204320 - Database Management

Chapter 5

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

Adapted for 204320

by Areerat Trongratsameethong

Addison-Wesley
is an imprint of
PEARSON

Chapter 5 Outline

- More Complex SQL Retrieval Queries
- More Complex SQL Insert, Update, and Delete
- Specifying Constraints as Assertions and Actions as Triggers
- Views (Virtual Tables) in SQL
- Schema Change Statements in SQL

More Complex SQL Retrieval Queries

- Additional features allow users to specify more complex retrievals from database:
  - Nested queries
  - Joined tables
  - Outer joins
  - Aggregate functions and grouping

Comparisons Involving NULL and Three-Valued Logic

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

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

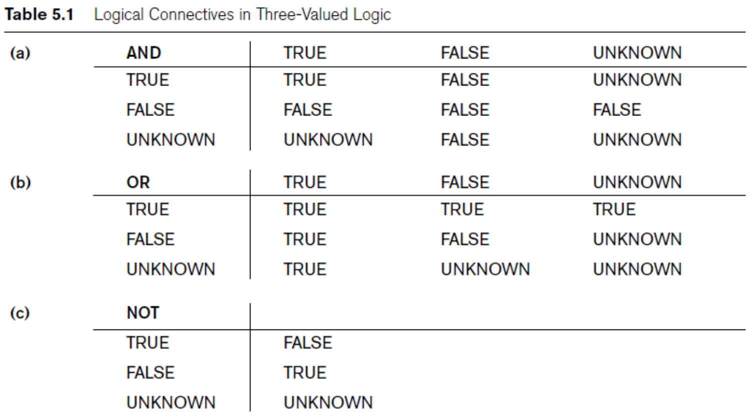
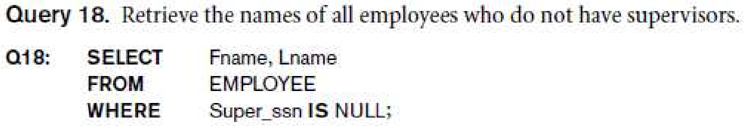


Table 5.1 Logical Connectives in Three-Valued Logic

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

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

- **SQL allows queries that check whether an attribute value is `NULL`:** สามารถใช้คำสั่ง SQL ตรวจสอบค่าที่เก็บอยู่ใน attribute ว่าเป็น NULL หรือไม่ โดยใช้คำสั่งดังแสดงด้านล่าง
  - `IS` or `IS NOT NULL`



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

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

Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries:** ประกอบด้วย 2 ส่วน คือ **query** ส่วนที่อยู่ด้านใน และ **query** ส่วนที่อยู่ด้านนอก
  - Complete select-from-where blocks within WHERE clause of another query: ส่วนที่อยู่ด้านใน คือ select-from-where ที่อยู่หลัง keyword WHERE
  - **Outer query:** query ส่วนที่อยู่ด้านนอก
- Comparison operator `IN`: operator ที่ใช้เปรียบเทียบระหว่าง outer และ inner
  - 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' );
```

แสดง **Pnumber** ซึ่งมีค่าไม่ซ้ำกัน

เปรียบเทียบ **Pnumber** กับ

ผลลัพธ์ที่ได้จาก **nested SELECT** ส่วนที่ **1**

OR

ผลลัพธ์ที่ได้จาก **nested SELECT** ส่วนที่ **2**

ตัวอย่างเช่น

ผลลัพธ์ **(Outer)**

Pnumber
1
2
3
4
5
…
n

ผลลัพธ์ **(nested/inner)** ส่วนที่ **1**

Pnumber
1
2

ผลลัพธ์ **(nested/inner)** ส่วนที่ **2**

Pno
2
5

ผลลัพธ์สุดท้าย

Pnumber
1
2
5

Nested Queries (cont'd.)

- Use tuples of values in comparisons
  - Place them within parentheses: หากต้องการเปรียบเทียบมากกว่า 1 attribute ให้ใส่วงเล็บ

<u>หมายเหตุ:</u> ตอน Select มาทุก field (รวมทั้ง Pno และ Hours) แต่ผลลัพธ์สุดท้ายให้แสดงเฉพาะ field ที่ระบุหลัง keyword SELECT

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

The following SQL statement selects all customers with a City of "Paris" or "London ".

```
SELECT  *                               // แสดงทุก fields ของ Customers
 FROM   Customers
WHERE  City IN ('Paris','London');      // เปรียบเทียบ field City ที่ได้กับ Paris และ London
```

Nested Queries (cont'd.)

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

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

ผลลัพธ์คือ พนักงานที่มีเงินเดือนมากกว่าทุกคนที่อยู่ในแผนกหมายเลข **5**

```
WHERE     Salary = ANY (SELECT …) is the same as
WHERE     Salary = SOME (SELECT …) is the same as
WHERE     Salary IN (SELECT …)
```

ถ้าใช้ **Salary > SOME** หรือ
Salary > ANY
ผลลัพธ์คือ พนักงานที่มีเงินเดือนมากกว่าบางคนที่อยู่ในแผนกหมายเลข **5**

Nested Queries (cont'd.)

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

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

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

ใช้ **join** แทนได้

E.Fname จาก **outer**

D.Dependent_name จาก **inner**

```
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 Queries

- **Correlated** nested query
  - Evaluated once for each tuple in the outer query
- EXISTS function
  - Check whether the result of a correlated nested query is empty or not
  - ***ตรวจสอบ outer กับ inner ถ้าเจอใน inner แค่ 1 ก็ return True ไม่ต้องเช็คให้ถึง record สุดท้ายของ inner***

http://www.dba-oracle.com/t_in_vs_exists_sql.htm

The Oracle documentation notes that:
"If the selective predicate is in the subquery, then use IN. If the selective predicate is in the parent query, then use EXISTS."

```
 SELECT *
  FROM customers
WHERE EXISTS (SELECT  *
              FROM   order_details
             WHERE customers.customer_id = order_details.customer_id);
```

→ ลูกค้าที่มี **order** ใน **order_details** อะไรก็ได้ (ขอให้มีอย่างน้อย **1 order)**

The EXISTS and UNIQUE Functions in SQL

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

```
SELECT  *
  FROM  Customers
WHERE  NOT EXISTS (SELECT   *
                          FROM  order_details
                        WHERE  customers.customer_id = order_details.customer_id);
```

→ ลูกค้าที่ไม่มี **order** ใน **order_details**

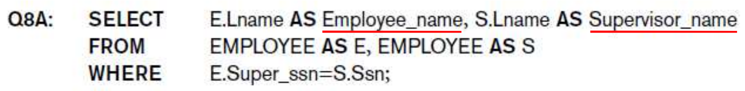
- SQL function `UNIQUE(Q)`
  - Returns `TRUE` if there are no duplicate tuples in the result of query Q

```
SELECT  *
  FROM  Student as S
WHERE  Unique (SELECT   CourseID
                     FROM   Enroll as E
                   WHERE   S.StudentID = E.StudentID);
```

→ นักศึกษาที่ลงทะเบียนแต่ละรายวิชาครั้งเดียว

Explicit Sets and Renaming of Attributes in SQL

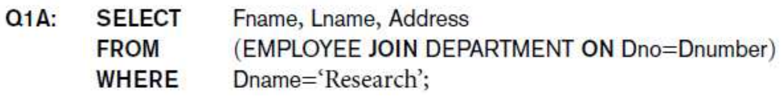
- Can use explicit set of values in WHERE clause
- Use qualifier AS followed by desired new name
  - **Rename** any attribute that appears in the result of a query



```
Q8A:   SELECT   E.Lname AS Employee_name, S.Lname AS Supervisor_name
       FROM     EMPLOYEE AS E, EMPLOYEE AS S
       WHERE    E.Super_ssn=S.Ssn;
```

Joined Tables in SQL and Outer Joins

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



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

```
SELECT       Customers.CustomerName, Orders.OrderID
  FROM       Customers INNER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY   Customers.CustomerName;
```

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

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

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

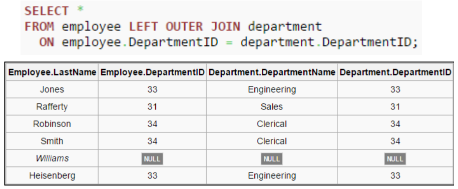
- **Inner join**
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation
- LEFT OUTER JOIN
  - Every tuple in left table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of right table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

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

Reference http://en.wikipedia.org/wiki/Join_(SQL)

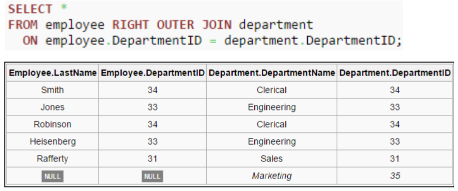
Employee table	
LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table	
DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing



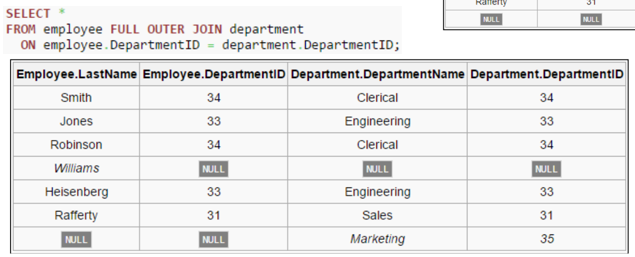
```
SELECT *
FROM employee LEFT OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	Engineering	33



```
SELECT *
FROM employee RIGHT OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*



```
SELECT *
FROM employee FULL OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

More Complex SQL - Insert, Update, Delete

- More complex of **insert** command:

```
INSERT INTO   first_table_name [(column1, column2, ... , columnN)]
        SELECT   column1, column2, ..., columnN
         FROM   second_table_name
       [WHERE condition];
```

[ ]: optional

```
INSERT INTO   Customers (CustomerName, Country)
              SELECT    SupplierName, Country
                FROM    Suppliers;
```

```
INSERT INTO   Customers (CustomerName, Country)
               SELECT   SupplierName, Country
                 FROM   Suppliers
                WHERE   Country='Germany';
```

Reference:
http://www.w3schools.com/sql/

PostgreSQL: Insert multiple rows

```
INSERT INTO films (code, title, did, date_prod, kind) VALUES
    ('B6717', 'Tampopo', 110, '1985-02-10', 'Comedy'),
    ('HG120', 'The Dinner Game', 140, DEFAULT, 'Comedy');
```

Reference:
http://www.postgresqltutorial.com/postgresql-update-join/

More Complex SQL - Insert, Update, Delete

- More complex of **update** command:

```
UPDATE ips
    SET  countryid = (select countryid from country where ips.iso=country.iso);
```

```
UPDATE   country p, ips pp
     SET  pp.countryid = p.countryid
 WHERE   pp.iso = p.iso;
```

```
UPDATE  [table1_name] AS t1 INNER JOIN [table2_name] AS t2 ON t1.[column1_name] =  t2.[column1_name]
     SET  t1.[column2_name] = t2.[column2_name];
```

```
UPDATE   business AS b INNER JOIN business_geocode AS g ON b.business_id = g.business_id
     SET  b.mapx = g.latitude, b.mapy = g.longitude
 WHERE   (b.mapx = '' or b.mapx = 0) and g.latitude > 0;
```

PostgreSQL

```
UPDATE A
SET A.c1 = expresion
FROM B
WHERE A.c2 = B.c2;
```

Reference:
http://www.postgresqltutorial.com/postgresql-update-join/

Reference: http://dba.stackexchange.com/questions/21152/how-to-update-one-table-based-on-another-tables-values-on-the-fly

More Complex SQL - Insert, Update, Delete

- Additional features allow users to specify more complex of **delete** command:

```
DELETE   t1, t2
  FROM   t1 INNER JOIN t2 INNER JOIN t3
 WHERE   t1.id=t2.id AND t2.id=t3.id;
```

```
DELETE  a1, a2
  FROM  db1.t1 AS a1 INNER JOIN db2.t2 AS a2
 WHERE  a1.id=a2.id;
```

```
DELETE   w
  FROM   WorkRecord2 w INNER JOIN Employee e ON EmployeeRun = EmployeeNo
WHERE    Company = '1' AND Date = '2013-05-06'
```

```
DELETE   WorkRecord2, Employee
  FROM   WorkRecord2 INNER JOIN Employee ON (EmployeeRun = EmployeeNo)
 WHERE   Company = '1' AND Date = '2013-05-06';
```

PostgreSQL

Reference: http://www.postgresqltutorial.com/postgresql-update-join/

```
DELETE FROM films USING producers
  WHERE producer_id = producers.id AND producers.name = 'foo';
```

```
DELETE FROM films
  WHERE producer_id IN (SELECT id FROM producers WHERE name = 'foo');
```

Reference: http://stackoverflow.com/questions/16481379/how-to-delete-using-inner-join-with-sql-server

Aggregate Functions in SQL

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

Aggregate Functions in SQL (cont'd.)

- **NULL values discarded when aggregate functions are applied to a particular column:** ค่า **NULL** จะไม่ถูกรวมเมื่อ **aggregation function** ใช้กับ **column** เช่น ถ้า **Salary** เป็น **NULL** และใช้ ฟังก์ชัน **Average** ค่า **NULL** จะไม่ถูกรวม

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

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

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';
```

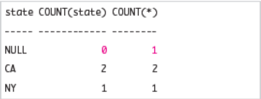
```
Q22:  SELECT   COUNT (*)
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    DNO=DNUMBER AND DNAME='Research';
```

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

Grouping: The GROUP BY and HAVING Clauses

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

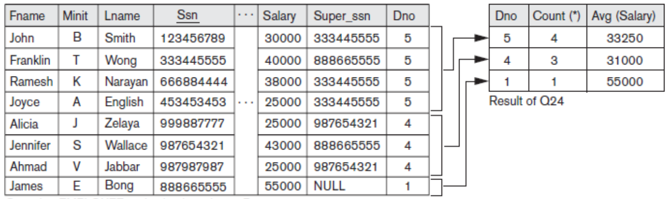
```
   SELECT   state, COUNT(state),
              COUNT(*)
     FROM   publishers
GROUP BY   state;
```



```
state COUNT(state) COUNT(*)
----- ------------ --------
NULL             0        1
CA               2        2
NY               1        1
```

Query 24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

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



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

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

Result of Q24

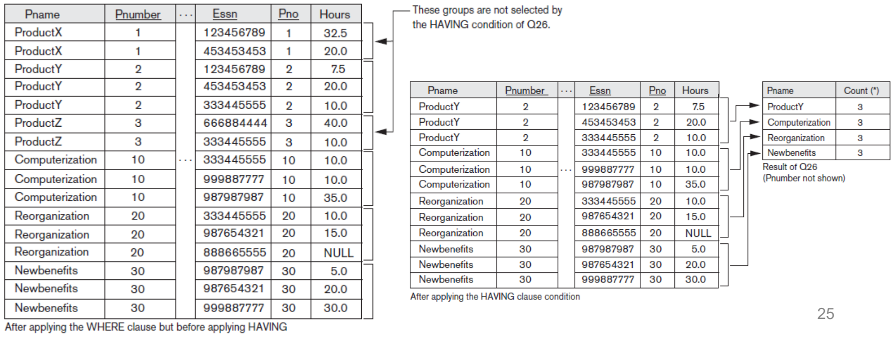
Grouping EMPLOYEE tuples by the value of Dno

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

- **HAVING** clause
  - Provides a **condition on the summary information**

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;
```



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

After applying the WHERE clause but before applying HAVING

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

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.)

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

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

Output = ?

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

Discussion and Summary of SQL Queries

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

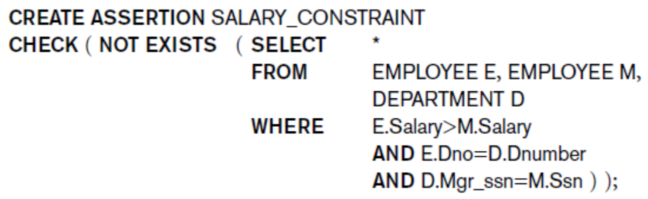
Specifying Constraints as Assertions and Actions as Triggers

- **CREATE ASSERTION**
  - Specify additional types of constraints outside scope of built-in relational model constraints
- **CREATE TRIGGER**
  - Specify automatic actions that database system will perform when certain events and conditions occur

Specifying General Constraints as Assertions in SQL

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

General Syntax: CREATE ASSERTION <name> CHECK(<condition>)



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

Introduction to Triggers in SQL

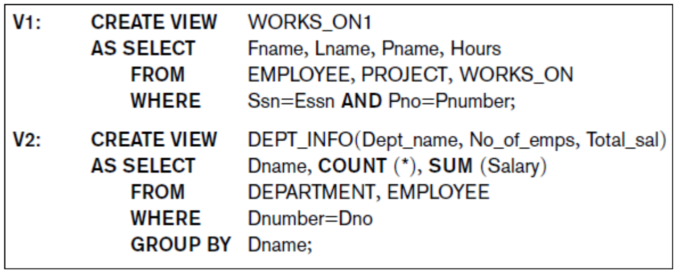
- CREATE TRIGGER statement
  - Used to monitor the database
- Typical trigger has three components:
  - **Event(s):** e.g. insert, update
  - **Condition**
  - **Action:** a sequence of SQL statements

```
   CREATE TRIGGER salary_trigger
BEFORE UPDATE ON employee_table REFERENCING NEW ROW AS n, OLD ROW AS o
    FOR EACH ROW IF n.salary <> o.salary THEN
…
END IF; ;
```

Reference: http://en.wikipedia.org/wiki/Database_trigger

Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables
  - Considered to be a virtual table
- **CREATE VIEW** command
  - Give table name, list of attribute names, and a query to specify the contents of the view



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

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

Specification of Views in SQL (cont'd.)

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

View Implementation, View Update, and Inline Views

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

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

QV1:   SELECT    Fname, Lname
       FROM      WORKS_ON1
       WHERE     Pname='ProductX';
```

```
SELECT    Fname, Lname
FROM      EMPLOYEE, PROJECT, WORKS_ON
WHERE     Ssn=Essn AND Pno=Pnumber
          AND Pname='ProductX';
```

View Implementation

- **View materialization approach**
  - Physically create a temporary view table when the view is first queried
  - Keep that table on the assumption that other queries on the view will follow
  - Requires efficient strategy for automatically updating the view table when the base tables are updated
- **Incremental update strategies**
  - DBMS determines what new tuples must be inserted, deleted, or modified in a materialized view table

View Update and Inline Views

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

```
(a):   UPDATE WORKS_ON
       SET        Pno =    ( SELECT    Pnumber
                             FROM      PROJECT
                             WHERE     Pname='ProductY' )
       WHERE      Essn IN  ( SELECT    Ssn
                             FROM      EMPLOYEE
                             WHERE     Lname='Smith' AND Fname='John' )
                  AND
                  Pno =    ( SELECT    Pnumber
                             FROM      PROJECT
                             WHERE     Pname='ProductX' );

(b):   UPDATE PROJECT    SET     Pname = 'ProductY'
       WHERE      Pname = 'ProductX';
```

การ **Update Base Table** ที่เกี่ยวข้องกับ **UV1 UV1** จะต้องถูก **Update** ด้วย ซึ่งอาจมีผลทำให้ **View** อื่นที่อ้างอิงถึง **UV1** แต่ใช้ข้อมูลต่างมุมมอง อาจจะได้ข้อมูลที่ไม่ถูกต้องได้

```
UV1:   UPDATE WORKS_ON1
       SET        Pname = 'ProductY'
       WHERE      Lname='Smith' AND Fname='John'
                  AND Pname='ProductX';
```

View Update and Inline Views (cont'd.)

- Clause `WITH CHECK OPTION`
  - Must be added at the end of the view definition if a view is to be updated

```
CREATE VIEW  VW_TechnicianEmployees
AS     SELECT  EmployeeID, Title, ManagerID
         FROM  HumanResources. Employee
        WHERE  Title LIKE '%technician%'
          WITH CHECK OPTION;
```

```
UPDATE  VW_TechnicianEmployees
    SET  Title = 'Chief'
 WHERE  EmployeeID=13
```

- **In-line view**
  - Defined in the `FROM` clause of an SQL query
  - ใช้ลดความซับซ้อนของ Query

Inline View Example http://www.orafaq.com/wiki/Inline_view

Schema Change Statements in SQL

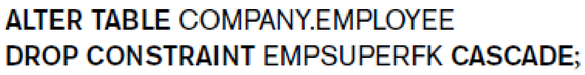
- **Schema evolution commands**
  - Can be done while the database is operational
  - Does not require recompilation of the database schema
- **`DROP` command**
  - Used to drop named schema elements, such as tables, domains, or constraint
- Drop behavior options:
  - `CASCADE` and `RESTRICT`
- Example:
  - `DROP SCHEMA COMPANY CASCADE;`
    - **CASCADE** Automatically drop objects (tables, functions, etc.) that are contained in the schema.
    - **RESTRICT** Refuse to drop the schema if it contains any objects. This is the default.

The ALTER Command

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

The ALTER Command (cont'd.)

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



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

Summary

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