_no | gender | COUNT(*) |
+---------+--------+----------+
| d001    | M      |    11111 |
| d004    | M      |    39885 |
| d004    | F      |    26790 |
| d005    | M      |    46218 |
| d005    | F      |    30740 |
| d006    | M      |    10921 |
| d007    | M      |    28176 |
| d007    | F      |    18746 |
| d008    | M      |    11587 |
| d009    | M      |    13101 |
+---------+--------+----------+
10 rows in set (1.07 sec)

mysql>
```

# Combining the WHERE and the HAVING Clause

- **Consider the query: we want to count the *total* number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work.**

- **INCORRECT QUERY:**

```
SELECT      Dno, COUNT (*)
FROM        EMPLOYEE
WHERE       Salary > 40000
GROUP BY    Dno
HAVING      COUNT (*) > 5;
```

it will select only departments that have more than five employees who each earn more than $40000.

The rule is that the `WHERE` clause is executed first, to select individual tuples or joined tuples; the `HAVING` clause is applied later, to select individual groups of tuples.

# Combining the WHERE and the HAVING Clause (continued)

## Correct Specification of the Query:

- **Note: the `WHERE` clause applies tuple by tuple whereas `HAVING` applies to entire group of tuples**

```
Q28:    SELECT     Dno, COUNT (*)
        FROM       EMPLOYEE
        WHERE      Salary>40000 AND Dno IN
                   ( SELECT     Dno
                     FROM       EMPLOYEE
        GROUP BY   Dno
                     HAVING     COUNT (*) > 5)
        GROUP BY   Dno;
```

# Use of `WITH`

- **The `WITH` clause allows a user to define a table that will only be used in a particular query (not available in all SQL implementations)**

- **Used for convenience to create a temporary "View" and use that immediately in a query**

- **Allows a more straightforward way of looking a step-by-step query**

A common table expression (CTE) is a named temporary result set that exists within the scope of a single statement and that can be referred to later within that statement, possibly multiple times.

# Example of WITH

- **See an alternate approach to doing Q28:**

```
Q28':   WITH  BIGDEPTS (Dno) AS
              (SELECTDno
               FROM EMPLOYEE
               GROUP BY Dno
               HAVING       COUNT (*) > 5)

        SELECT   Dno, COUNT (*)
        FROM     EMPLOYEE
        WHERE    Salary>40000 AND Dno IN BIGDEPTS
        GROUP BY Dno;
```

# Use of CASE

- **SQL also has a CASE construct**
- **Used when a value can be different based on certain conditions.**
- **Can be used in any part of an SQL query where a value is expected**
- **Applicable when querying, inserting or updating tuples**

# EXAMPLE of use of CASE

- **The following example shows that employees are receiving different raises in different departments (A variation of the update U6)**

**U6':**

```
UPDATE    EMPLOYEE
SET       Salary =
CASE      WHEN Dno = 5    THEN  Salary + 2000
          WHEN Dno = 4    THEN  Salary + 1500
          WHEN Dno = 1    THEN  Salary + 3000
```

# EXPANDED Block Structure 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 as Triggers

# Specifying Constraints as Assertions and Actions as Triggers

- **<u>Semantic Constraints:</u> The following are beyond the scope of the EER and relational model**

- **`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 goes beyond a simple CHECK which applies to individual 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 which make it a rule for an "active database":**
  - **Event(s)**
  - **Condition**
  - **Action**

# USE OF TRIGGERS

- **AN EXAMPLE with standard Syntax.(Note : other SQL implementations like PostgreSQL use a different syntax.)**

**R5:**
**CREATE TRIGGER** SALARY_VIOLATION
**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 (Virtual Tables) in SQL

- **Concept of a view in SQL**
  - Single table derived from other tables called the **defining tables**
  - Considered to be a virtual table that is not necessarily populated

# 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
  - In V1, attributes retain the names from base tables. In V2, attributes are assigned names

```
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

```
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;
```

**WORKS_ON1**

| Fname | Lname | Pname | Hours |
|---|---|---|---|

**DEPT_INFO**

| Dept_name | No_of_emps | Total_sal |
|---|---|---|

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

- **Once a View is defined, SQL queries can use the View relation in the FROM clause**

- **View is 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**

- **Strategy1: Query modification approach**
  - Compute the view as and when needed. Do not store permanently
  - 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 Materialization

- **Strategy 2: View materialization**
  - 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

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

# View Materialization (contd.)

- **Multiple ways to handle materialization:**
  - **immediate update** strategy updates a view as soon as the base tables are changed
  - **lazy update** strategy updates the view when needed by a view query
  - **periodic update** strategy updates the view periodically (in the latter strategy, a view query may get a result that is not up-to-date). This is commonly used in Banks, Retail store operations, etc.

# View Update

- **Update on a view defined on a single table without any aggregate functions**
  - Can be mapped to an update on underlying base table- possible if the primary key is preserved in the view

- **Update not permitted on aggregate views. E.g.,**

**UV2:**

```
UPDATE      DEPT_INFO
SET         Total_sal=100000
WHERE       Dname='Research';
```

**cannot be processed because `Total_sal` is a computed value in the view definition**

# View Update

- **In MySQL, views are not only query-able but also updatable. It means that you can use the INSERT or UPDATE statement to insert or update rows of the base table through the updatable view. In addition, you can use DELETE statement to remove rows of the underlying table through the view.**

- **However, to create an updatable view, the SELECT statement that defines the view must not contain any of the following elements:**
  - Aggregate functions such as MIN, MAX, SUM, AVG, and COUNT.
  - DISTINCT
  - GROUP BY clause.
  - HAVING clause.
  - UNION or UNION ALL clause.
  - Left join or outer join.
  - Subquery in the SELECT clause or in the WHERE clause that refers to the table appeared in the FROM clause.
  - Reference to non-updatable view in the FROM clause.
  - Reference only to literal values.
  - Multiple references to any column of the base table.

# View Update and Inline Views

- **View involving joins**
  - Often not possible for DBMS to determine which of the updates is intended

- **Clause `WITH CHECK OPTION`**
  - Must be added at the end of the view definition if a view is to be updated to make sure that tuples being updated stay in the view

- **In-line view**
  - Defined in the `FROM` clause of an SQL query (e.g., we saw its used in the WITH example)

# Views as authorization mechanism

- **SQL query authorization statements (GRANT and REVOKE) are described in detail in Chapter 30**

- **Views can be used to hide certain attributes or tuples from unauthorized users**

- **E.g., For a user who is only allowed to see employee information for those who work for department 5, he may only access the view DEPT5EMP:**

```
CREATE VIEW   DEPT5EMP  AS
SELECT        *
FROM          EMPLOYEE
WHERE         Dno = 5;
```

# Views as authorization mechanism

```
mysql> CREATE VIEW TEST AS SELECT * FROM dept_emp WHERE
dept_no = 'd001';
Query OK, 0 rows affected (0.69 sec)

mysql>
mysql> SELECT * FROM TEST limit 5;
+--------+---------+------------+------------+
| emp_no | dept_no | from_date  | to_date    |
+--------+---------+------------+------------+
|  10017 | d001    | 1993-08-03 | 9999-01-01 |
|  10055 | d001    | 1992-04-27 | 1995-07-22 |
|  10058 | d001    | 1988-04-25 | 9999-01-01 |
|  10108 | d001    | 1999-12-06 | 2001-10-20 |
|  10140 | d001    | 1991-03-14 | 9999-01-01 |
+--------+---------+------------+------------+
5 rows in set (0.00 sec)

mysql> DROP VIEW TEST;
Query OK, 0 rows affected (0.00 sec)
```

# Schema Change Statements in SQL

# Schema Change Statements in SQL

- **Schema evolution commands**
  - DBA may want to change the schema 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;`
  - This removes the schema and all its elements including tables, views, constraints, etc.

# The ALTER table 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);`

# Adding and Dropping Constraints

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

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

# Dropping Columns, Default Values

- **To drop a column**
  - Choose either CASCADE or RESTRICT
  - CASCADE would drop the column from views etc. RESTRICT is possible if no views refer to it.

```
ALTER TABLE COMPANY.EMPLOYEE
            DROP COLUMN Address CASCADE;
```

- **Default values can be dropped and altered :**

```
ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn
          DROP DEFAULT;

ALTER TABLE COMPANY.DEPARTMENT
         ALTER COLUMN Mgr_ssn SET DEFAULT '333445555';
```

# Table 7.2 Summary of SQL Syntax

**Table 7.2** Summary of SQL Syntax

```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                            { , <column name> <column type> [ <attribute constraint> ] }
                            [ <table constraint> { , <table constraint> } ] )

DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>

SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]

<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                       { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )

<grouping attributes> ::= <column name> { , <column name> }

<order> ::= ( ASC | DESC )

INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )
```

*continued on next slide*

# Table 7.2 (continued) Summary of SQL Syntax

**Table 7.2** Summary of SQL Syntax

| |
|---|
| DELETE FROM <table name><br>[ WHERE <selection condition> ] |
| UPDATE <table name><br>SET <column name> = <value expression> { , <column name> = <value expression> }<br>[ WHERE <selection condition> ] |
| CREATE [ UNIQUE] INDEX <index name><br>ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )<br>[ CLUSTER ] |
| DROP INDEX <index name> |
| CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]<br>AS <select statement> |
| DROP VIEW <view name> |

NOTE: The commands for creating and dropping indexes are not part of standard SQL.

# Summary

- **Complex SQL:**
  - Nested queries, joined tables (in the FROM clause), outer joins, aggregate functions, grouping

- **Handling semantic constraints with `CREATE ASSERTION` and `CREATE TRIGGER`**

- **`CREATE VIEW` statement and materialization strategies**

- **Schema Modification for the DBAs using `ALTER TABLE, ADD and DROP COLUMN, ALTER CONSTRAINT` etc.**