Chapter 5

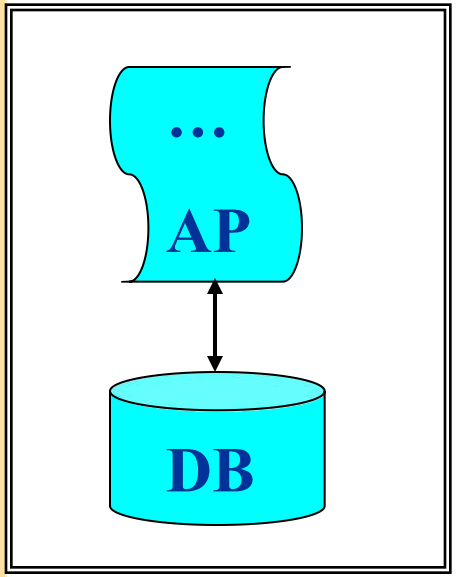
SQL: Advanced Queries, Assertions, Triggers, and Views

Manipulating DB by using SQL

DBA

construct

1. Define DB
   (SQL DDL)
2. Load data



DB System

DML

manipulate

User

SQL (Ch.4)

- **Data Definition Language**
  - **CREATE, DROP, ALTER**

SQL (Ch.4, 5)

- **Data Manipulation Language**
  - **Query: SELECT**
  - **Update: INSERT, DELETE, UPDATE**

Chapter Outline

- SQL Data Definition and Data Types
- Specifying Constraints in SQL
- Basic Retrieval Queries in SQL
- INSERT, DELETE, and UPDATE Statements in SQL
- More Complex SQL Queries
- Specifying Constraints as Assertions and triggers
- Views (Virtual Tables) in SQL
- Schema Change Statements in SQL

NULLS IN SQL QUERIES

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

Q14:

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    SUPERSSN IS NULL
```

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

NESTING OF QUERIES

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

Q1N:

```
SELECT  FNAME, LNAME, ADDRESS
FROM    EMPLOYEE
WHERE   DNO IN (SELECT  DNUMBER
                FROM    DEPARTMENT
                WHERE   DNAME='Research' )
```

DNUMBER
5

Q1:

```
SELECT  FNAME, LNAME, ADDRESS
FROM    EMPLOYEE, DEPARTMENT
WHERE   DNAME='Research' AND NUMBER=DNO
```

NESTING OF QUERIES (cont.)

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator **IN** compares a value *v* with a set (or multi-set) of values V, and evaluates to **TRUE** if *v* is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*
- In this example, the nested query is *not correlated* with the outer query

```
Q1N:  SELECT  FNAME, LNAME, ADDRESS
      FROM    EMPLOYEE
      WHERE   DNO IN (SELECT  DNUMBER
                      FROM    DEPARTMENT
                      WHERE   DNAME='Research' )
```

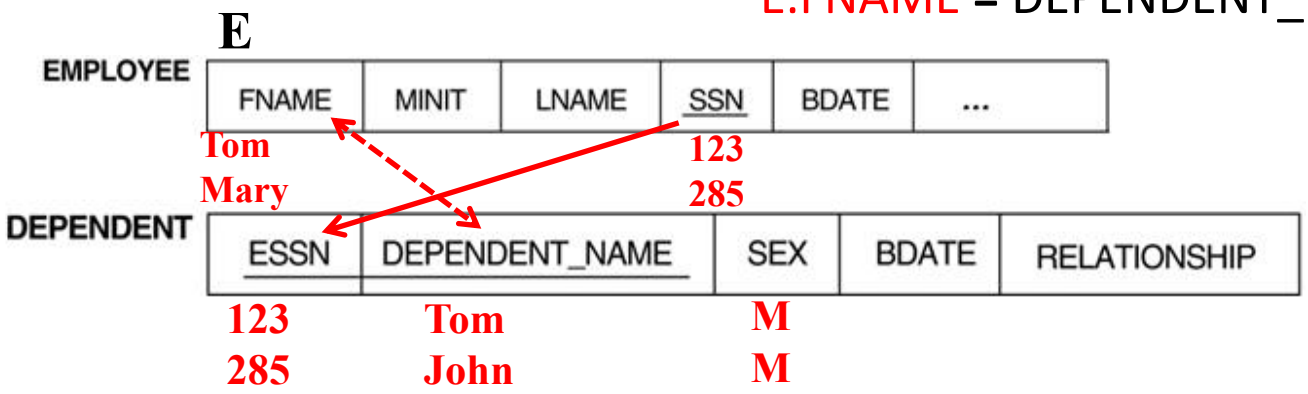
CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a ***nested query*** references an attribute of a relation declared in the ***outer query*** , the two queries are said to be ***correlated***
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*
- Query 16: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q16: SELECT  E.FNAME, E.LNAME
     FROM    EMPLOYEE AS E
     WHERE   E.SSN IN (SELECT ESSN
                       FROM   DEPENDENT
                       WHERE  ESSN = E.SSN AND
                              E.FNAME = DEPENDENT_NAME)
```

E.SSN: 123

ESSN
123

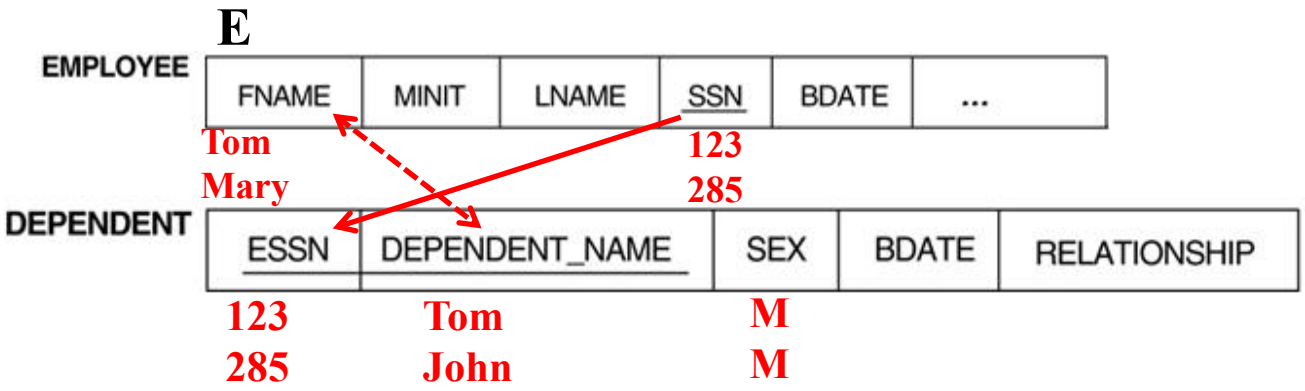


CORRELATED NESTED QUERIES

- In Q12, the nested query has a different result *for each tuple* in the outer query
- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query.

  For example, Q12 may be written as in Q12A

```
Q12A: SELECT   E.FNAME, E.LNAME
      FROM     EMPLOYEE E, DEPENDENT D
      WHERE    E.SSN = D.ESSN AND E.FNAME = D.DEPENDENT_NAME
```



CORRELATED NESTED QUERIES

- Most implementations of SQL ***do not*** have this operator
- The **CONTAINS** operator compares two *sets of values* , and returns TRUE if one set contains all values in the other set
  (reminiscent of the *division* operation of algebra).
  - Query 3: Retrieve the name of each employee who works on *all* the projects controlled by department number 5.

```
Q3: SELECT   FNAME, LNAME
    FROM     EMPLOYEE
    WHERE  ( (SELECT   PNO
              FROM     WORKS_ON
              WHERE    SSN=ESSN)
              CONTAINS
              (SELECT   PNUMBER
               FROM     PROJECT
               WHERE    DNUM=5) )
```

123

PNO
9
2
7

285

PNO
6
9
2

PNUMBER
9
2

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	...

123

WORKS_ON

ESSN	PNO	HOURS
123	9	

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM
	9		5

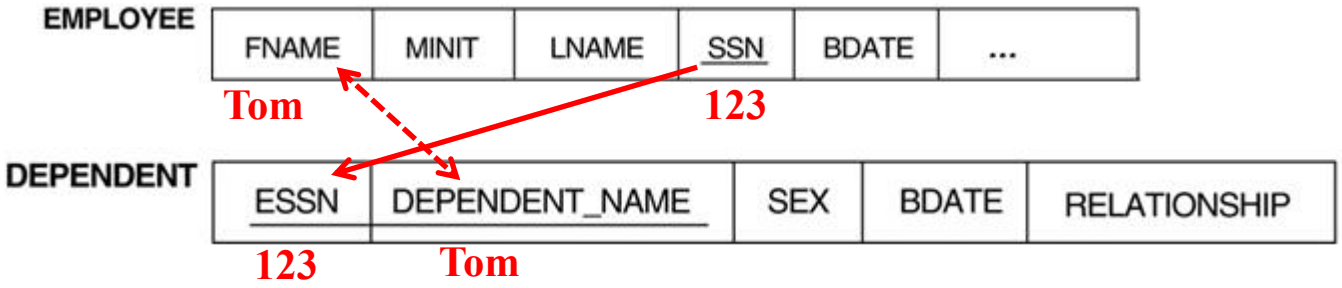
CORRELATED NESTED QUERIES (cont.)

- In Q3, the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5
- The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different *for each employee tuple* because of the correlation

THE EXISTS FUNCTION (cont.)

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12B: SELECT   FNAME, LNAME
      FROM     EMPLOYEE
      WHERE    EXISTS  (SELECT   *
                        FROM     DEPENDENT
                        WHERE    SSN = ESSN AND FNAME = DEPENDENT_NAME)
```

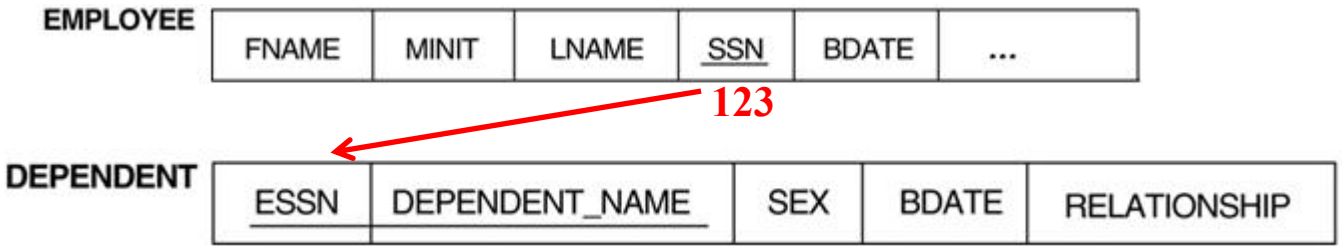


THE EXISTS FUNCTION

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

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

  - In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist* , the EMPLOYEE tuple is selected
  - EXISTS is necessary for the expressive power of SQL



EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 17: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q17: SELECT   DISTINCT ESSN
     FROM     WORKS_ON
     WHERE    PNO IN (1, 2, 3)
```

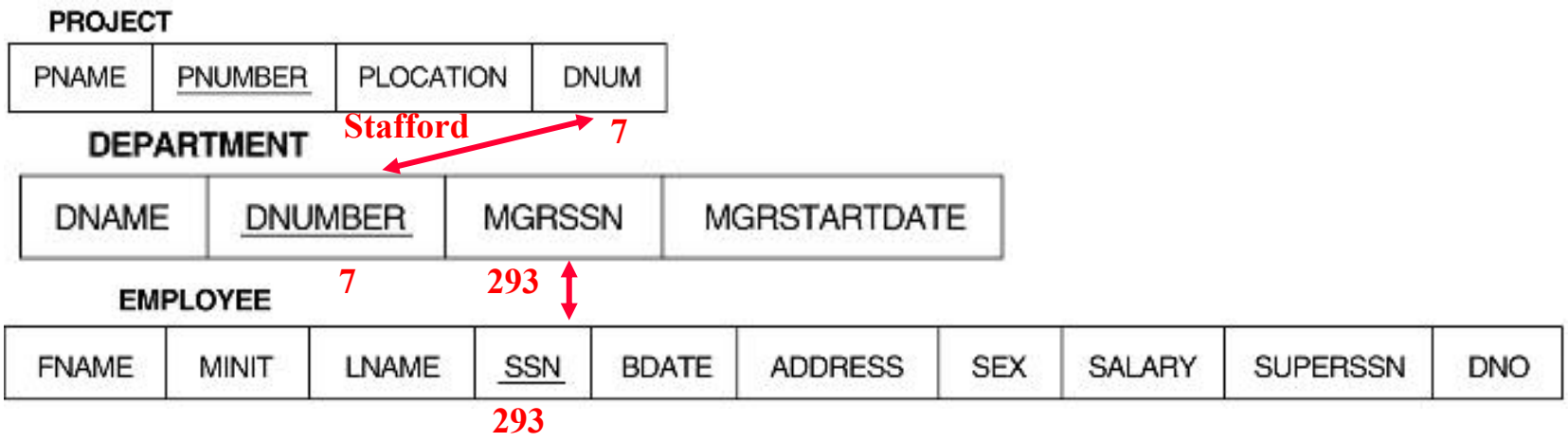
WORKS_ON

ESSN	PNO	HOURS
123	2	
123	9	
285	9	
310	1	
310	3	
…	…	

Joined Relations Feature in SQL2

- Can specify a "joined relation" in the FROM-clause
- Looks like any other relation but is the result of a join
- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

Q2:
```
SELECT   PNUMBER, DNUM, LNAME, BDATE, ADDRESS
FROM     ((PROJECT JOIN DEPARTMENT ON DNUM=DNUMBER)
         JOIN EMPLOYEE ON MGRSSN=SSN) )
WHERE    PLOCATION='Stafford'
```

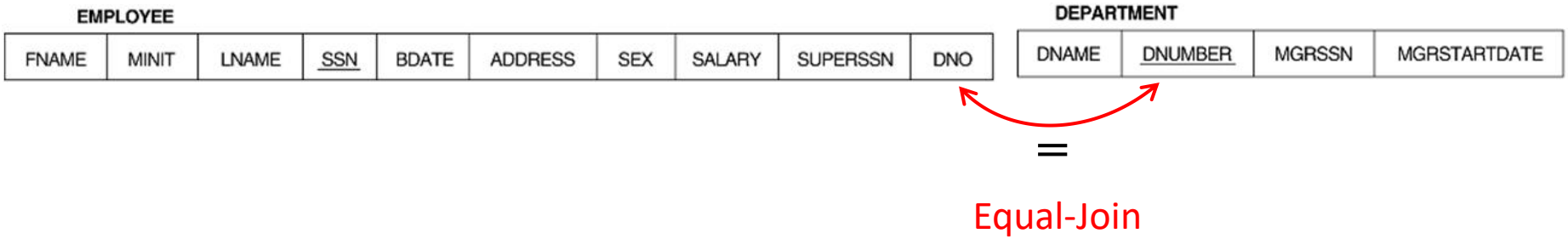


Joined Relations Feature in SQL2 (cont.)

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

could be written as:

Q1: **SELECT** FNAME, LNAME, ADDRESS
FROM (EMPLOYEE **JOIN** DEPARTMENT **ON** DNUMBER=DNO)
WHERE DNAME='Research'

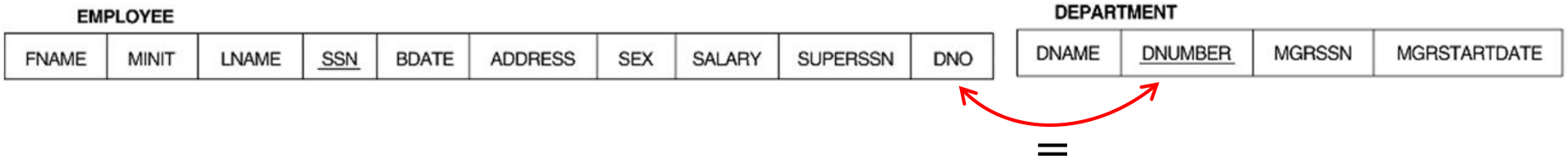


Natural Join

Q1: **SELECT** FNAME, LNAME, ADDRESS
FROM (EMPLOYEE **JOIN** DEPARTMENT **ON** DNUMBER=DNO)
WHERE DNAME='Research'

or as:

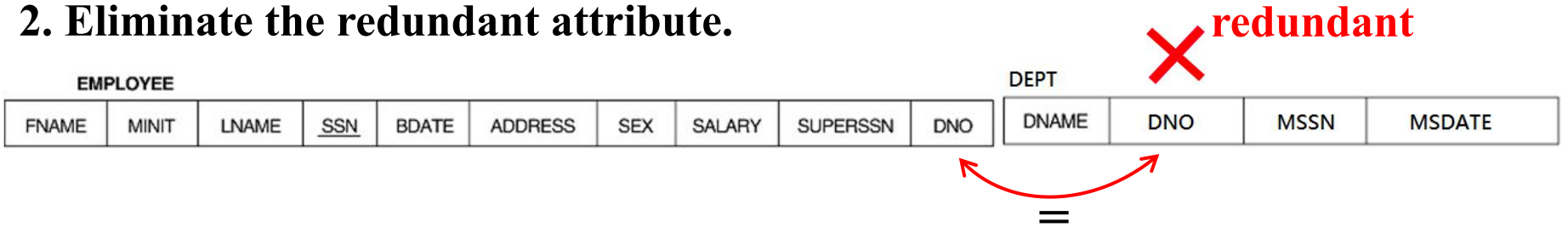
Q1: **SELECT** FNAME, LNAME, ADDRESS
FROM (EMPLOYEE **NATURAL JOIN** DEPARTMENT
AS DEPT(DNAME, DNO, MSSN, MSDATE)
WHERE DNAME='Research'



Natural Join: 問題:Left outer join 跟 natural join?

1. Equal-Join on the attributes with the same name.

2. Eliminate the redundant attribute.



Deficiency of JOIN Operation

- To find the employees who are manager and the department which they manage.

```
SELECT    FNAME, LNAME, DNAME
FROM      EMPLOYEE DEPARTMENT
WHERE     MGRSSN = SSN
```

RESULT	FNAME	LNAME	DNAME
	Franklin	Wong	Research
	Jennifer	Wallace	Administration
	James	Borg	Headquarters

- How about to include all the employees who are manager and not managers?

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

8 tuples

MGRSSN=SSN

3 tuples

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

Left Outer Join Operation

- A list of all employee names and also the name of the departments they manage *if they happen to manage a department*;
- Apply an operation **LEFT OUTER JOIN,** denoted by ⟕ , to retrieve the result as follows:

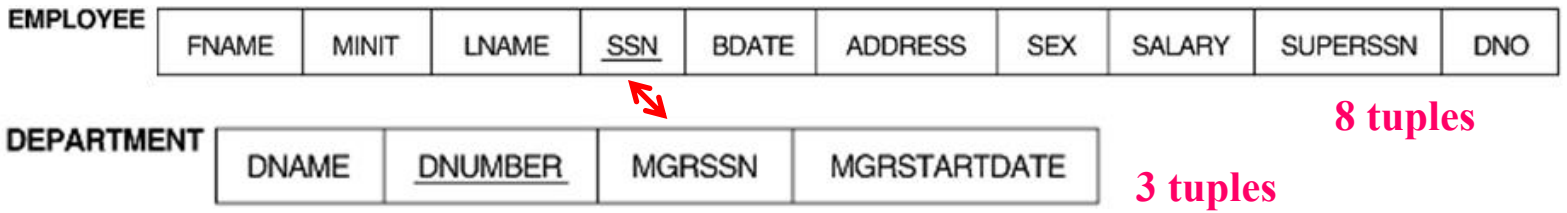
SELECT FNAME, LNAME, DNAME
FROM (EMPLOYEE **LEFT OUTER JOIN** DEPARTMENT **ON** MGRSSN=SSN)

RESULT	FNAME	LNAME	DNAME
	John	Smith	null
	Franklin	Wong	Research
	Alicia	Zelaya	null
	Jennifer	Wallace	Administration
	Ramesh	Narayan	null
	Joyce	English	null
	Ahmad	Jabbar	null
	James	Borg	Headquarters

8 tuples

3 tuples

RESULT	FNAME	LNAME	DNAME
	Franklin	Wong	Research
	Jennifer	Wallace	Administration
	James	Borg	Headquarters



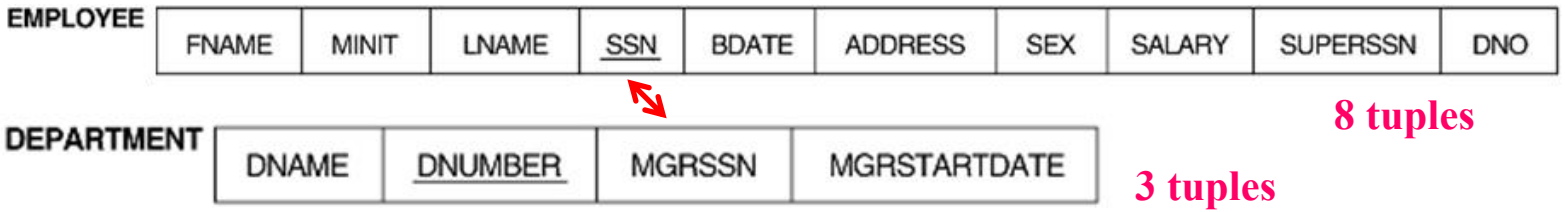
Outer Join

- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)
- Examples:

To include all the employees who are manager and not managers.

```
SELECT   FNAME, LNAME, DNAME
FROM     (EMPLOYEE LEFT OUTER JOIN DEPARTMENT ON MGRSSN=SSN)
```

```
SELECT   FNAME, LNAME, DNAME
FROM     (DEPARTMENT RIGHT OUTER JOIN EMPLOYEE ON MGRSSN=SSN)
```



AGGREGATE FUNCTIONS

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

The type of previous queries:

```
SELECT    FNAME, LNAME, ADDRESS
FROM      EMPLOYEE
WHERE     DNO = 5
```

How about the following queries:

- How many employees in the company?
- What is the average, minimum, and maximum salary in each department?
- For each department in which more than two employees work, retrieve their total salary in the department.

AGGREGATE FUNCTIONS 集合

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- Query 15:

  Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q15:  SELECT  MAX(SALARY), MIN(SALARY), AVG(SALARY)
      FROM    EMPLOYEE
```

  - Some SQL implementations *may not allow more than one function* in the SELECT-clause

- Query 16:

  Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16:  SELECT  MAX(SALARY), MIN(SALARY), AVG(SALARY)
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DNO=DNUMBER AND DNAME='Research'
```

EMPLOYEE

Maximum salary?

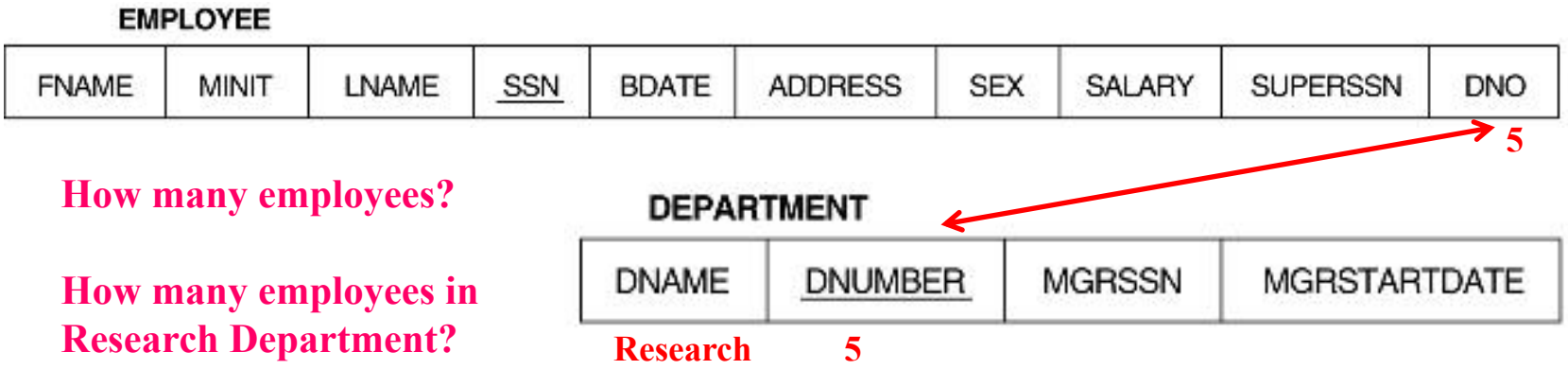
FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

AGGREGATE FUNCTIONS (cont.)

- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

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

Q18: **SELECT** **COUNT** (*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER **AND** DNAME='Research'



GROUPING

- In many cases, we want to apply the aggregate functions *to **subgroups** of tuples in a relation*
- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

How many employees in each department?

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

GROUPING (cont.)

- 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

  - the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO
  - The COUNT and AVG functions are applied to each such group of tuples separately
  - The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples

FNAME	MINIT	LNAME	SSN	• • •	SALARY	SUPERSSN	DNO
John	B	Smith	123456789		30000	333445555	5
Franklin		Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	• • •	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	null	1

Grouping EMPLOYEE tuples by the value of DNO.

DNO	COUNT (*)	AVG (SALARY)
5	4	33250
4	3	31000
1	1	55000

Result of Q24.

GROUPING (cont.)

- Query 21: For **each project**, retrieve the project number, project name, and **the number of employees** who work on that project.

```
Q21:  SELECT     PNUMBER, PNAME, COUNT(*)
      FROM       PROJECT, WORKS_ON
      WHERE      PNUMBER=PNO
      GROUP BY   PNUMBER, PNAME
```

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

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0

PNAME	PNUMBER		ESSN	PNO	HOURS	
ProductX	1		123456789	1	32.5	2
ProductX	1		453453453	1	20.0	
ProductY	2		123456789	2	7.5	3
ProductY	2		453453453	2	20.0	
ProductY	2		333445555	2	10.0	
ProductZ	3	. . .	666884444	3	40.0	

THE HAVING-CLAUSE

- To retrieve the values of these functions for only those *groups that satisfy certain conditions*
- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

Q26: For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

```
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

2 3 2

PNUMBER = PNO
GROUP BY PNUMBER, PNAME

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

After applying the WHERE clause but before applying HAVING.

FIGURE 8.6 Results of GROUP BY and HAVING. (b) 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.

COUNT (*) > 2

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

Result of Q26
(PNUMBER not shown).

Q26:

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

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

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

- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause.

Summary of SQL Queries (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query

Constraints as Assertions

- General constraints:
  - constraints that do not fit in the basic SQL categories

    e.g. key constraint, entity constraint, referential integrity constraint.
- Mechanism:

  CREAT ASSERTION
  - components include: a constraint name, followed by **CHECK**, followed by a condition

The salary of an employee must not be greater than the salary of the manager of the department that the employee works for.

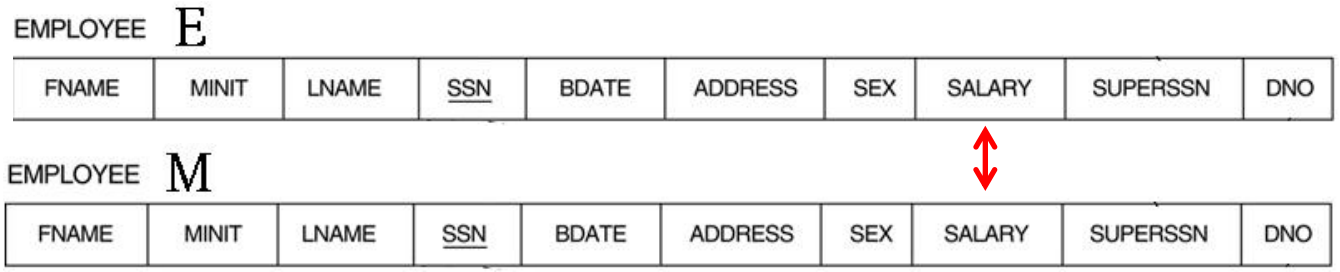
Assertions: An Example

- Constraint:

  The salary of an employee must not be greater than the salary of the manager of the department that the employee works for.

```
CREATE   ASSERTION SALARY_CONSTRAINT
CHECK    (NOT EXISTS (SELECT *
                      FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
                      WHERE E.SALARY > M.SALARY AND
                       E.DNO=D.NUMBER AND D.MGRSSN=M.SSN))
```

- Specify a query that violates the condition; include inside a **NOT EXISTS** clause
- Query result must be empty
  - if the query result is not empty, the assertion has been violated



SQL Triggers

- **To monitor a database and take action when a condition occurs**
- Triggers are expressed in a syntax similar to assertions and include the following:
  - event (e.g., an update operation)
  - condition
  - action (to be taken when the condition is satisfied)
- A trigger **to compare an employee's salary to his/her supervisor during insert or update operations:**

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
        SALARY, SUPERSSN ON EMPLOYEE
FOR EACH ROW
WHEN
        (NEW.SALARY > (SELECT SALARY
                       FROM EMPLOYEE
                       WHERE SSN = NEW.SUPERSSN))
INFORM_SUPERVISOR (NEW.SUPERSSN,NEW.SSN);
```

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO	
			886				30000	997		(original)
			886			update →	40000	997		(NEW)
			997				38000	123		

Views in SQL

- A view is a "virtual" table that is derived from other tables (base tables or other virtual tables)
- Allows full query operations
- Allows for limited update operations (since the table may not physically be stored)
- A convenience for expressing certain operations

Two views specified on the database schema

WORKS_ON1

FNAME	LNAME	PNAME	HOURS

DEPT_INFO

DEPT_NAME	NO_OF_EMPS	TOTAL_SAL

WORKS_ON

ESSN	PNO	HOURS

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

Base tables

Names of employees working on the project ProductX?

SQL Views: An Example

- Specify a different WORKS_ON table

```
CREATE VIEW    WORKS_ON1
AS   SELECT    FNAME, LNAME, PNAME, HOURS
     FROM      EMPLOYEE, PROJECT, WORKS_ON
     WHERE     SSN=ESSN AND PNO=PNUMBER;
```

- We can specify SQL queries on a newly create table (view):

```
SELECT    FNAME, LNAME
FROM      WORKS_ON1
WHERE     PNAME='ProjectX';
```

WORKS_ON1

FNAME	LNAME	PNAME	HOURS

- When no longer needed, a view can be dropped:

```
DROP      WORKS_ON1;
```

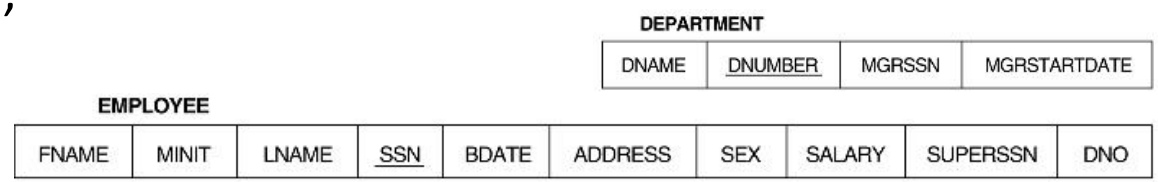
Specification of Views

- SQL command: **CREATE VIEW**
  - a table (view) name
  - a possible list of attribute names, e.g.,

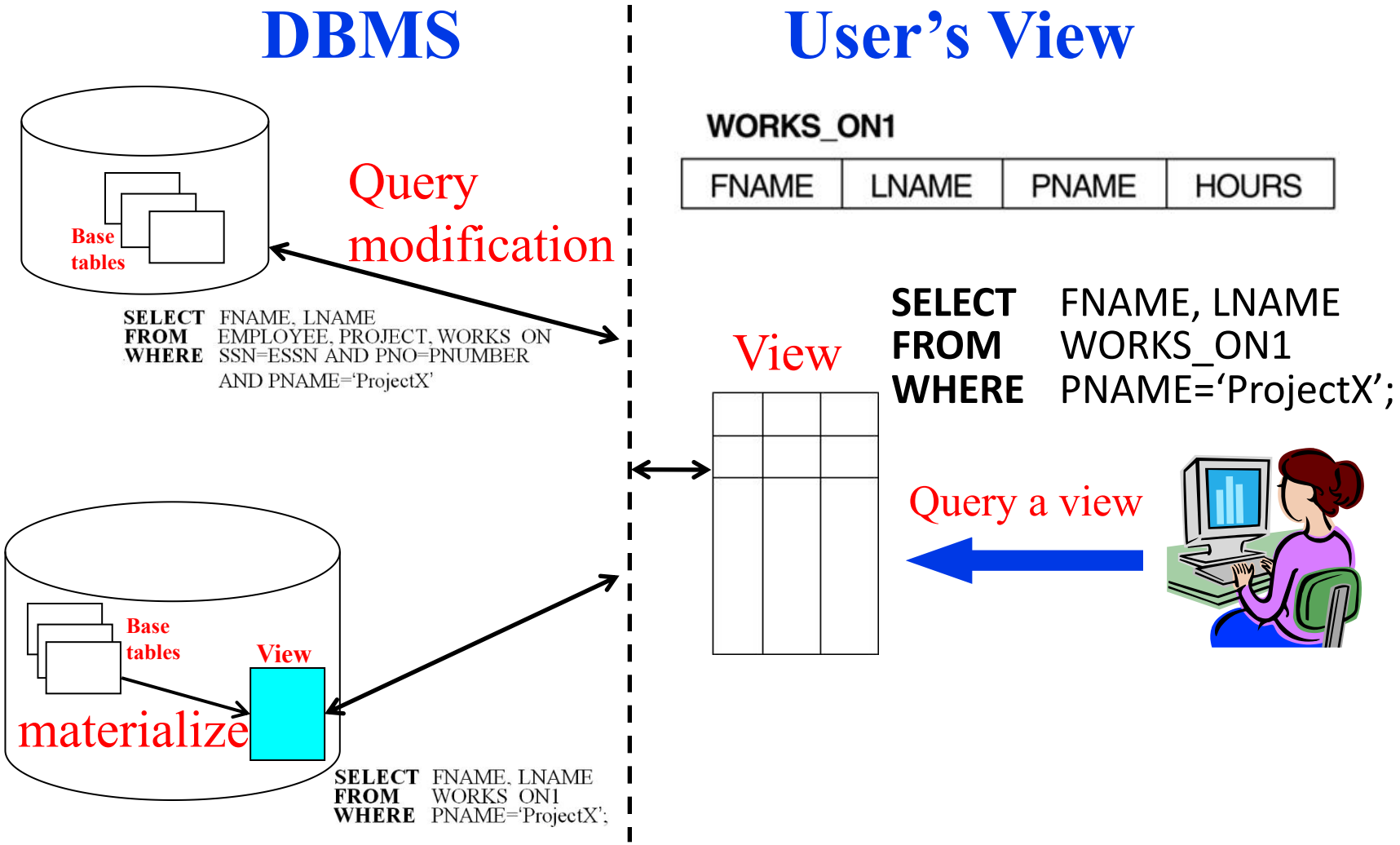
    (when arithmetic operations are specified or

    when we want the names to be different from the attributes in the base relations)
  - a query to specify the table contents

```
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;
```



Two View Implementations



Query Modification

```
SELECT    FNAME, LNAME
FROM      WORKS_ON1                          // view
WHERE     PNAME='ProjectX';
```

Query modification
to base tables

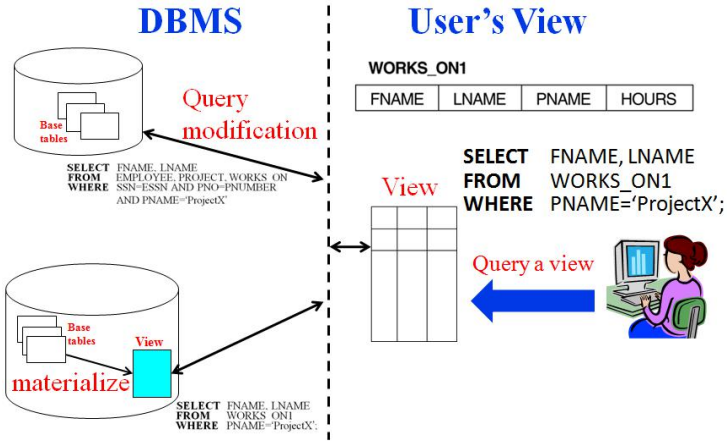
```
SELECT    FNAME, LNAME
FROM      EMPLOYEE, PROJECT, WORKS_ON        //base tables
WHERE     SSN=ESSN AND PNO=PNUMBER  AND
          PNAME='ProjectX'
```

Efficient View Implementation

- **Query modification:**

  present the view query in terms of a query on the underlying base tables
  - disadvantage: inefficient for views defined via complex queries (especially if additional queries are to be applied to the view within a short time period)
- **View materialization:**

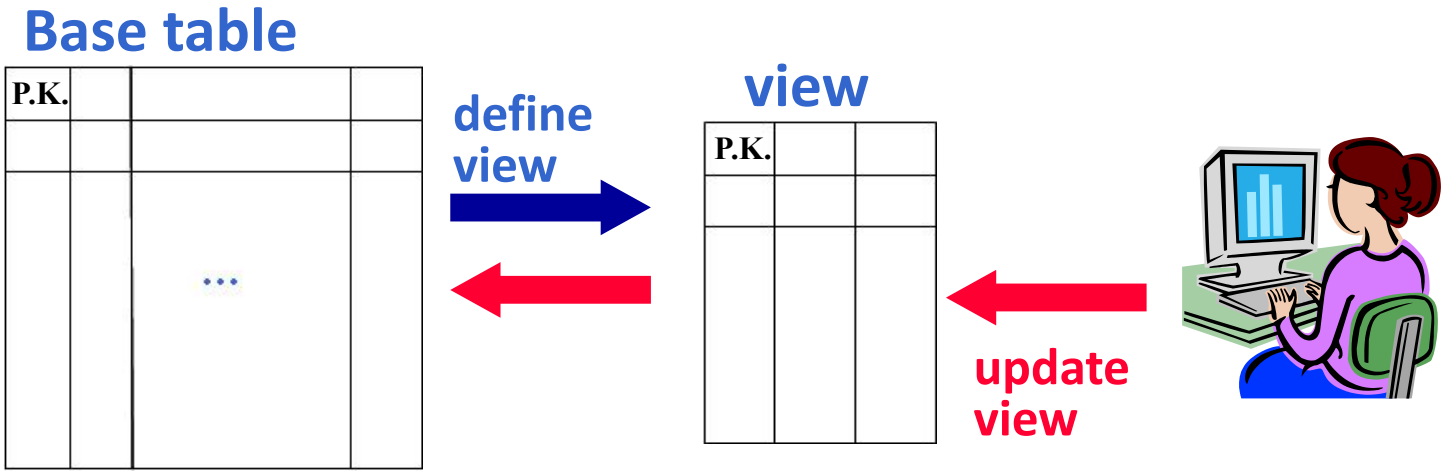
  involves physically creating and keeping a temporary table
  - assumption: other queries on the view will follow
  - concerns: maintaining correspondence between the base table and the view when the base table is updated
  - strategy: incremental update



View Update

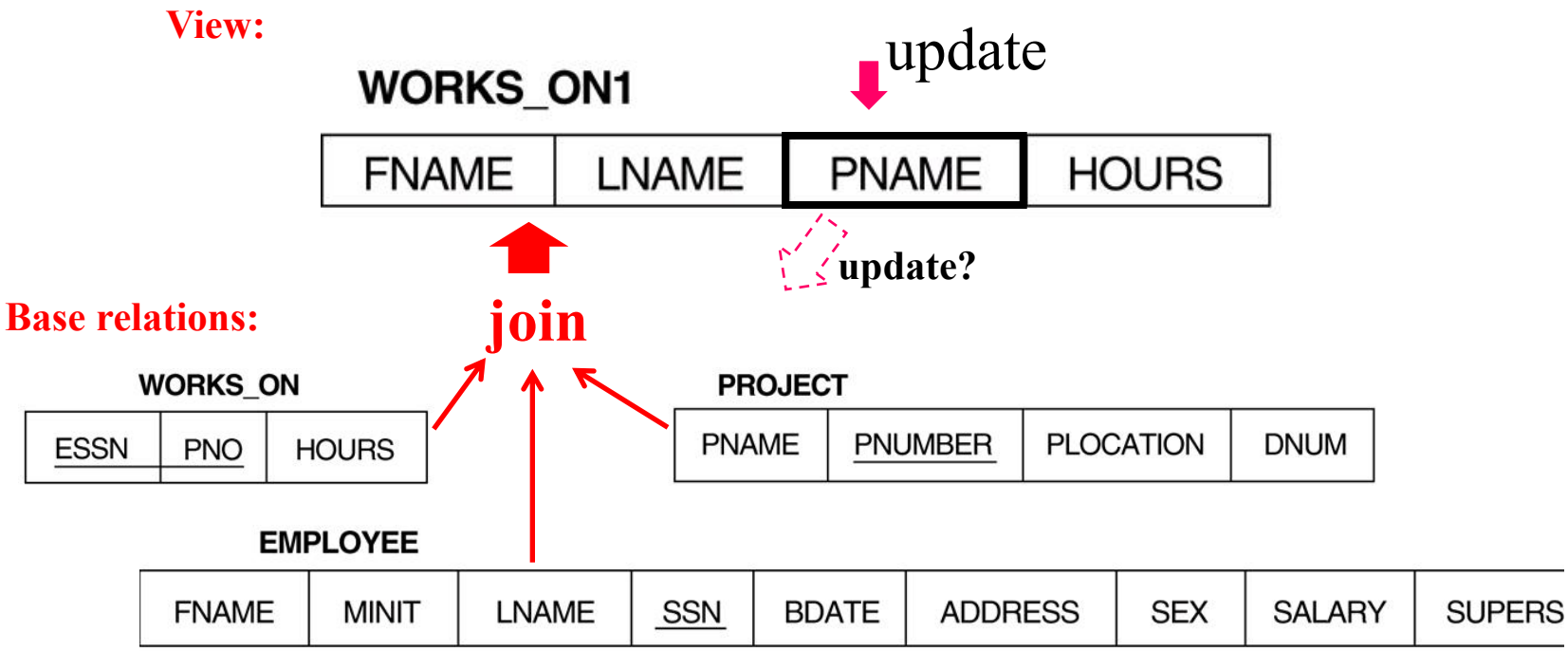
- Update on a single view is possible if the view attributes contain
  - the Primary Key of the base relation, and
  - as all attributes with the NOT NULL constraint that do not have default values specified.
- **WITH CHECK OPTION:**

  must be added to the definition of a view if the view is to be updated
  - to allow check for updatability and to plan for an execution strategy



Un-updatable Views

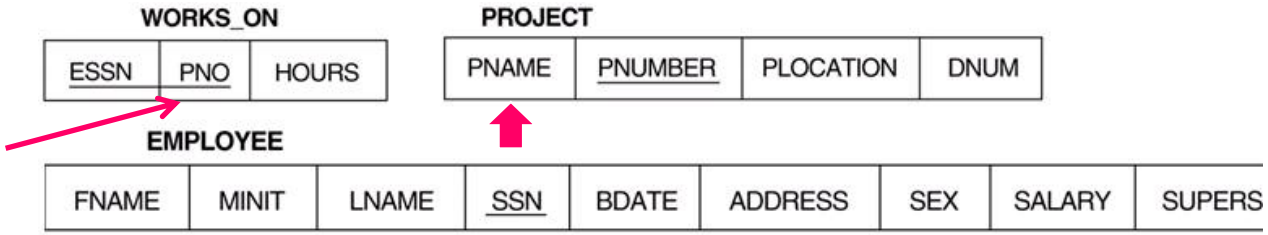
UPDATE WORKS_ON1
SET PNAME='ProductY'
WHERE LNAME='Smith' **AND** FNAME='John' **AND** PNAME='ProductX';



Two Possible Modifications

```
UPDATE    WORKS_ON
SET       PNO =    (SELECT    PNUMBER
                   FROM       PROJECT
                   WHERE      PNAME='ProjectY')
WHERE     ESSN IN  (SELECT    SSN
                   FROM       EMPLOYEE
                   WHERE      LNAME='Smith' AND FNAME='John')
          AND
          PNO =    (SELECT    PNUMBER
                   FROM       PROJECT
                   WHERE      PNAME='ProductX');
```

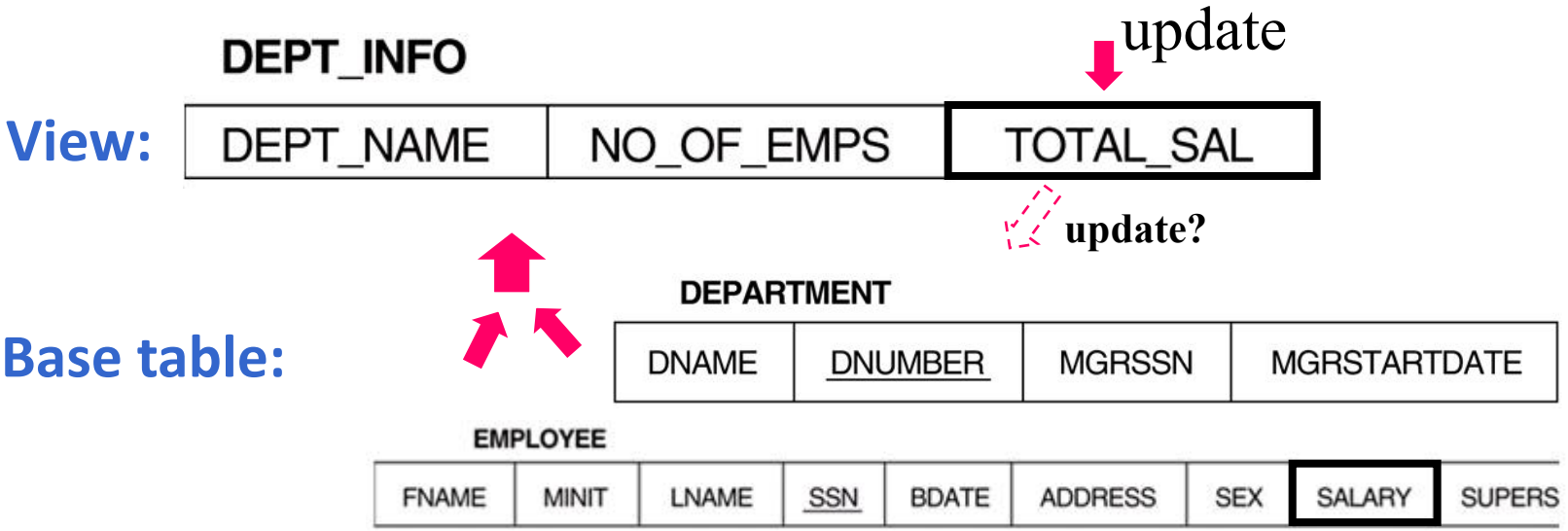
```
UPDATE    PROJECT SET PNAME='ProductY'
WHERE     PNAME='ProductX';
```



Un-updatable Views

- Views defined using groups and aggregate functions are not updateable

```
UPDATE      DEPT_INFO
SET         TOTAL_SAL=100000
WHERE       DNAME='Research'
```



DROP TABLE

- Used to remove a relation (base table) *and its definition*
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

DROP TABLE DEPENDENT**;**

ALTER TABLE

- Used to add an attribute to one of the base relations
- The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is *not allowed* for such an attribute
- Example:

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

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

關於考試

- 會有英文題目
- 會考考古題及網路學園內的自我評量
- 會考SQL
- 不能帶任何字典
- 不能使用鉛筆作答
- 手機關機且不可放在桌面