Chapter 7

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

Table of Contents

7.1.1 Comparisons Involving NULL and Three-Valued Logic

- **It is possible for tuples to have a null value, denoted by NULL, for some of their attributes**
- **Meanings of NULL**
  - Value is unknown (age), Value is withheld (phone), Value is not applicable (Phd)
  - SQL does not distinguish between the different meanings of NULL
- **The result of any arithmetic expression involving NULL is NULL**
- **Any comparison with NULL returns UNKNOWN**
- **All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes**
  - avg, min, max, sum, count
  - **SELECT COUNT(*) AS** StuNum

    FROM STUDENT
- **The predicate 'IS NULL' or 'IS NOT NULL' can be used to check for NULL value of an attribute**
  - Query 18: Retrieve the names of all employees who do not have supervisors
  - **SELECT** Fname, Lname

    FROM EMP

    WHERE Super_ssn **IS NULL**;

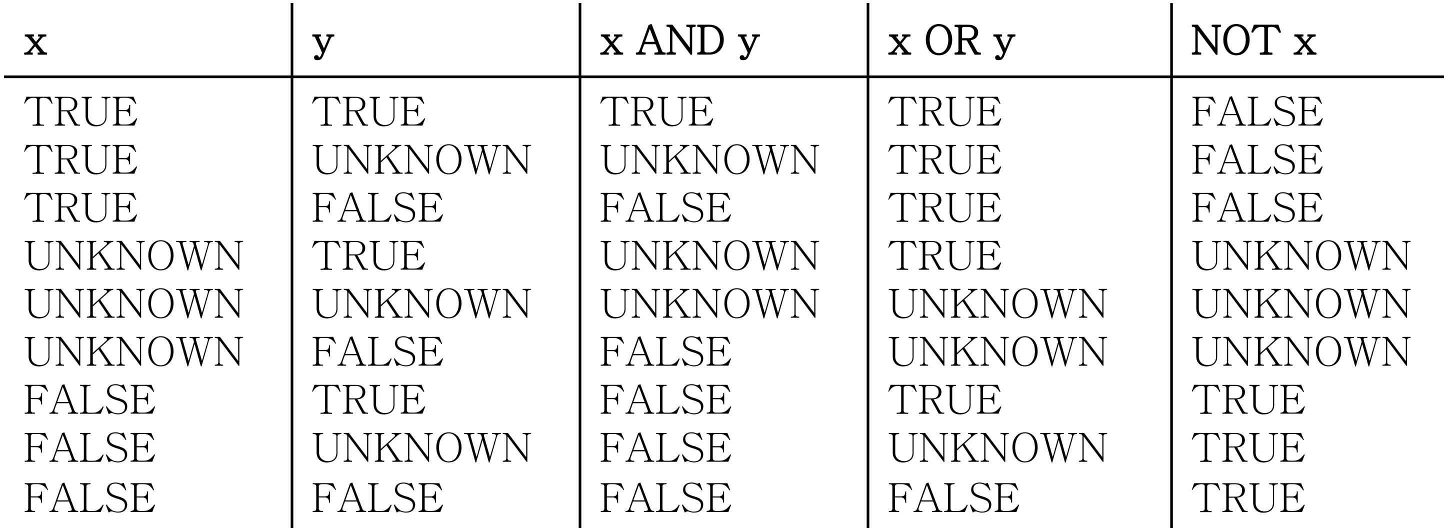
7.1.1 Comparisons Involving NULL and Three-Valued Logic

- **SQL uses a three-valued logic: TRUE, FALSE, UNKNOWN**
  - Result of WHERE clause predicate is treated as false if it evaluates to unknown
- **Table represents the values of the results of comparison conditions, which would typically appear in the WHERE clause of an SQL query**
  - If the truth-values of TRUE=1, FALSE=0, UNKNOWN=1/2

    Then AND of two truth-values is the **minimum** of those values

    OR of two truth-values is the **maximum** of those values

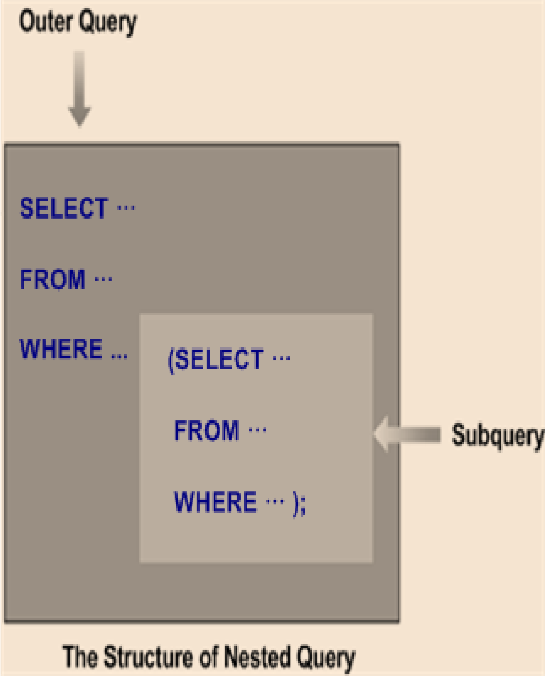
    Negation of truth-value V is **1-V**



x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

7.1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

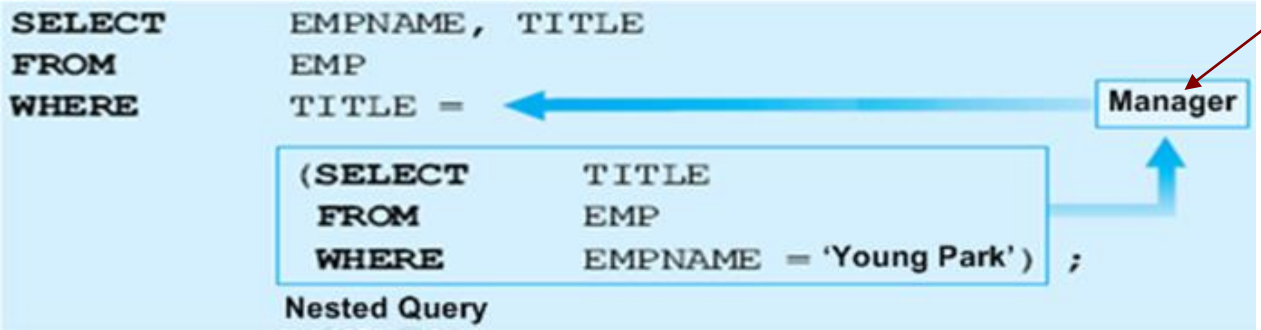
- **Nested queries (중첩 질의)**
  - SQL provides a mechanism for the nesting of subqueries
  - A nested query (or subquery) is a SELECT-FROM-WHERE expression that is nested within another query (outer query)
  - A nested query is usually used to return data that will be used in the outer query as a condition to further restrict the data to be retrieved
    - A common use of subqueries is to perform tests for set membership and set comparisons
  - Complex query can be expressed easily with several logical steps of a nested query



- **Nested queries can be used in various ways**
  - Subquery can return a single value and it can be compared with another value in WHERE clause of the outer query
  - Subquery can return a relation and it can be used in various ways in WHERE clause of the outer query
  - Subquery can have their relations appear in FROM clause of the outer query, just like any stored relation can

7.1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- **When a subquery returns a single value**
  - Subquery is evaluated before the outer query and its result is evaluated with a comparison operator (=,<,<=,>,>=,<>) in WHERE clause of the outer query
  - Ex: Retrieve the names and titles of all employees whose titles are the same as "Young Park"



```
SELECT     EMPNAME,  TITLE
FROM       EMP
WHERE      TITLE =
           (SELECT    TITLE
            FROM      EMP
            WHERE     EMPNAME = 'Young Park')  ;
```

⇒

EMPNAME	TITLE
Young Park	Manager
Min Cho	Manager

EMP

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
2106	Chang Kim	Staff	1003	2500000	2
3426	Young Park	Manager	4377	3000000	1
3011	Soo Lee	Director	4377	4000000	3
1003	Min Cho	Manager	4377	3000000	2
3427	Jong Choi	Staff	3011	1500000	3
1365	Sang Kim	Staff	3426	1500000	1
4377	Sung Lee	President	-	5000000	2

7.1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- **In general, the subquery will return a relation which is a set or multiset of tuples**

– The comparison operator IN, ANY(SOME), ALL, and EXISTS can be used in WHERE clause of the outer query

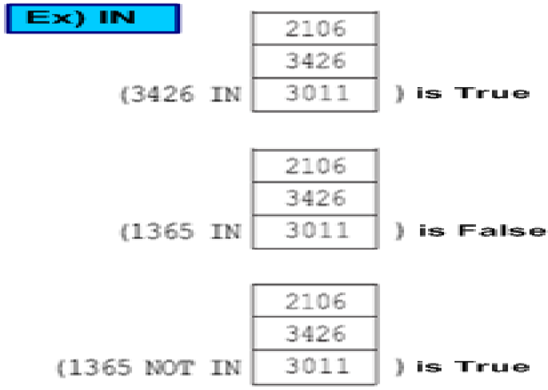
– The **IN** compares a value v with a set (or multiset) of values V and evaluates to TRUE if v is one of the elements in V
  - WHERE Dno IN (1,3)

– The **= 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) include {>, >=, <, <=, <>}
  - WHERE Dno = ANY (1,3), WHERE Dno > ANY (1,3)

– The **ALL** operator can also be combined with each of those operators
  - WHERE Dno > ALL (1,3), WHERE Dno <> ALL (1,3)

– The **EXISTS** operator returns the value TRUE if the subquery result contains at least one tuple (not empty)

– IN, ANY, ALL, EXISTS can be negated by putting **NOT** in front

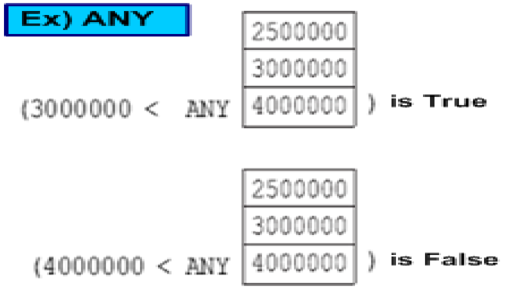


Ex) IN

2106
3426
(3426 IN 3011) is True

2106
3426
(1365 IN 3011) is False

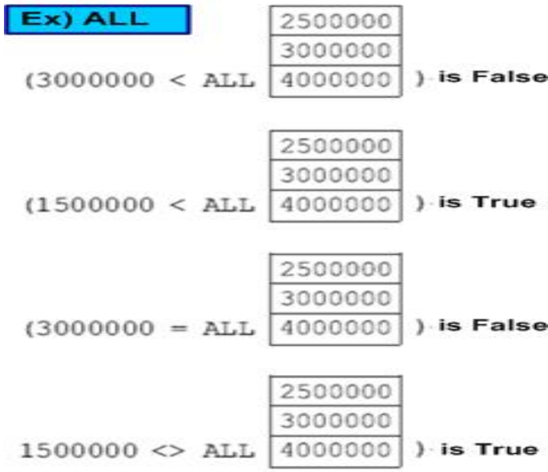
2106
3426
(1365 NOT IN 3011) is True



Ex) ANY

2500000
3000000
(3000000 < ANY 4000000) is True

2500000
3000000
(4000000 < ANY 4000000) is False



Ex) ALL

2500000
3000000
(3000000 < ALL 4000000) is False

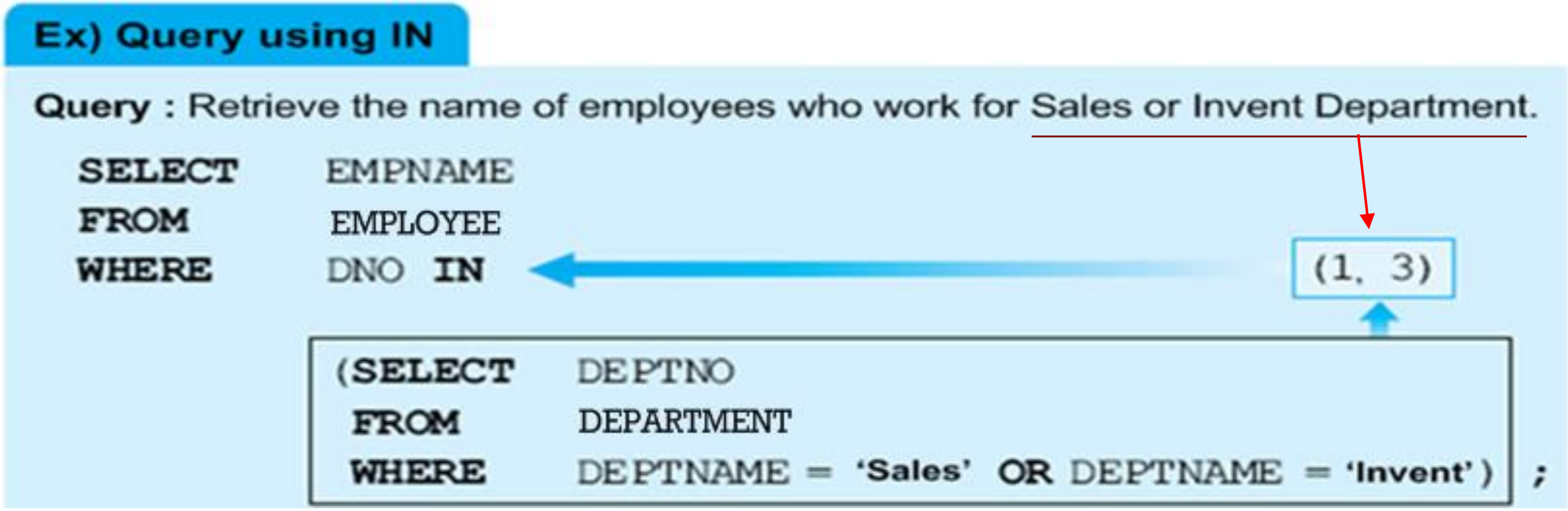
2500000
3000000
(1500000 < ALL 4000000) is True

2500000
3000000
(3000000 = ALL 4000000) is False

2500000
3000000
1500000 <> ALL 4000000) is True

7.1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- **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



Ex) Query using IN

Query : Retrieve the name of employees who work for Sales or Invent Department.

```
SELECT     EMPNAME
FROM       EMPLOYEE
WHERE      DNO  IN                                      (1, 3)

           (SELECT    DEPTNO
            FROM      DEPARTMENT
            WHERE     DEPTNAME = 'Sales'  OR DEPTNAME = 'Invent')  ;
```

EMPLOYEE

<u>EMPNO</u>	EMPNAME	TITLE	MANAGER	SALARY	DNO
2106	Chang Kim	Staff	1003	2500000	2
3426	Young Park	Manager	4377	3000000	1
3011	Soo Lee	Director	4377	4000000	3
1003	Min Cho	Manager	4377	3000000	2
3427	Jong Choi	Staff	3011	1500000	3
1365	Sang Kim	Staff	3426	1500000	1
4377	Sung Lee	President	-	5000000	2

DEPARTMENT

<u>DEPTNO</u>	DEPTNAME	FLOOR
1	Sales	8
2	Planning	10
3	Invent	9
4	General	7

7.1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- **Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a manager or as a worker of the department that controls the project.**
  - Q4:

```
(SELECT   DISTINT Pnumber  FROM PROJECT, DEPART, EMP
 WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn AND Lname='Smith')
 UNION
(SELECT   DISTINT Pno AS Pnumber FROM WORKS_ON, EMP
 WHERE    Essn=Ssn AND Lname='Smith');
```

- **Query 4 can be rephrased to use nested queries as shown in Q4A**
  - Q4A:

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

7.1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- **SQL allows the subtuple of values in comparisons by place them <u>within parentheses</u>**
  - Ex: Select Essns of all employees who work for the same (project, hours) combination on some project that employee 'John Smith' whose Ssn='12345' works on

  -
    ```
    SELECT   DISTINCT Essn
     FROM     WORKS_ON
     WHERE  (Pno, HOURS) IN (SELECT   Pno, Hours
                              FROM     WORKS_ON
                              WHERE Essn=‘12345’);
    ```
- **We can use explicit set of values in WHERE clause**
  - Query 17: Retrieve the SSNs of all employees who work on project numbers 1, 2 or 3

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

7.1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- **Find the name of employees whose salary is grater than the salary of all the employees in department 5**

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

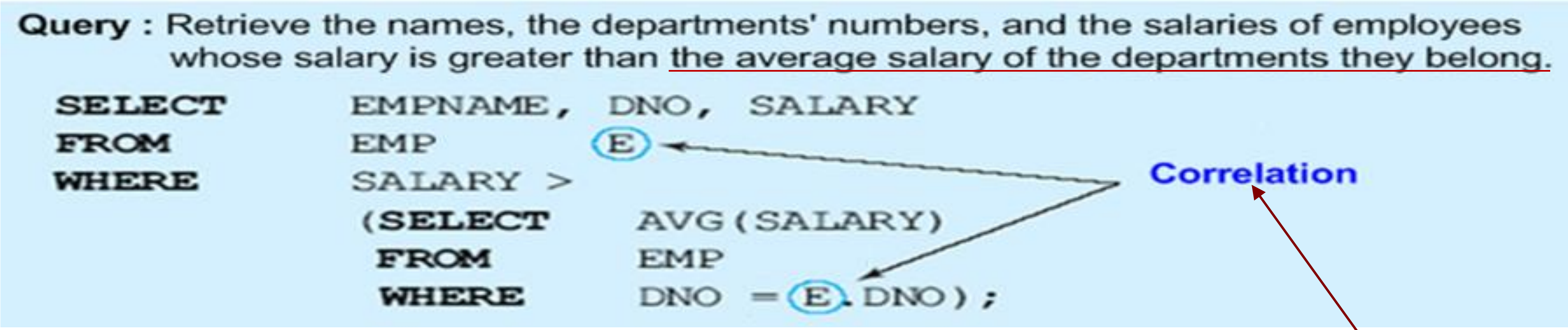
- **In general, we can have several levels of nested queries**
  - The scoping rule is that a reference to an unqualified attribute refers to the relation declared in the innermost nested query
  - It is generally advisable to create tuple variables (aliases) for all the tables referenced in an SQL query to avoid potential errors and ambiguities
  - Query 16: Retrieve the names of each employee who has a dependent with the same first name and the same sex as the employee

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

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

7.1.3 Correlated Nested Queries

- **Correlated nested query (상관 중첩 질의)**
  - Correlated subquery (synchronized subquery) is a subquery that contains a reference to an attribute in the outer query
  - The subquery is evaluated once for each tuple processed by the outer query



Query : Retrieve the names, the departments' numbers, and the salaries of employees whose salary is greater than the average salary of the departments they belong.

```
SELECT      EMPNAME,  DNO,  SALARY
FROM        EMP       E
WHERE       SALARY >
            (SELECT     AVG(SALARY)
             FROM       EMP
             WHERE      DNO  = E.DNO);
```

  - Above English sentence can be rephrased to fit into a correlated nested query such that: For each salary of EMP tuple of the outer query, evaluate the nested query which retrieves the average salary values for all EMP tuples with the same DNO as that of EMP tuple of the outer query; if the salary value of EMP tuple of the outer query is greater than the average salary of the nested query, then select that EMP tuple of the outer query.

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
2106	Chang Kim	Staff	1003	100000	2
3426	Young Park	Manager	4377	300000	5
1003	Min Cho	Manager	4377	300000	2
1365	Sang Kim	Staff	3426	100000	5
4377	Sung Lee	President	-	500000	2

EMPNAME	DNO	SALARY
Sung Lee	2	50000
Young Park	5	30000

7.1.3 Correlated Nested Queries

- **Query 16: Retrieve the names of each employee who has a dependent with the same first name and the same sex as the employee**
  - For each Ssn of EMP tuple, evaluate the correlated nested query which retrieves the Essn values for all DEPENDENT tuples with the same first name and the same sex as those of EMP tuple; if the Ssn value of EMP tuple is in the result of the nested query, then select that EMP tuple
  - Q16:

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

- **A query written with nested SELECT-FROM-WHERE blocks and using the "= or IN" comparison operators can always be expressed as a single block query**
  - Retrieve the names of each employee who has a dependent with the same first name and the same sex as the employee
  - Q16A:

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

7.1.4 The EXISTS and UNIQUE Functions in SQL

- **EXISTS operator**
  - The EXISTS checks whether the result of a subquery is empty or not
  - EXIST and NOT EXISTS are typically used in conjunction with a correlated subquery
- **Query 16: Retrieve the names of each employee who has a dependent with the same first name and the same sex as the employee**

  -
    ```
    Q16: SELECT E.Fname, E.Lname
         FROM   EMP AS E
         WHERE E.Ssn IN (SELECT D.Essn
                         FROM   DEPENDENT AS D
                         WHERE  D.Dependent_name=E.Fname AND D.Sex=E.Sex);
    ```
  - For each EMP tuple, evaluate the correlated subquery which retrieves all DEPENDENT tuples with the same Essn, the same first name, and the same sex as those of EMP tuple; if at least one tuple EXISTS in the result of the nested query, then select that EMP tuple

  -
    ```
    Q16B: SELECT  E.Fname, E.Lname
          FROM    EMP AS E
          WHERE   EXISTS (SELECT *
                          FROM   DEPENDENT AS D
                          WHERE  D.Essn=E.Ssn AND
                                 D.Dependent_name=E.Fname AND D.Sex=E.Sex);
    ```

7.1.4 The EXISTS and UNIQUE Functions in SQL

- **Query 6: Retrieve the names of employees who have no dependents**
  - For each EMP tuple, the correlated subquery selects all DEPENDENT tuples whose Essn value matches the EMP Ssn; if the result is empty, no dependents are related to the employee, so we select that EMP tuple and retrieve its Fname and Lname
  -

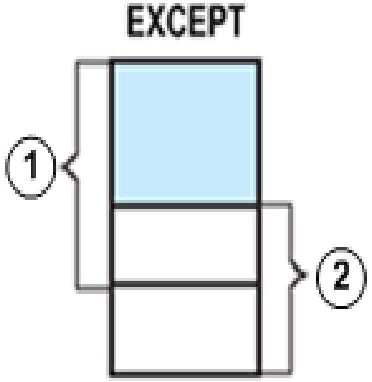
```
SELECT  Fname, Lname
FROM    EMP AS E
WHERE NOT EXISTS (SELECT  *
                  FROM    DEPENDENT AS D
                  WHERE   D.Essn=E.Ssn) ;
```

- **Query 7: List the names of managers who have at least one dependent**
  -

```
SELECT   Fname, Lname
FROM     EMP AS E
WHERE    EXISTS (SELECT *
                 FROM   DEPART AS R
                 WHERE  R.Mgr_ssn=E.Ssn) ;
         AND
         EXISTS (SELECT *
                 FROM   DEPENDENT AS D
                 WHERE  D.Essn=E.Ssn)
```

7.1.4 The EXISTS and UNIQUE Functions in SQL

- **Query 3A: Retrieve the name of each employee who works on all the projects controlled by department number 5**



  - Q3A:

```
Q3A: SELECT Fname, Lname
     FROM   EMP AS E
     WHERE NOT EXISTS (( SELECT  Pnumber
                         FROM    PROJ
                         WHERE   Dnum=5)          ①
                         EXCEPT (SELECT Pno
                                 FROM   WORKS_ON AS W      ②
                                 WHERE  W.Essn=E.Ssn));
```

① The first subquery (which is not correlated with the outer query) selects all projects controlled by department 5 => Let's say P1, P3

② The second subquery (which is correlated) selects all projects that the particular employee works on => Let's say P1, P2, P3

③ If ① - ② = (P1,P3)-(P1,P2,P3) = ∅, then it means that the employee works on all the projects controlled by department 5 and is therefore selected

- **UNIQUE(Q) function**
  - Returns TRUE if there are no duplicate tuples in the result of query Q
  - This can be used to test whether the result of a nested query is a set or a multiset

7.1.5 Subqueries in the From Clause

- **SQL allows a subquery expression to be used in a FROM clause (Inline view)**
- **Ex: Find all the producers' names of the movies in which Tom Cruise stars**
  - Movie(title, year, length, stuioName, producerID)
  - StarsIn(movieTitle, movieYear, starName)
  - Producer(Pname, address, PID, netWorth)
  -
    ```
    SELECT Pname FROM Producer, Movie, StarsIn
     WHERE PID=producerID AND title=movieTitle AND year=movieYear AND
               starname='Tom Cruse';
    ```
- **Above join query can be rephrased to use a subquery in the FROM clause**
  - If we have a relation that gives the producers' IDs of the movies in which Tom Cruse stars, then it would be a simple matter to look up the names of those producers in the Producer relation as shown in the following query
  -
    ```
    SELECT Pname
     FROM   Producer (SELECT producerID
                      FROM   Movie, StarsIn
                      WHERE title=movieTitle AND year=movieYear AND
                            starname='Tom Cruise') PIDforTomCruse
     WHERE PID = PIDforTom.producerID;
    ```

7.1.6 Joined Tables in SQL

- **The concept of a joined table (or joined relation) was incorporated into SQL to permit users to specify a table resulting from a join operation in the FROM clause of a query**
- **Query 1: Retrieve the name and address of all employees who work for the 'Research' department**
  - Q1: **SELECT** Fname, Lname, Address
    FROM EMP, DEPART
    WHERE Dno=Dnumber **AND** Dname='Research';
  - Q1A: **SELECT** Fname, Lname, Address
    FROM (EMP **JOIN** DEPART **ON** Dno=Dnumber)
    WHERE Dname='Research';
- **Multiway join**
  - Can nest join specifications; that is, one of the tables in a join may itself be a joined table
  - Allows the specification of the join of three or more tables as a single joined table
- **Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name**
  - Q2: **SELECT** Pnumber, Dnum, Lname **FROM** PROJ, DEPART, EMP
    WHERE Dnum=Dnumber **AND** Mgr_ssn=Ssn **AND** Plocation='Stafford';
  - Q2A: **SELECT** Pnumber, Dname, Lname
    FROM ((PROJ **JOIN** DEPART **ON** Dnum=Dnumber) **JOIN** EMP **ON** Mgr_ssn=Ssn)
    WHERE 'Plocation='Stafford';

7.1.6 Joined Tables in SQL and Outer Joins

- **Joined table (or joined relation)**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query
  - This construct may be easier to comprehend than mixing together all the join and select conditions in the WHERE clause
- **THETA JOIN**
  - *R* **JOIN** *S* **ON** (allows for arbitrary comparison relationships such as ≥)
  - Query 1: Retrieves the name and address of all employees who work for the 'Research' department
  - Q1:

```
Q1:  SELECT  Fname, Lname, Address
     FROM    EMP, DEPART
     WHERE   Dno=Dnumber AND  Dname='Research';
```

  - Q1A:

```
Q1A: SELECT  Fname, Lname, Address
     FROM    (EMP JOIN DEPART ON Dno=Dnumber)  // joined table
     WHERE   Dname='Research';
```

  - An *equijoin* is a theta join using the equality operator
- **NATURAL JOIN**
  - *R* **NATURAL JOIN** *S* is an implicit equijoin on attributes that have the same name from R and S, and retains only one copy of each common column
  - Q1B:

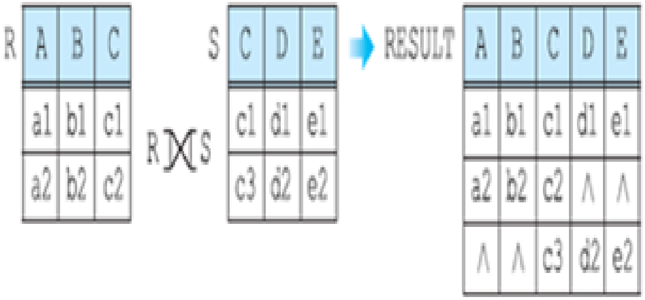
```
Q1B: SELECT Fname, Lname, Address
     FROM   (EMP NATURAL JOIN DEPART AS DEPT(Dname, Dno, Mssn, Msdate)
     WHERE Dname='Research' ;
```

  - The implied join condition of Q1B is EMP.Dno=DEPT.Dno

7.1.6 Joined Tables in SQL and Outer Joins

- **The key word CROSS JOIN is used to specify the CARTESIAN PRODUCT**
- **OUTER JOIN**
  - An extension of the join operation that avoids loss of information
  - It computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join
  - If the join attributes have the same name, one can also specify the natural join variation of outer joins, called a natural outer join

Full Outer Join



R

A	B	C
a1	b1	c1
a2	b2	c2

S

C	D	E
c1	d1	e1
c3	d2	e2

RESULT

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b2	c2	Λ	Λ
Λ	Λ	c3	d2	e2

- **MovieStar(name, address, birthdate), MovieExec(name, cert#, netWorth)**
- **FULL OUTER JOIN**
  - MovieStar **NATURAL FULL OUTER JOIN** MovieExec
    - Those representing individuals who are both stars and executives
    - An individual who is a star but not an executive
    - An executive who is not also a star
- **LEFT (RIGHT) OUTER JOIN**
  - MovieStar **NATURAL LEFT (RIGHT) OUTER JOIN** MovieExec
    - Every tuple in left (right) table must appear in result
    - If it does not have a matching tuple, it is padded with NULL values for the attributes of the right (left) table

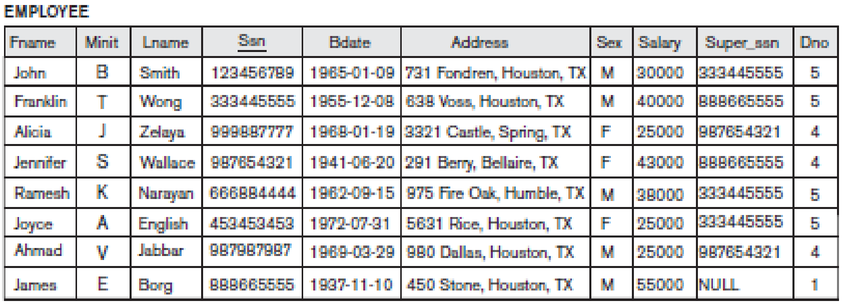
7.1.7 Aggregate Functions in SQL

- **Built-in aggregate functions are used to summarize information from multiple tuples into a single-tuple summary**
  - avg, min, max, sum, count(=number of values)
- **Aggregate functions can be used in the SELECT or HAVING clause**
  - Query 20: Find the sum of the salaries of all employees of the 'Research' department, as well as the average salary in this department
  - **SELECT** **SUM** (Salary), **AVG** (Salary)
    FROM EMP, DEPART
    WHERE Dno=Dnumber **AND** Dname='Research';
- **In general, NULL values are discarded when aggregate functions are applied to a particular attribute**
  - However, COUNT(*) returns the number of all the tuples in the result of the query
  - **Query 22: Retrieve the number of employees in the 'Research' department**
  - **SELECT COUNT** (*) **FROM** EMP, DEPART
    WHERE Dno=Dnumber **AND** DNAME='Research';
- **Aggregate functions can be used in selection conditions involving nested queries**
  - **Query 5: Retrieve the names of all employees who have two or more dependents**
  - **SELECT** Lname, Fname **FROM** EMP AS E
    WHERE (**SELECT COUNT**(∗) **FROM** DEPENDENT AS D **WHERE** D.Essn=E.Ssn) >=2;

7.1.8 Grouping: The GROUP BY and HAVING Clauses

- **There are circumstances where we would like to apply the aggregate function to a group of sets of tuples**
  - The attribute(s) given in the GROUP BY clause are used to form group
  - Aggregate function is applied to each such group independently
  - The grouping attributes specified by the GROUP BY clause should also appear in the SELECT clause
- **Query 24: For each department, retrieve the department number, the number of employees in the department, and their average salary**

  -
    ```
    SELECT      Dno, COUNT(*), AVG(Salary)
    FROM        EMP
    GROUP BY    Dno ;
    ```



EMPLOYEE

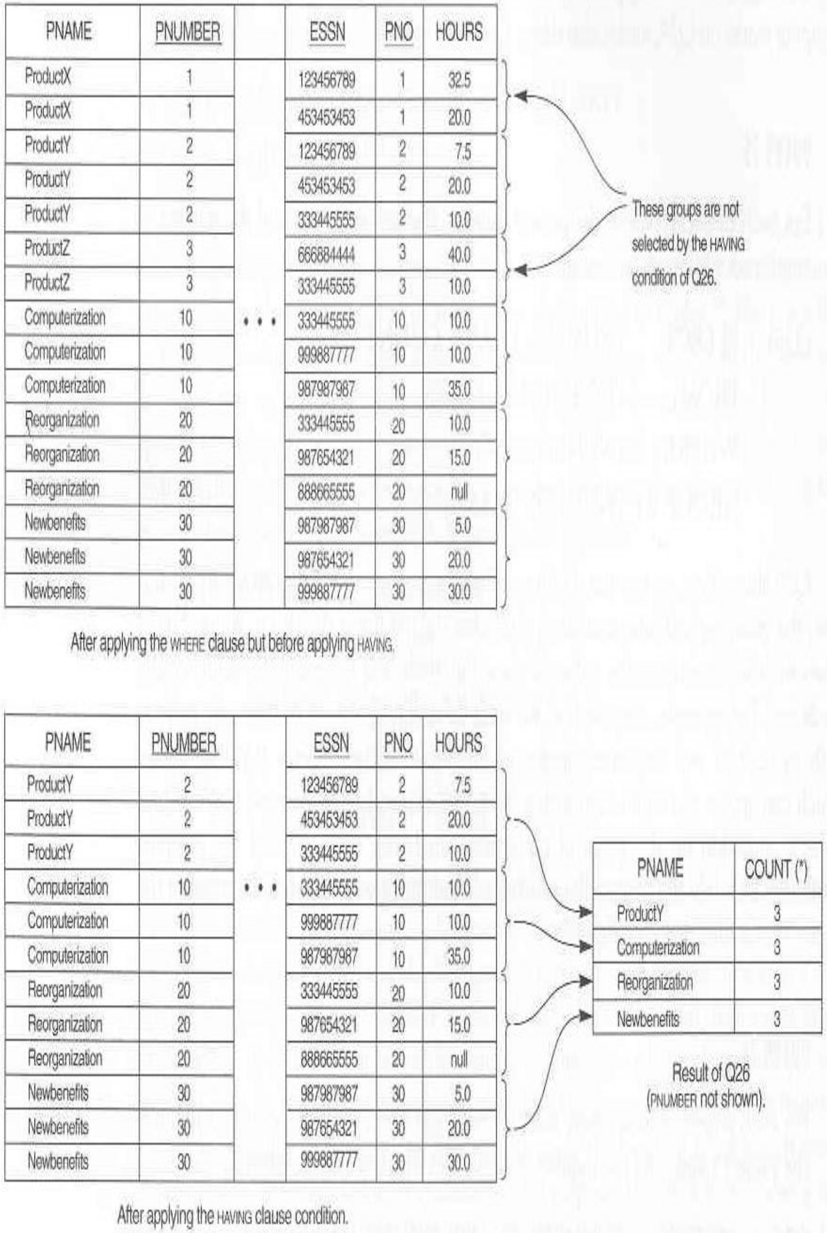
Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	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

- **If NULLs exist in grouping attribute**
  - Separate group created for all tuples with a NULL value in grouping attribute
- **Query 25: For each project, retrieve the project number, the project name, and the number of employees who work on that project**

  -
    ```
    SELECT      Pnumber, Pname, COUNT (*)
    FROM        PROJ, WORKS_ON
    WHERE       Pnumber = Pno
    GROUP BY    Pnumber;
    ```

7.1.8 Grouping: The GROUP BY and HAVING Clauses

- **HAVING clause**
  - Provides a condition on the GROUP BY clause
  - 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
  - **SELECT** Pnumber, Pname, **COUNT**(*)
    FROM PROJ, WORKS_ON
    WHERE Pnumber=Pno
    GROUP BY Pnumber
    HAVING **COUNT**(*) > 2 ;
  - Query 27: For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project
  - **SELECT** Pnumber, Pname, **COUNT**(*)
    FROM PROJ, WORKS_ON, EMP
    WHERE Pnumber = Pno **AND** Essn = Ssn **AND** Dno=5
    GROUP BY Pnumber;



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

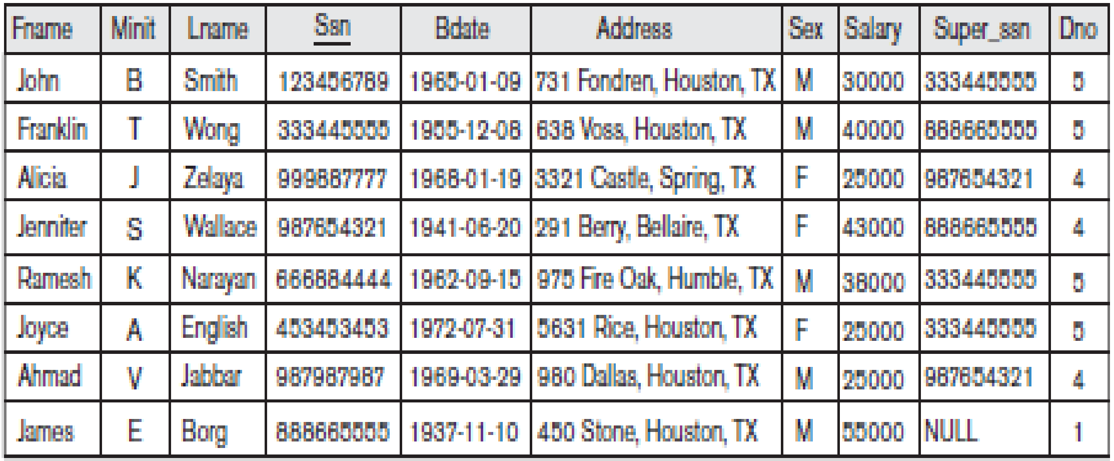
PNAME	COUNT (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26 (PNUMBER not shown).

After applying the HAVING clause condition.

7.1.8 Grouping: The GROUP BY and HAVING Clauses

- **Q28: For each department that has more than three employees, retrieve the department number and the number of its employees who are making more than $30,000 ⇒ (Dno : 5, Number of emp : 2)**
  - Following query is incorrect because it will select only department that have more than 3 employees who each earn more than $30,000
  -
```
SELECT      Dno. COUNT(*)
 FROM       EMP
 WHERE      Salary>30000
 GROUP BY Dno
 HAVING     COUNT(*) > 3;
```



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

  - One way to write Q28 correctly is to use the following nested query
  -
```
SELECT      Dno. COUNT(*)
 FROM       EMP
 WHERE      Salary>30000 AND Dno IN (SELECT     Dno
                                     FROM       EMP
                                     GROUP BY   Dno
                                     HAVING     COUNT(*) > 3)
 GROUP BY Dno
```

7.1.9 Other SQL Constructs: With and CASE

- **The WITH clause allows a user to define a table that will be used only in one query and then dropped**
- Q28'

```
Q28' WITH BIGDEPTS (Dno) AS
                (SELECT     Dno
                 FROM       EMP
                 GROUP BY Dno
                 HAVING    COUNT(*) > 3
         SELECT      Dno. COUNT(*)
         FROM        EMP
         WHERE      Salary>30000 AND Dno IN BIGDEPTS
         GROUP BY Dno
```

- **SQL also has a CASE construct which can be used when a value can be different based on certain conditions**
  - Supposes we want to give employees different raise amounts depending on which department they work for
  - U6':

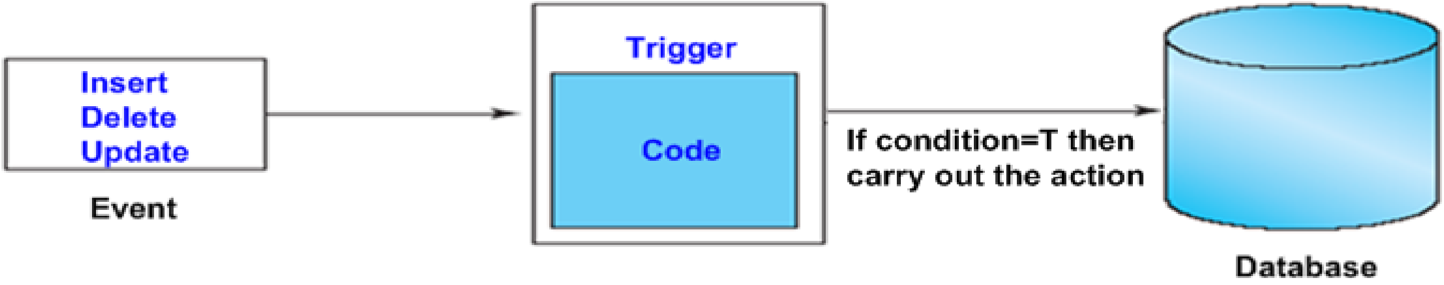
```
U6': UPDATE EMP
     SET Salary = CASE
                  WHEN Dno = 5 THEN Salary + 2000
                  WHEN Dno = 4 THEN Salary + 1500
                  ELSE  Salary + 1000;
                 END
```

7.1.10 Discussion and Summary of SQL Queries

- **Additional features allow users to specify more complex retrievals from database**
  - Nested queries, joined tables, aggregate functions, grouping
  - **SELECT** <attribute and function list>
    FROM <table list> [nested query]
    [**WHERE** <condition> [nested query] **]**
    [**GROUP BY** <grouping attribute(s)> **]**
    [**HAVING** <group condition> **]**
    [**ORDER BY** <attribute list> **];**
- **Conceptual sequence of a retrieval query**
  1) Cartesian product of relations in **FROM** statement
  2) Retrieve tuples which satisfy the condition of **WHERE** statement
  3) Grouping the tuples of (2)'s result by **GROUP BY** statement
  4) Retrieve some groups by applying a condition with **HAVING** statement
  5) Applying aggregation function to the remaining groups and ordering those attributed listed in **SELECT** statement

7.2 Specifying Constraints as Assertions and Actions as Triggers

- **Trigger and assertion are used to specify additional types of constraints that are outside the scope of the built-in relational model constraints such as primary keys, referential integrity, check constraints, etc**
- **TRIGGER**
  - The SQL statements that are executed automatically when a specified condition occurs during insert/delete/update operations



- **ASSERTION**
  - A Boolean-valued SQL expression that must be true at all times
  - DB modification is allowed unless it causes the assertion to become false
  - Trigger specifies the action to be executed when a DB operation violates a constraint; On the other hand, assertion is specified not to execute a DB operation which will violate a constraints
  - It is usually used to specify constraints which involve more than two relations
  - However, it should be used carefully because a complex assertion can cause considerable overhead

7.2.2 Introduction to Triggers in SQL

- **Triggers are usually used to enforce the business rules which may not be expressed by a built-in relational model constraints**
- **A trigger is a statement that is executed automatically by the system as a side effect of a modification to DB**
  - This type of functionality is generally referred to as active databases
- **Triggers follow an event-condition-action (ECA) model**

  1) Specify the **events** under which the trigger is examined
     - Database modification such as insert, delete, and update

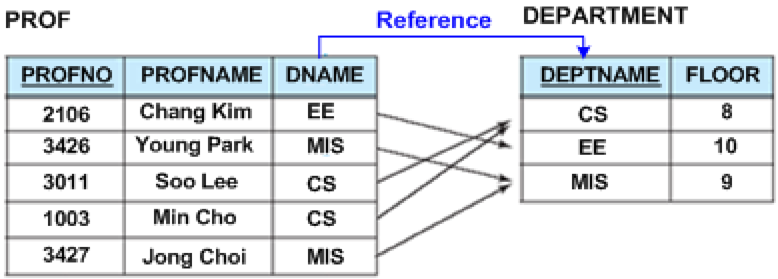
  2) Specify the **conditions** under which the trigger is to be executed.
     - Any true/false expression

  3) Specify the **actions** to be taken when the trigger executes.
     - Sequence of SQL statements that will be automatically executed, but it could be a DB transaction or an external program
- **There are three types of trigger-insert, delete, and update-and the number of each of type trigger to be declared to a table is zero or more**

7.2.2 Introduction to Triggers in SQL

- **TRIGGER statement**
  - CREATE TRIGGER <trigger_name>

```
{ BEFORE | AFTER | INSTEAD OF } { event [ OR … ] } ON <table_name>
  [ FOR EACH { ROW | STATEMENT } ]
  [ WHEN <condition>]
  BEGIN <SQL statement(s)> END
```



PROF

PROFNO	PROFNAME	DNAME
2106	Chang Kim	EE
3426	Young Park	MIS
3011	Soo Lee	CS
1003	Min Cho	CS
3427	Jong Choi	MIS

DEPARTMENT

DEPTNAME	FLOOR
CS	8
EE	10
MIS	9

  - If no condition is specified then condition is always true; Otherwise, a following condition is evaluated
    - **AFTER**: Executes the condition after the event: event->condition->action
    - **BEFORE**: Executes the condition before the event: condition->action->event
  - The optional keyword **FOR EACH ROW** specify that the rule action will be executed for each tuple that is affected by the triggering event
    - **OLD ROW** can be used to refer to a deleted tuple or to a tuple before it was updated
    - **NEW ROW** can be used to refer to a newly inserted or newly updated tuple
  - The optional keyword **FOR EACH STATEMENT (default)** specify that the rule action will be executed once for the triggering event, no matter how many tuples are affected
    - **OLD TABLE** or **NEW TABLE** can be used to refer to temporary tables containing all the affected tuples

7.2.2 Introduction to Triggers in SQL

- **Let's write a SQL trigger that applies to the MovieExec(name, address, certNo, netWorth) table so that it foils any attempt to lower the net worth of a movie executive**
  - It is necessary to write one trigger for the update event
  - The optional keyword **FOR EACH ROW** specify that the rule action will be executed for each tuple that is affected by the triggering event
    - **OLD ROW** is used to refer to a deleted tuple or to a tuple before it was updated
    - **NEW ROW** is used to refer to a newly inserted or newly updated tuple

```
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec              // Event
REFERENCING
    OLD ROW AS OldTuple,
    NEW ROW AS NewTuple
FOR EACH ROW
WHEN (OldTupe.netWorth > NewTuple.netWorth)        // Condition
    UPDATE MovieExec                               // Action
    SET netWorth = OldTuple.netWorth
    WHERE certNo = NewTuple.certNo;
```

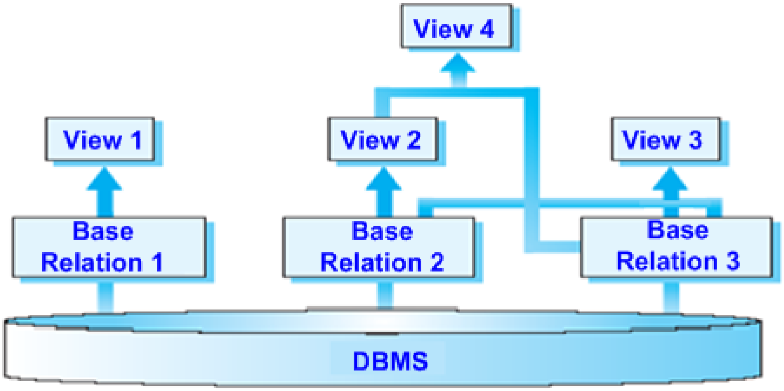
7.2.2 Introduction to Triggers in SQL

- **Let's write a SQL trigger that applies to the MovieExec(name, address, cert#, netWorth) table so that it prevents the average net worth of movie executives from dropping below $500,000.**
  - It is necessary to write one trigger for each of three events: insert, delete, and update of relation MovieExec. (But the following code shows the trigger for the update event)
  - The optional keyword **FOR EACH STATEMENT** specify that the rule action will be executed once for the triggering event, no matter how many tuples are affected
    - **OLD TABLE** or **NEW TABLE** can be used to refer to temporary tables containing all the affected tuples before the update or after the update, respectively

```
CREATE TRIGGER AveNetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec                    // Event
REFERENCING
    OLD TABLE AS OldStuff,
    NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))    //Condition
BEGIN                                                    // Action
    DELETE FROM MovieExec
    WHERE (name, address, certNo, netWorth) IN NewStuff;
    INSERT INTO MovieExec
    SELECT * FROM OldStuff;
END
```

7.3 Views (Virtual Tables) in SQL

- **Concept of a view in SQL**
  - A view is a virtual table derived from other tables which are base tables or previously defined views
  - A view does not necessarily exist in physical forms; This limits the possible update operations, but it does not provide any limitations on querying a view



- **We can think of a view as a way of specifying a table that we need to reference frequently, even though it may not exist physically**
  1) We may frequently issue queries that retrieve the employee name and the the project names that the employee works on
  2) Rather than having to specify the join of the three tables EMP, WORKS_ON, and PROJECT every time we issue this query, we can define a view that is specified as the result of these joins
  3) Then we can issue queries on the view, which are specified as single table retrievals rather than as retrievals involving three tables

7.3 Views (Virtual Tables) in SQL

- **CREATE VIEW command**
  - The view is given a table name, a list of attribute names, and a query to specify the contents of the view
  - CREATE VIEW <view_name> [(attribute(s))]
    AS <query expression>
    [WITH CHECK OPTION];
  - If none of the view attributes results from applying arithmetic or aggregation operations, we do not have to specify new attribute names for the view

```
CREATE VIEW EMP_Dno3
AS SELECT       Ename, Super_ssn
FROM            EMP
WHERE           Dno=3;
```

```
CREATE VIEW DEPTINFO(Dname, Totalsal)
AS SELECT Dname, SUM(Salary)
FROM         DEPART, EMP
WHERE        Dnumber=Dno
GROUP BY     Dname;
```

  - The clause WITH CHECK OPTION must be added for the system to keep for data consistency if a view is to be updated
    - If a tuple inserted or updated into the view does not satisfy the view's where clause condition, the insertion or update is rejected
- **DROP VIEW command**
  - DROP VIEW view_name [RESTRICT | CASCADE]

```
UPDATE EMP_Dno3
SET        Dno = 2
WHERE  Name = 'Kim';
```

7.3.2 Specification of Views in SQL

- **Main advantages of a view**
  - Views are used as a security and authorization mechanism because they can restrict users to see only specified columns and rows in a table
  - Several different views can be provided from the same data
  - Views are used to simplify the specification of complex queries

```
v2:  CREATE VIEW DEPT_INFO(Dname, No_of_emps, Total_sal)
      AS SELECT  Dname, COUNT(*), SUM(Salary)
      FROM       EMP, DEPT
      WHERE      Dno=Dnumber
      GROUP BY   Dname;
```

  - Views can also be used for data independence: Suppose that an EMP table is divided by two EMP1 and EMP2 tables for some reason. Then we cannot use an application program which use the EMP table. However, if we create an EMP view, then we can still use an application program

```
Ex: CREATE VIEW EMP
     AS SELECT EMP1.Name, EMP1.Bdate, EMP2.Dno
     FROM      EMP1, EMP2
     WHERE     EMP1.Ssn = EMP2.Ssn
```

7.3.3 View Implementation, View Update, and Inline Views

- **If a user modifies the base tables on which the view is defined, the view must automatically reflect these changes**

1. **Query modification approach**
   - If a user creates a view, only its definition is stored in a system catalog
   - For example, the left query would be automatically modified to the right query by the DBMS since it has a definition of the view EMP_Dno3

```
SELECT  *                       SELECT Name, Sex, Super_ssn
FROM    EMP_Dno3      ⇒         FROM   EMP
WHERE Sex='male';               WHERE Dno=3 AND Sex='male';
```

   - The disadvantage of this approach is that it is inefficient for views defined via complex queries that are time-consuming to execute

2. **View materialization approach**
   - Physically create a view table (materialized view) when the view is first queried
   - Thereafter, queries based on materialized view can be faster than re-computing view each time
   - Difficulty is maintaining the currency of view while base table(s) are being updated
     - The task of keeping a materialized view up-to-date with the underlying data is known as materialized view maintenance

7.3.3 View Implementation, View Update, and Inline Views

- **Updating of views is complicated and can be ambiguous**
  - In general, an update on a view defined on a single table without any aggregate functions can be mapped to an update on the underlying base table under certain conditions

1) An update on a view defined on a single table

INSERT INTO EMP_PUBLIC_Dno3 ⇒ **INSERT INTO** EMP
VALUES (Kim, male, 1324) **VALUES**(Kim, male, NULL … NULL, 1324)

2) An update on a view which includes aggregate functions does not make much sense

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

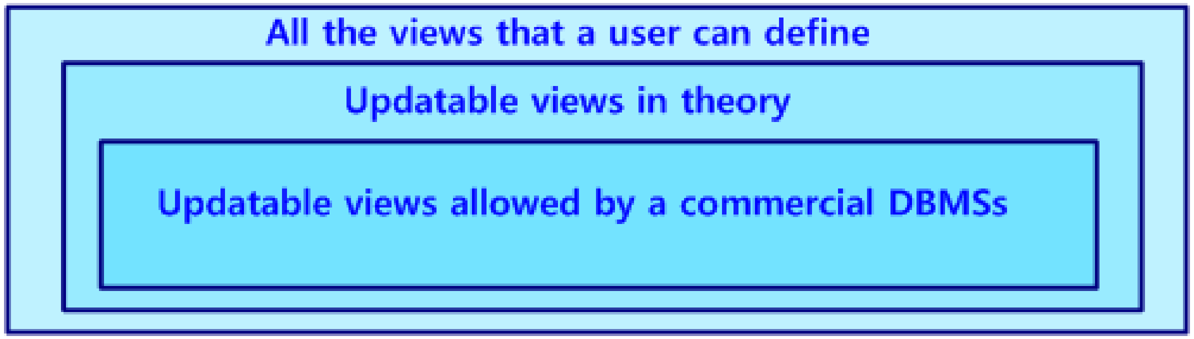
```
CREATE VIEW DEPT_INFO(Dname, No_of_emps, Total_sal)
AS SELECT   Dname, COUNT(*), SUM(Salary)
FROM        EMP, DEPT
WHERE       Dno=Dnumber
GROUP BY    Dname;
```

3) For a view involving joins, an update operation may be mapped to update operations on the underlying base relations in multiple ways

7.3.3 View Implementation, View Update, and Inline Views

- **In summary, we can make the following observations**

1. A view with a single defining table is updatable
   - If the view attributes contain the primary key of the base relation
   - All attributes with the NOT NULL constraint but have default values specified
2. Views defined using grouping and aggregate functions are not updatable
3. Views defined on multiple tables using joins are generally not updatable


All the views that a user can define

Updatable views in theory

Updatable views allowed by a commercial DBMSs

- **Inline view**
  - It is possible to define a view in the FROM clause of an SQL query
  -
```
 SELECT Pname
 FROM   Producer, (SELECT producerID FROM  StarsIn, Movie
                   WHERE movieTitle=title AND movieYear=year AND
                         starname='Tom Cruise') PIDforTom
 WHERE PIDforTom.producerID=PID;
```

7.4 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 or RESTRICT
    - Ex: **DROP SCHEMA** COMPANY **CASCADE**;
    - Ex: **DROP TABLE** DEPENDENT **RESTRICT**;
- **Alter table actions include:**
  - Adding or dropping a attribute. (CASCADE or RESTRICT for drop)
    - Ex: **ALTER TABLE** EMP
      ADD COLUMN Job VARCHAR(12);
  - Changing a column definition
    - Ex: **ALTER TABLE** DEPARTMENT **ALTER COLUMN** Mgr_ssn
      SET DEFAULT '3344555';
- **Change constraints specified on a table**
  - Add or drop a named constraint
    - Ex: **ALTER TABLE** EMPLOYEE
      DROP CONSTRAINT EMPSUPERFK CASCDE;

Summary

- **Complex SQL:**
  - Nested queries, joined tables, aggregate functions, grouping

- **ASSERTION and TRIGGER**
  - CREATE ASSERTION <name>

    CHECK (<condition>)   // usually EXISTS and NOT EXIST style of SQL condition
  - DB modification is allowed if the results of the condition is TRUE

- **Views**
  - Virtual or derived tables
  - Materialized view

- **Schema Change Statements in SQL**