

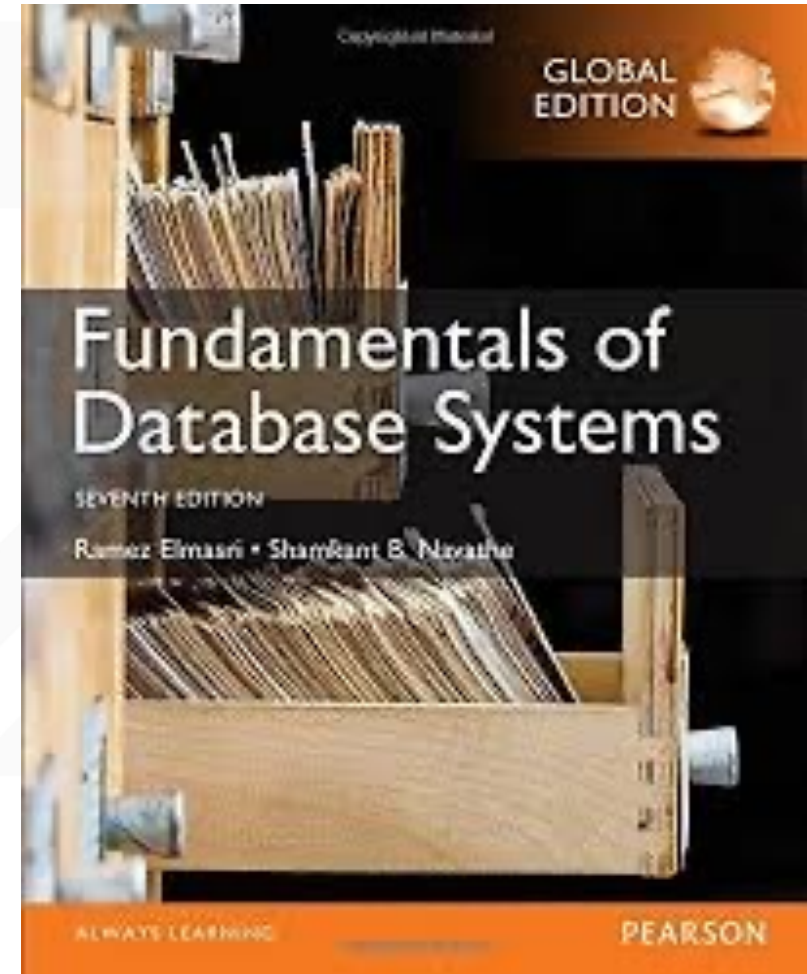
Database Systems

Program in Computer Engineering
Faculty of Engineering

King Mongkut's Institute of Technology Ladkrabang

Text

- Ramez Elmasri and Shamkant B. Navathe.
“**Fundamentals of Database Systems**”
7th Edition., Pearson, 2017



Chapter 7

More SQL:

Complex Queries, Views, and Schema Modification

modify ໄດ້ໂດຍ ສຳຄັນ ຈາກ ສຳນັກ

Outline

- More Complex SQL Retrieval Queries
- Views (Virtual Tables) in SQL
- Schema Modification in SQL

More Complex SQL Retrieval Queries

- Additional features allow users to specify more complex retrievals from database:
 - Nested queries, *query for query*
 - 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 *မသိ*
 - Unavailable or withheld value *မရသေး*
 - Not applicable attribute *မသက်သက်*
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
 - **TRUE**, **FALSE**, and **UNKNOWN** (like Maybe)
- **NULL = NULL comparison is avoided**
မသိမသိမသိမသိမသိမသိ



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		



- 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;

~~~~~  
 Checkว่า เป็น NULL ใช่มั้ย



# Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries** *หรือ query ที่ sub query ใ้*
  - Complete select-from-where blocks within WHERE clause of another query
  - **Outer query and nested subqueries**
- **Comparison operator IN** *ซึ่งอาจมีได้มากกว่า 1*
  - Compares value  $v$  with a set (or multiset) of values  $V$
  - Evaluates to TRUE if  $v$  is one of the elements in  $V$

*Complex structure*

*IN = ตรวจสอบว่า  $v$  อยู่ใน set หรือไม่*

Q4A:

SELECT  
FROM  
WHERE

*ค้นหาคำว่า*  
DISTINCT Pnumber *นำลงข้างล่าง*  
PROJECT

Pnumber **IN**  
( SELECT Pnumber  
FROM PROJECT, DEPARTMENT, EMPLOYEE  
WHERE Dnum=Dnumber AND  
Mgr\_ssn=Ssn AND Lname='Smith' )

Select the project number of projects that have an employee with last name 'Smith' involved as **manager**.

**OR** = union

Pnumber **IN**  
( SELECT Pno  
FROM WORKS\_ON, EMPLOYEE  
WHERE Essn=Ssn AND Lname='Smith' );

Select the project number of projects that have an employee with last name 'Smith' involved as **worker**.

*2 3 ข้าง WO แล้ว ~ หาว่าชื่อ code หรือ*

↑ ใช้ tuple ในการเปรียบเทียบ

- 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' );
```

Select the ESSN of all employees who work the same (project, hours) combination on some project that employee 'John Smit' (whose SNN = '123456789') works on.

หาตัวเลข อย่างใดอย่างหนึ่ง ที่น้อยกว่า

- 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 );
```

any  
ค่าที่น้อยกว่า

10000

20000

30000

Return the name of employees whose salary is greater than the salary of all the employees in department 5.

ทำจนจบ Department 5

หลีกเลี่ยง

- **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 );

มีค่าเหมือนกัน

ชื่อซ้ำ ต้องระบุให้ชัดเจน

# 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:

**Q16A:**

|               |                                              |
|---------------|----------------------------------------------|
| <b>SELECT</b> | E.Fname, E.Lname                             |
| <b>FROM</b>   | EMPLOYEE <b>AS</b> E , DEPENDENT <b>AS</b> D |
| <b>WHERE</b>  | E.Ssn = D.Essn                               |
|               | <b>AND</b> E.Sex = D.Sex                     |
|               | <b>AND</b> E.Fname = D.Dependent_name;       |

*Handwritten red notes:* "และชื่อคน" (and name) with a bracket pointing to the two AND conditions.

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

# The EXISTS and UNIQUE Functions in SQL for correlating queries

- **EXISTS** function

ตรวจสอบว่ามีข้อมูลหรือไม่

- 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)** *check ว่ามี 1 เดียวไหม*

- Returns TRUE if there are no duplicate tuples in the result of query Q

ชื่อเหมือนกัน

**Q16B:** Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```

SELECT Fname, Lname
FROM Employee AS E
WHERE EXISTS (
    SELECT *
    FROM DEPENDENT AS D
    WHERE E.Ssn = D.Essn AND
          E.Sex = D.Sex AND
          E.Firstname = D.Dependent_name );
  
```

ชื่อเหมือนกัน

PK = FK



**Q6:** Retrieve the names of employees who have no dependents.

```
SELECT Fname, Lname
FROM Employee AS E
WHERE NOT EXISTS ( SELECT *
                     FROM DEPENDENT
                     WHERE Ssn = Essn );
```

คนที่ไม่มีลูก

คนที่ไม่มีลูก

ไม่มีลูก

ให้สืบหาคนที่ มีลูกบ้าง ผู้ใหม่ ถ้าหาแล้ว เข้าใจดี

**Q7:** List the names of managers who have at least one dependent.

**SELECT** Fname, Lname  
**FROM** Employee  
**WHERE EXISTS** (

มีลูกอย่างน้อย 1 คน

**SELECT** \*  
**FROM** DEPENDENT  
**WHERE** Ssn = Essn )

Select all DEPENDENT  
tuples related to an  
EMPLOYEE

**AND**  
**EXISTS**

ได้ SSN

**SELECT** \*  
**FROM** Department  
**WHERE** Ssn = Mgr\_Ssn )

Select all DEPARTMENT  
tuples managed by the  
EMPLOYEE

ได้ manager

# 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);

1, 2, 3

|                                   |                                     |
|-----------------------------------|-------------------------------------|
| Employee_name<br><del>LNAME</del> | Supervisor_name<br><del>LNAME</del> |
|                                   |                                     |

- Use qualifier AS followed by desired new name

- Rename any attribute that appears in the result of a query

Attribute  
 'Lname' → 'Employee\_name'

**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;

# Specifying Joined Tables in the FROM Clause of SQL

ชื่อเต็ม Natural Join

- **Joined table** คือ Table ที่ผสมกัน จาก table ในข้อนี้ Attribute 2 ก้อน  
 • Permits users to specify a table resulting from a join operation in the FROM clause of a query  
 join ของ 2 ตารางนี้ จะผสมกัน  
 • 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';

ใช้ Attribute นี้เป็นหลัก



# 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

why?

# INNER and OUTER Joins

FULL

- **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

Wing

# 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 ค่าเฉลี่ย
  - **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
- ระวัง รวมคนที่ไม่รู้ (null)  
 $AVG \neq \frac{SUM}{COUNT}$  → ระวังผลเป็น null ด้วย  
 ระวัง Group ที่จะมีค่าเป็น null condition ใด



# 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;

Salary (this going)

output

- 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;

ไม่สนใจ

- 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 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** บอกจำนวน column ที่สนใจ
  - Specifies grouping attributes
- **COUNT (\*)** counts the number of rows in the group

ทราบจำนวนค่าสิ่งๆ


# Examples of GROUP BY *หมายถึง attribute ที่ต้อง group*

- The grouping attribute must appear in the SELECT clause:

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

- 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;

*อันนี้*  *Project join WORKS\_ON Pnum = Pno*

- **HAVING clause** *เลือก condition ที่กลุ่ม*
  - 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;

```

Group by

work\_on

now Having

| 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           | <u>Pnumber</u> | ... | <u>Essn</u> | <u>Pno</u> | 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  |   |

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

Result of Q26  
(Pnumber not shown)

After applying the HAVING clause condition



# Combining the WHERE and the HAVING Clause

เวลาใช้ 2 ข้อนี้ ระวัง แล้วแยกกันนะ

- 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:**

พินิจพิจารณา  
ที่ลบบรรทัด

```

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

สปีด, น

X ไม่ได้ ควรทำตาม Query 28 มากกว่า

## Correct Specification of the Query:

- Note: the WHERE clause applies tuple by tuple whereas HAVING applies to entire group of tuples.  
 (Note: WHERE clause applies tuple by tuple, and HAVING applies to the whole group)

**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** (1 2 3 4)  
          ( **SELECT**    Dno  
          **FROM**    EMPLOYEE  
          **GROUP BY** Dno  
          **HAVING**    **COUNT** (\*) > 5)  
          (Note: DNO is the department number, and the inner query checks for departments with more than 5 employees)

# EXPANDED Block Structure of SQL Queries

ရှမ်း ပြန်လည်ရှာဖွေ Complex

```

SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ] condition မှတစ်ခု tuple
[ GROUP BY <grouping attribute(s)> ] ခွဲခြား
[ HAVING <group condition> ] မရှိပါ case မှတစ်ခု group
[ ORDER BY <attribute list> ]; စာလုံးများ ordered စာလုံးများ

```

Complex

# Views (Virtual Tables) in SQL

การใส่ condition

- 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

SQL optimizer หรือ  
Query optimizer


คือ แปลง SQL ที่ใส่เข้าไปให้เป็น SQL ที่เร็ว equivalence กัน หรือ  
ได้ผลลัพธ์ (เท่า/อ้อม) ตามต้องการ แล้วให้มันยุ่งกับตัวน้อยๆ ลดลง ใช้งานเร็วขึ้น

# 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                                   |
| สิ่งที่พบที่<br>จริง | 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;                                      |

- 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
- **Strategy 1: 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

ข้อเสีย

ถ้าทำ query ของ query ไม่เร็ว

ถ้าเยอะ

↓

หากให้เร็ว เพิ่มเนื้อที่ มากขึ้น

ใช้พื้นที่ มาก แทน ดำเนินการเร็ว

## • Strategy 2: View materialization

- Physically create a temporary view table when the view is first queried สื่อบรรณ (ใช้ฐาน base, ตาราง)
- 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



- Multiple ways to handle materialization:
  - **immediate update** strategy updates a view as soon as the base tables are changed *เมื่อมีทราอัปเดตที่ขึ้น ต่อไป อัปเดตที่ based table ด้วย*
  - **lazy update** strategy updates the view when needed by a view query *ถ้าไม่มีทราใช้จน แล้วจะเกิดทรเปลี่ยน / ไม่เปลี่ยนแปลง ก็จะไม่มีการ อัปเดต เมื่อไหร่จะใช้ค่อย อัปเดต*
  - **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.  
*Update set เวลา ตามที่กำหนด*

# 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

ระวัง!!

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

**UV2: UPDATE**

**SET**

**WHERE**

**DEPT\_INFO**

**Total\_sal=100000**

**Dname='Research';**

ข้อมูลที่เกิดจากการคำนวณ  
ไม่สามารถ ปรับเปลี่ยนได้

cannot be processed because **Total\_sal** is a **computed value** in the view definition

- **View involving joins**

กรณี joins ของ view ต้องได้ข้อมูลจริง ไม่ขาด/Null

- Often not possible for DBMS to determine which of the updates is intended

- **Clause WITH CHECK OPTION**

ให้ check ภายหลังได้

- 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

# Views as authorization mechanism

ใครที่สามารถเห็นได้

- 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;
```

# Schema Change Statements in SQL

- **Schema evolution commands** *เปลี่ยนโครงสร้างได้*
  - 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;`

ลบล้าง  
ให้ drop table ที่ใช้ข้อมูลค้างอยู่
  - This removes the schema and all its elements including tables, views, constraints, etc.

ไม่สามารถ drop ถ้ามีอะไรอยู่ค้างอยู่

↑  
การเปลี่ยนแปลง

# 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.

เปลี่ยนชื่อ table ก่อน drop column

**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

สรุปก่อน

**Table 7.2** Summary of SQL Syntax

|                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>CREATE TABLE &lt;table name&gt; ( &lt;column name&gt; &lt;column type&gt; [ &lt;attribute constraint&gt; ]<br/> { , &lt;column name&gt; &lt;column type&gt; [ &lt;attribute constraint&gt; ] }<br/> [ &lt;table constraint&gt; { , &lt;table constraint&gt; } ] )</p>                                                                                                                                                          |
| <p>DROP TABLE &lt;table name&gt;</p>                                                                                                                                                                                                                                                                                                                                                                                              |
| <p>ALTER TABLE &lt;table name&gt; ADD &lt;column name&gt; &lt;column type&gt;</p>                                                                                                                                                                                                                                                                                                                                                 |
| <p>SELECT [ DISTINCT ] &lt;attribute list&gt;<br/> FROM ( &lt;table name&gt; { &lt;alias&gt; }   &lt;joined table&gt; ) { , ( &lt;table name&gt; { &lt;alias&gt; }   &lt;joined table&gt; ) }<br/> [ WHERE &lt;condition&gt; ]<br/> [ GROUP BY &lt;grouping attributes&gt; [ HAVING &lt;group selection condition&gt; ] ]<br/> [ ORDER BY &lt;column name&gt; [ &lt;order&gt; ] { , &lt;column name&gt; [ &lt;order&gt; ] } ]</p> |
| <p>&lt;attribute list&gt; ::= ( *   ( &lt;column name&gt;   &lt;function&gt; ( ( [ DISTINCT ] &lt;column name&gt;   * ) ) )<br/> { , ( &lt;column name&gt;   &lt;function&gt; ( ( [ DISTINCT ] &lt;column name&gt;   * ) ) ) } ) )</p>                                                                                                                                                                                            |
| <p>&lt;grouping attributes&gt; ::= &lt;column name&gt; { , &lt;column name&gt; }</p>                                                                                                                                                                                                                                                                                                                                              |
| <p>&lt;order&gt; ::= ( ASC   DESC )</p>                                                                                                                                                                                                                                                                                                                                                                                           |
| <p>INSERT INTO &lt;table name&gt; [ ( &lt;column name&gt; { , &lt;column name&gt; } ) ]<br/> ( VALUES ( &lt;constant value&gt; , { &lt;constant value&gt; } ) { , ( &lt;constant value&gt; { , &lt;constant value&gt; } ) }<br/>   &lt;select statement&gt; )</p>                                                                                                                                                                 |

**Table 7.2** Summary of SQL Syntax

---

DELETE FROM <table name>  
[ WHERE <selection condition> ]

---

UPDATE <table name>  
SET <column name> = <value expression> { , <column name> = <value expression> }  
[ WHERE <selection condition> ]

---

CREATE [ UNIQUE] INDEX <index name>  
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )  
[ CLUSTER ]

---

DROP INDEX <index name>

---

CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]  
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
- **CREATE VIEW** statement and materialization strategies
- Schema Modification for the DBAs using **ALTER TABLE** , **ADD** and **DROP COLUMN** , **ALTER CONSTRAINT** etc.

