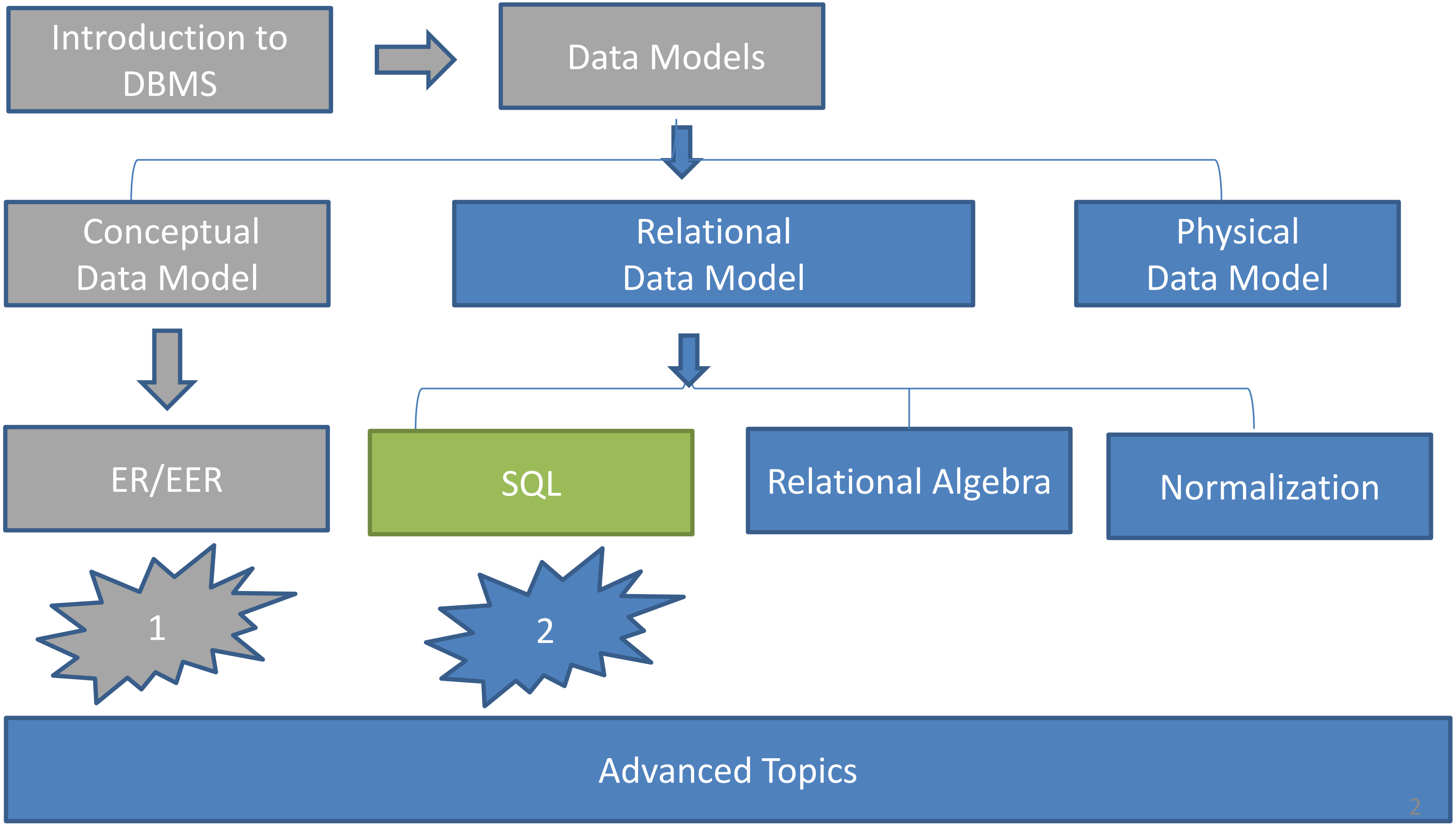
An Introduction to the Database Management Systems

By
Hossein Rahmani

Slides originally by Book(s) Resources

Road Map

(Might change!)



Introduction to DBMS

Data Models

Conceptual Data Model

Relational Data Model

Physical Data Model

ER/EER

SQL

Relational Algebra

Normalization

1

2

Advanced Topics

SQL Advanced

- More Complex SQL Retrieval Queries
- 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**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- 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.)

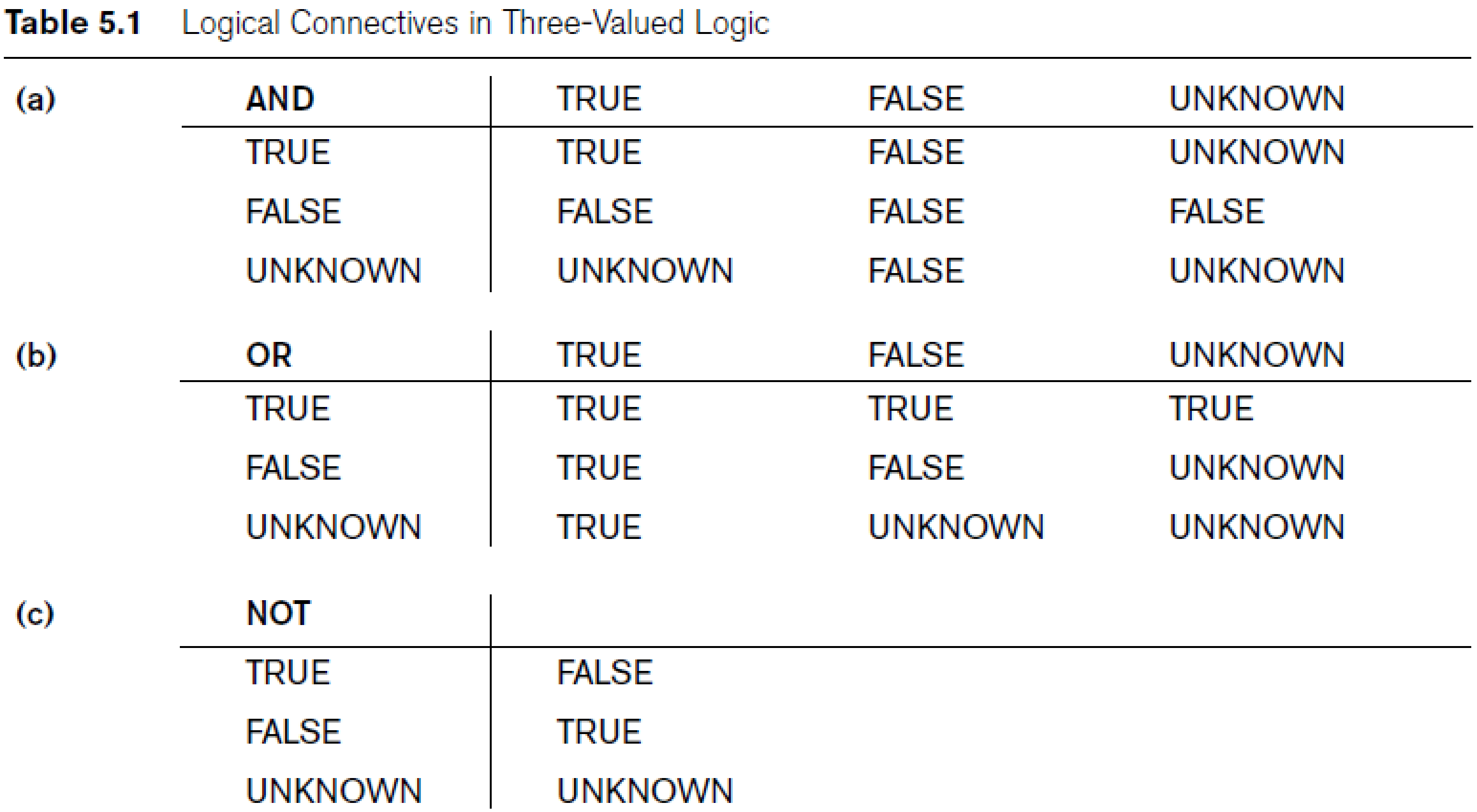


Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

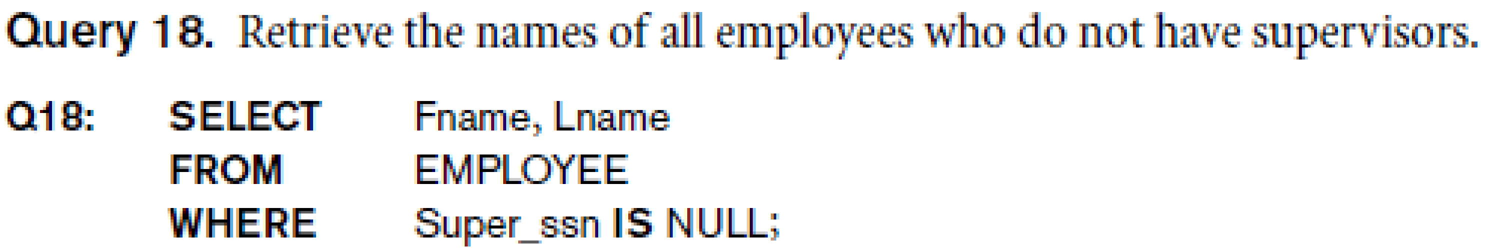
OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(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`
  - `IS` or `IS NOT NULL`



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

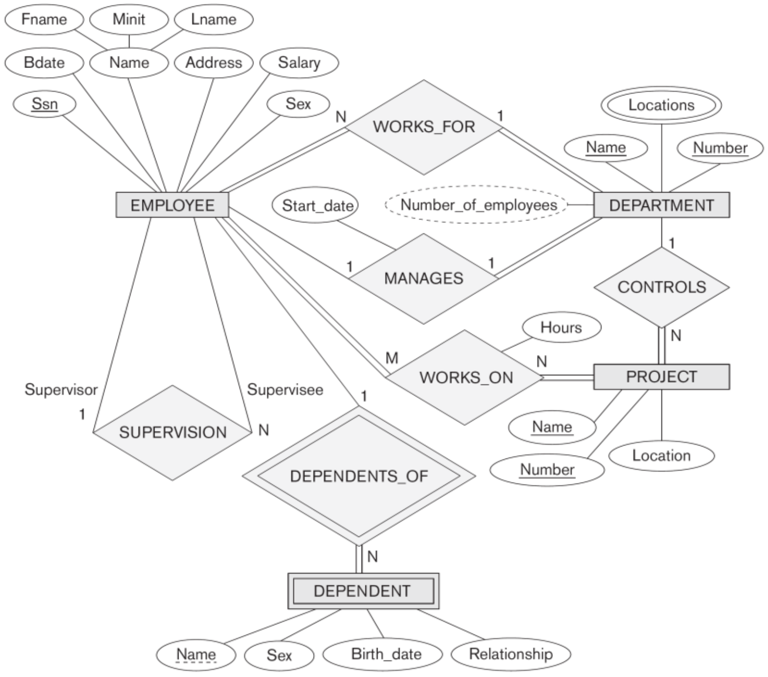
```
Q18:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Super_ssn IS NULL;
```

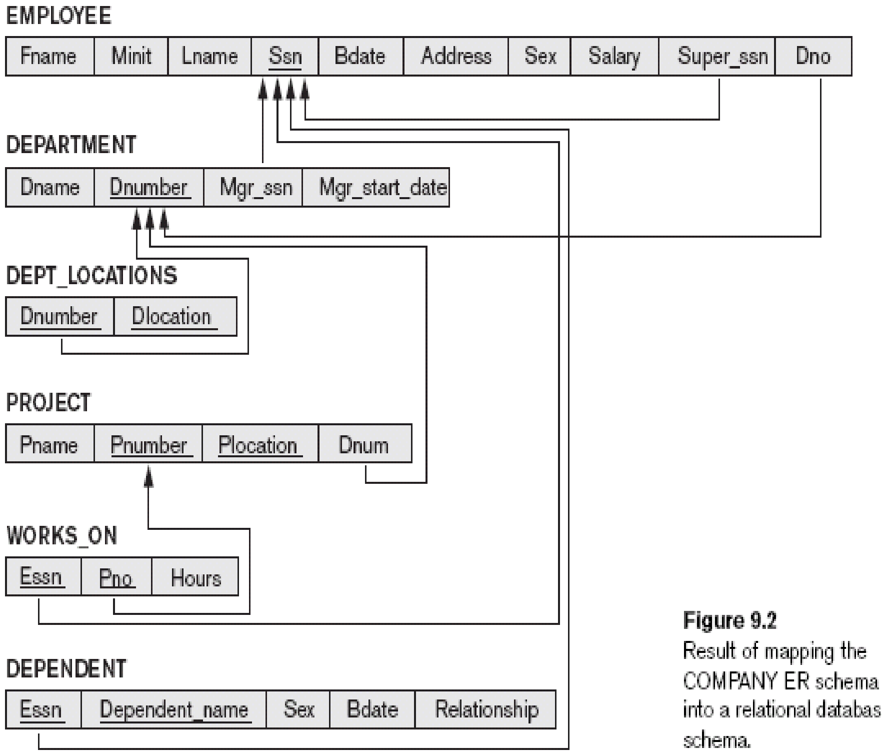
Nested Queries, Tuples, and Set/Multiset Comparisons

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

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

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.





EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

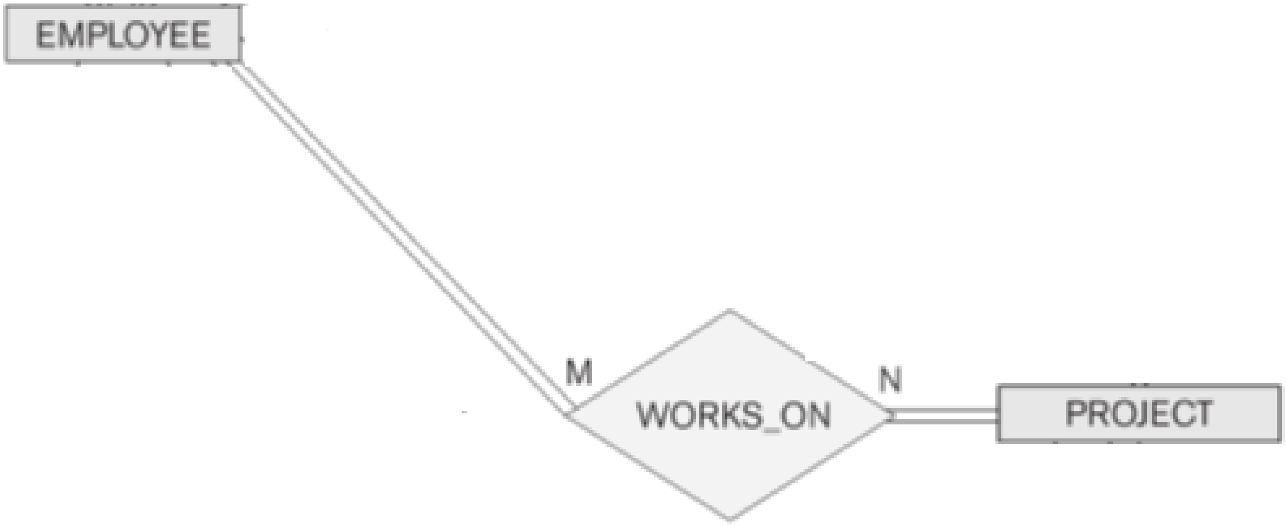
WORKS_ON

Essn	Pno	Hours

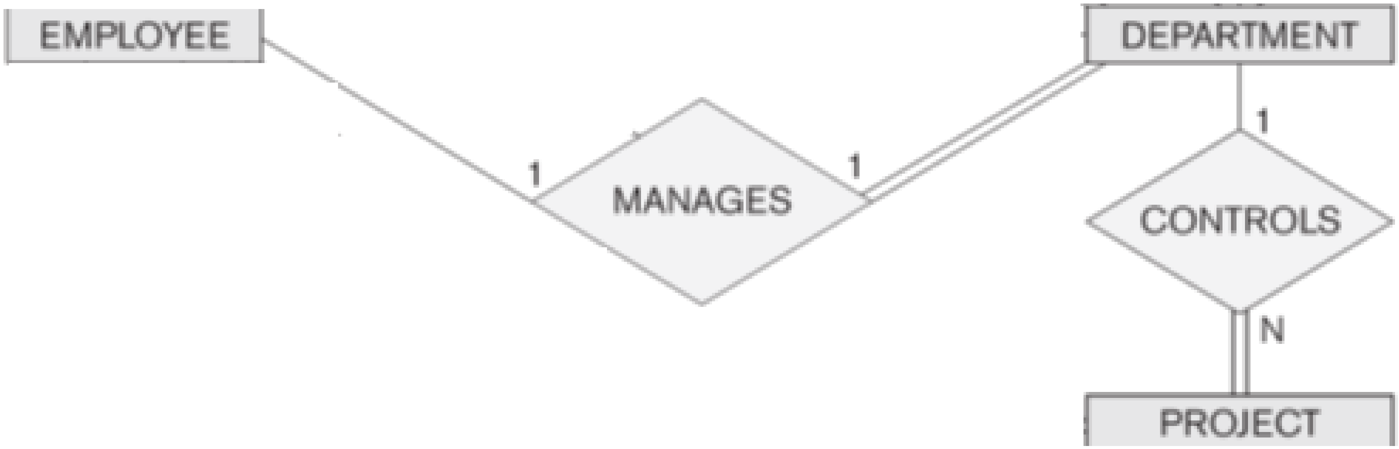
DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

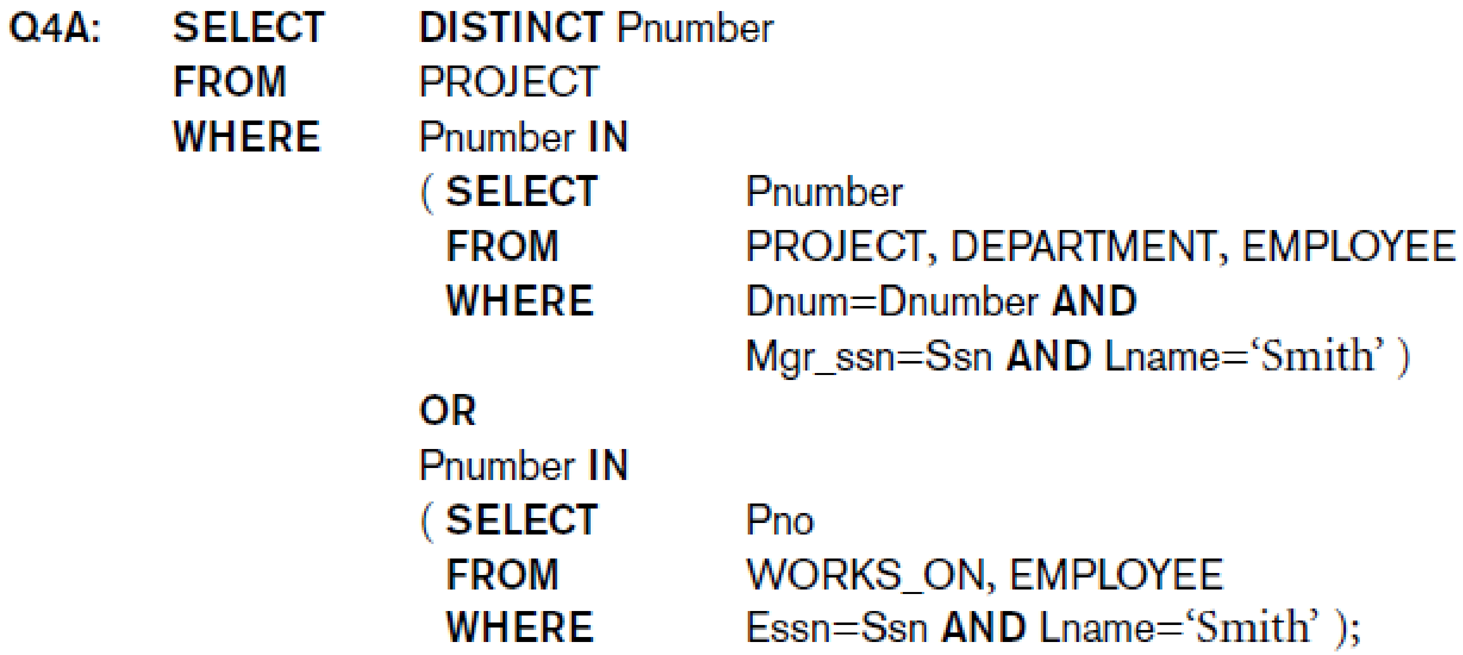
Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.


EMPLOYEE
M
WORKS_ON
N
PROJECT

OR


EMPLOYEE
DEPARTMENT
1
MANAGES
1
1
CONTROLS
N
PROJECT

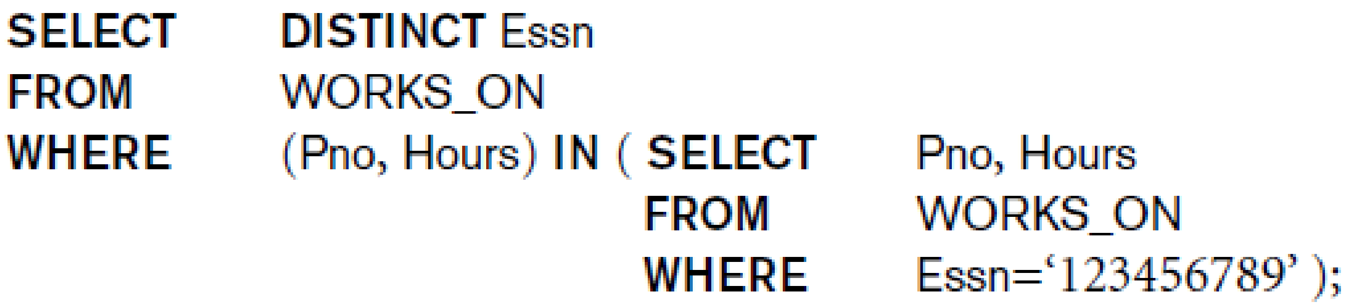
Nested Queries (cont’d.)



```
Q4A:   SELECT     DISTINCT Pnumber
       FROM       PROJECT
       WHERE      Pnumber IN
                  ( SELECT      Pnumber
                    FROM        PROJECT, DEPARTMENT, EMPLOYEE
                    WHERE       Dnum=Dnumber AND
                                Mgr_ssn=Ssn AND Lname=‘Smith’ )
                  OR
                  Pnumber IN
                  ( SELECT      Pno
                    FROM        WORKS_ON, EMPLOYEE
                    WHERE       Essn=Ssn AND Lname=‘Smith’ );
```

Nested Queries (cont'd.)

- Use tuples of values in comparisons
  - Place them within parentheses

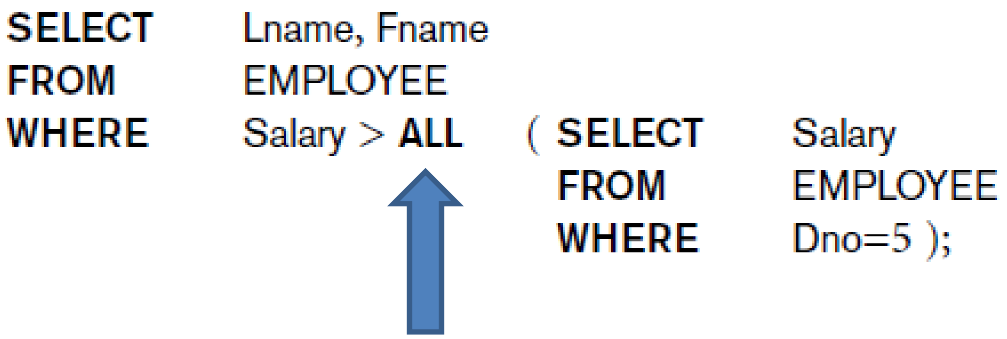


```
SELECT      DISTINCT Essn
FROM        WORKS_ON
WHERE       (Pno, Hours) IN ( SELECT      Pno, Hours
                              FROM        WORKS_ON
                              WHERE       Essn='123456789' );
```

This query will select the Essns of all employees who their work pattern (project, hours) is similar to employee 'John Smith' (whose Ssn ='123456789')

Nested Queries (cont'd.)

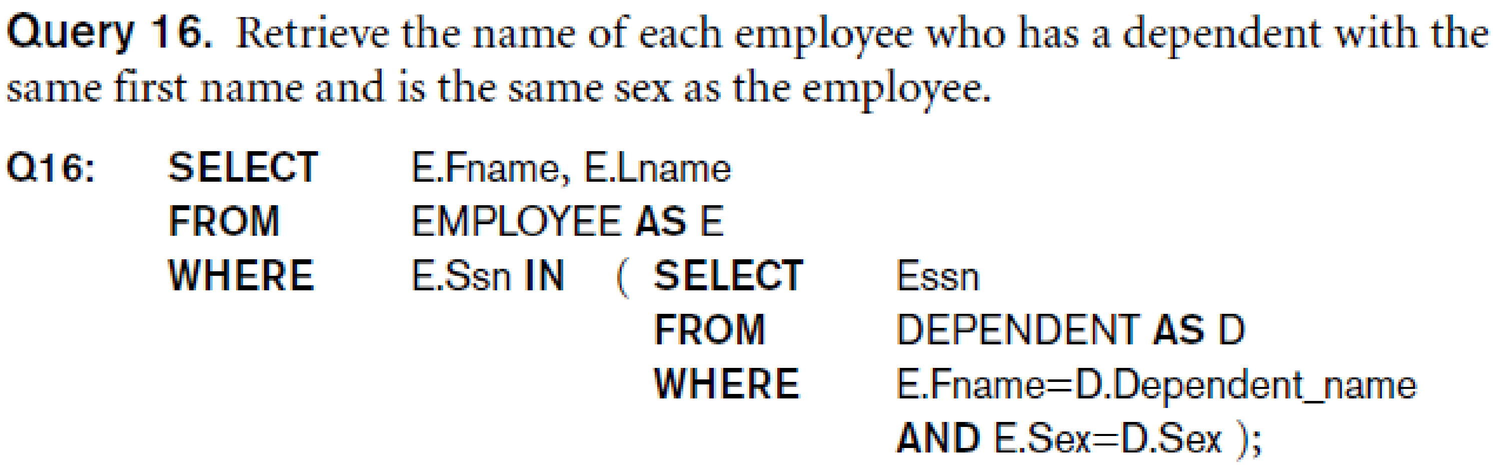
- Use other comparison operators to compare a single value *v*
  - = ANY (or = SOME) operator
    - Returns TRUE if the value *v* is equal to some value in the set *V* and is hence equivalent to IN
  - Other operators that can be combined with ANY (or SOME): >, >=, <, <=, and <>



```
SELECT   Lname, Fname
FROM     EMPLOYEE
WHERE    Salary > ALL   ( SELECT   Salary
                          FROM     EMPLOYEE
                          WHERE    Dno=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.

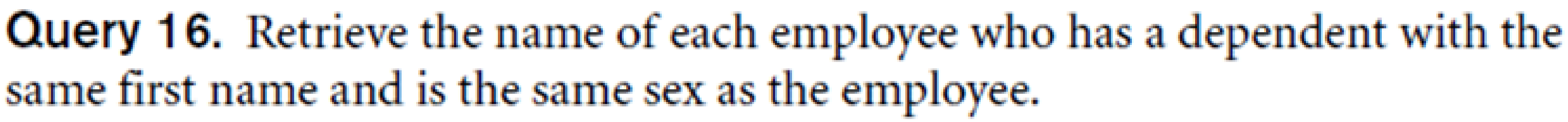
```
Q16:    SELECT    E.Fname, E.Lname
        FROM      EMPLOYEE AS E
        WHERE     E.Ssn IN  ( SELECT    Essn
                              FROM      DEPENDENT AS D
                              WHERE     E.Fname=D.Dependent_name
                                        AND E.Sex=D.Sex );
```

Correlated Nested Queries

- Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be **correlated**.
- **Correlated** nested query
  - Evaluated once for each tuple in the outer query

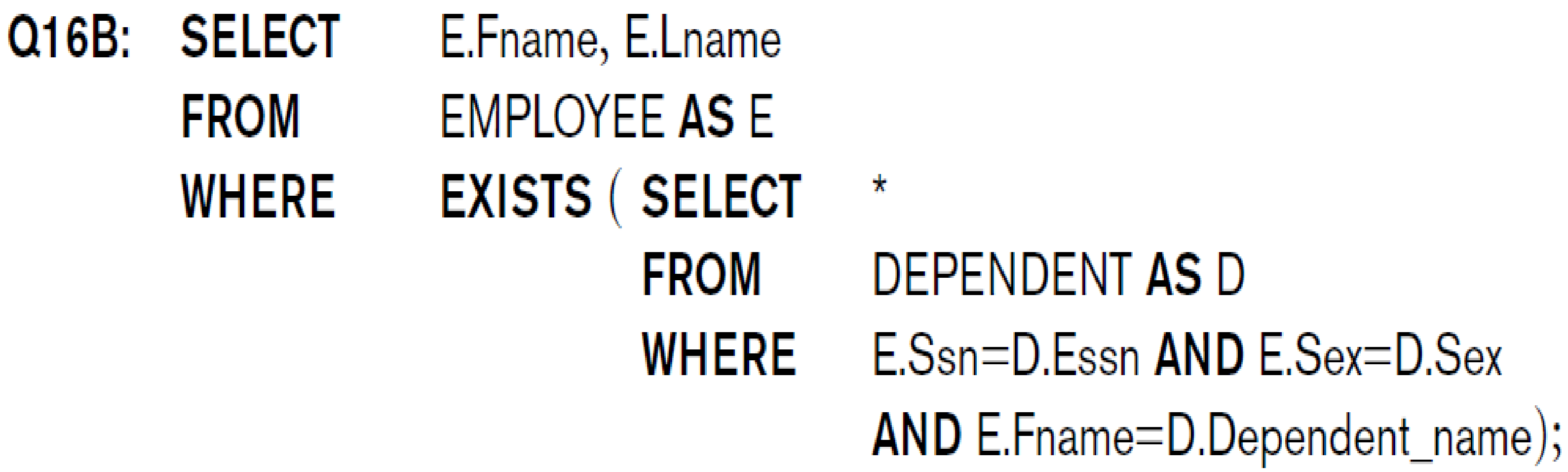
The EXISTS and UNIQUE Functions in SQL

- `EXISTS` function
  - Check whether the result of a correlated nested query is empty or not
- `EXISTS` and `NOT EXISTS`
  - Typically used in conjunction with a correlated nested query
- SQL function `UNIQUE(Q)`
  - Returns `TRUE` if there are no duplicate tuples in the result of query Q



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.

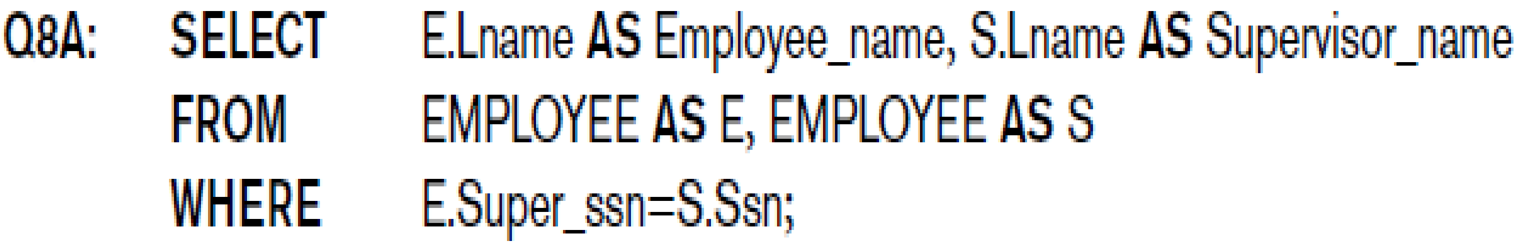
Formulate Query 16 in an alternative form that uses EXISTS as in Q16B:



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

Explicit Sets and Renaming of Attributes in SQL

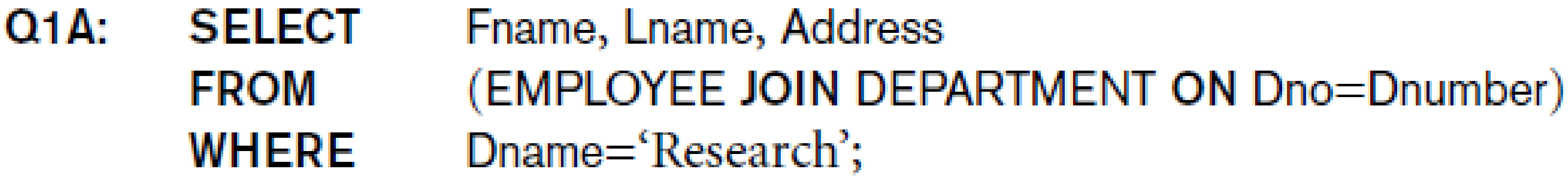
- Can use explicit set of values in WHERE clause
- Use qualifier AS followed by desired new name
  - Rename any attribute that appears in the result of a query



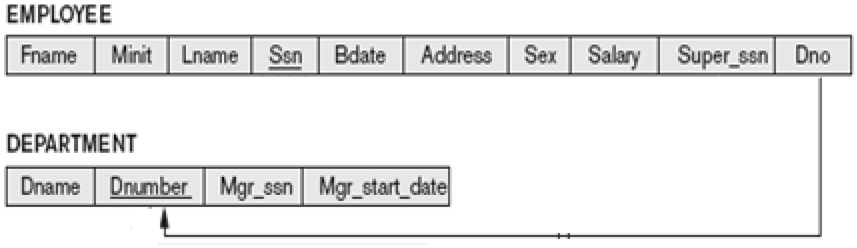
```
Q8A:    SELECT    E.Lname AS Employee_name, S.Lname AS Supervisor_name
        FROM      EMPLOYEE AS E, EMPLOYEE AS S
        WHERE     E.Super_ssn=S.Ssn;
```

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';
```



EMPLOYEE

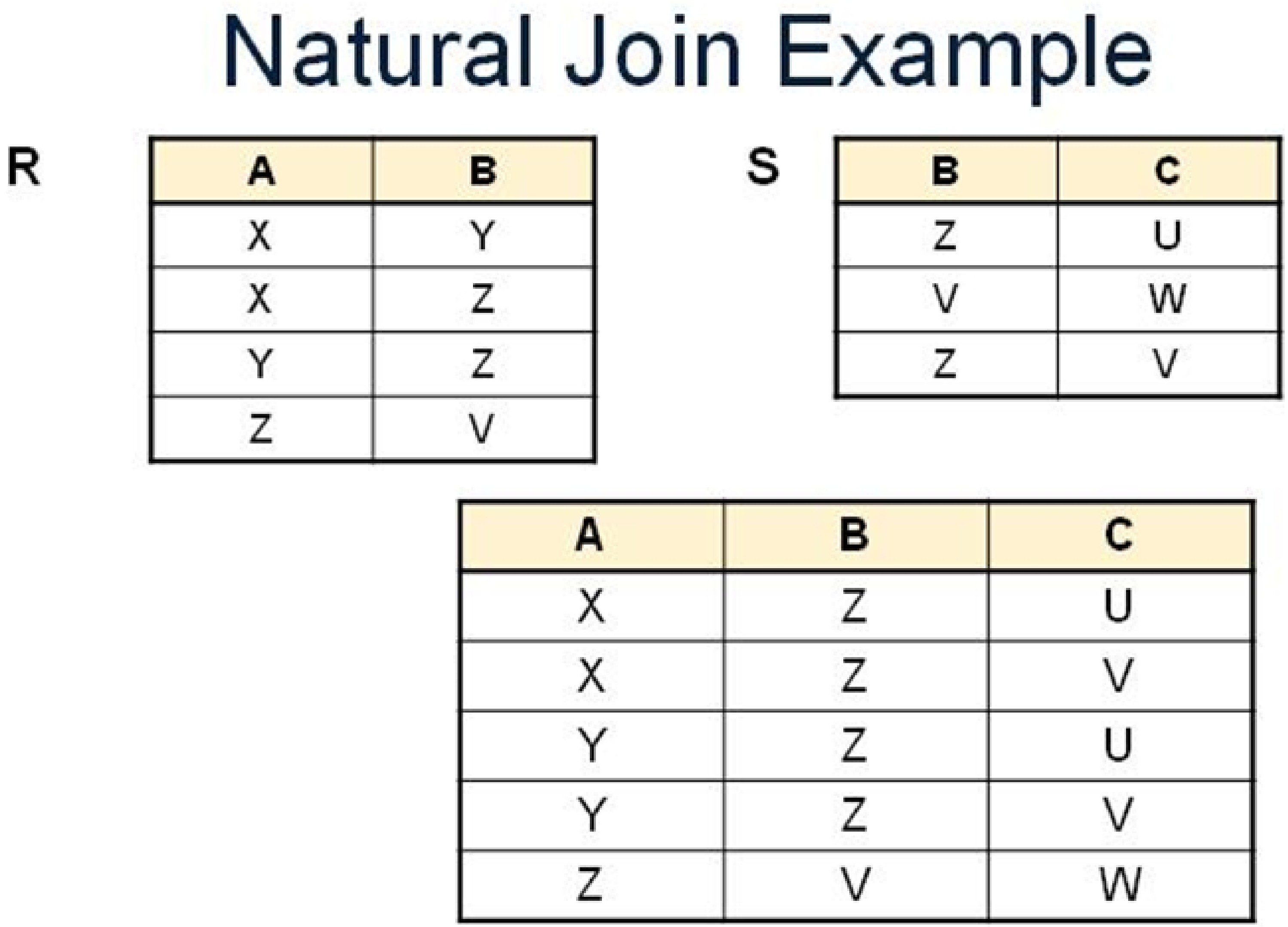
Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

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
  - No join condition specified
  - Implicit EQUIJOIN condition for each pair of attributes with same name from R and S



Natural Join Example

R

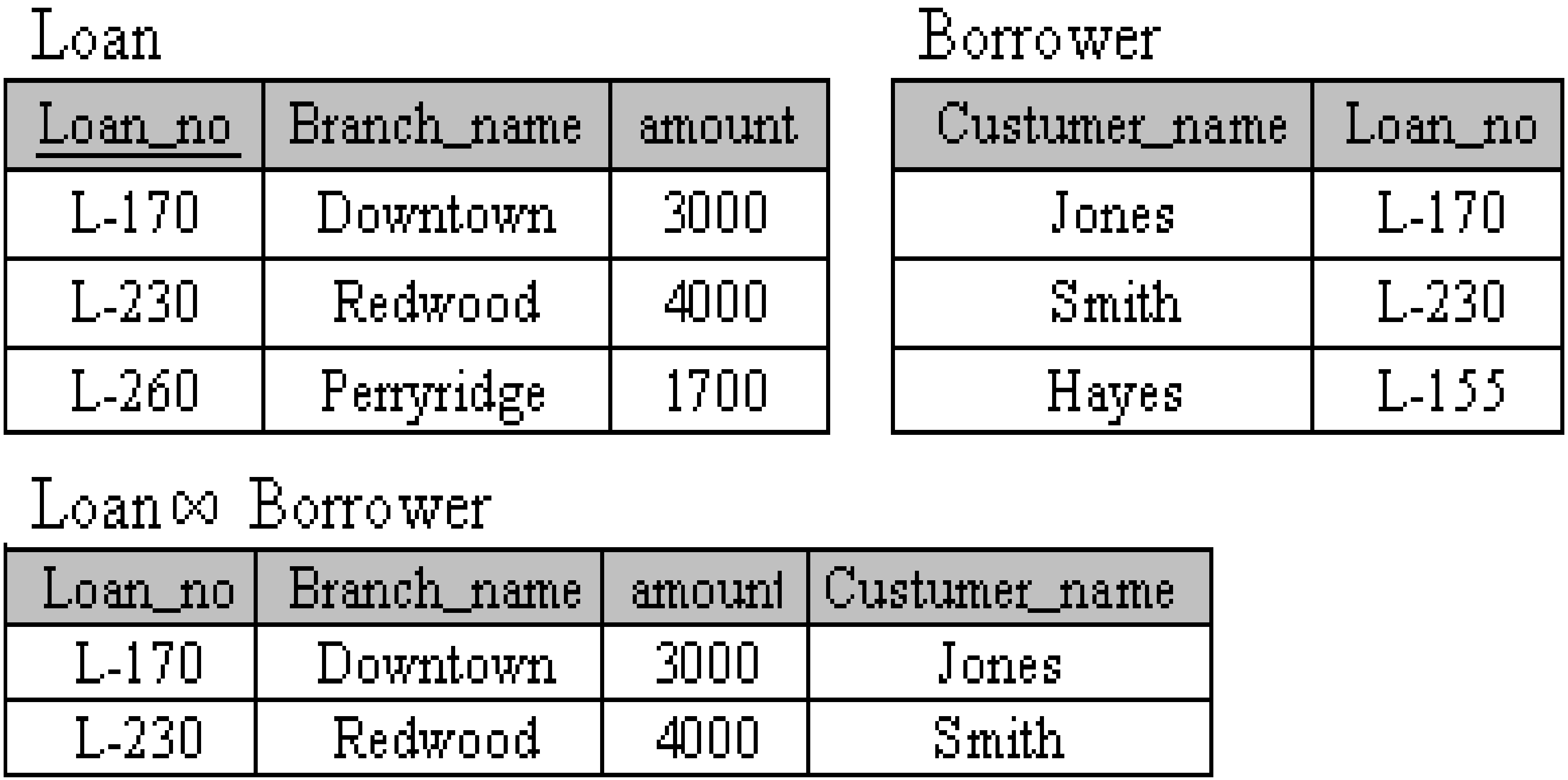
A	B
X	Y
X	Z
Y	Z
Z	V

S

B	C
Z	U
V	W
Z	V

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

Natural Join



Loan

<u>Loan_no</u>	Branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Borrower

Custumer_name	Loan_no
Jones	L-170
Smith	L-230
Hayes	L-155

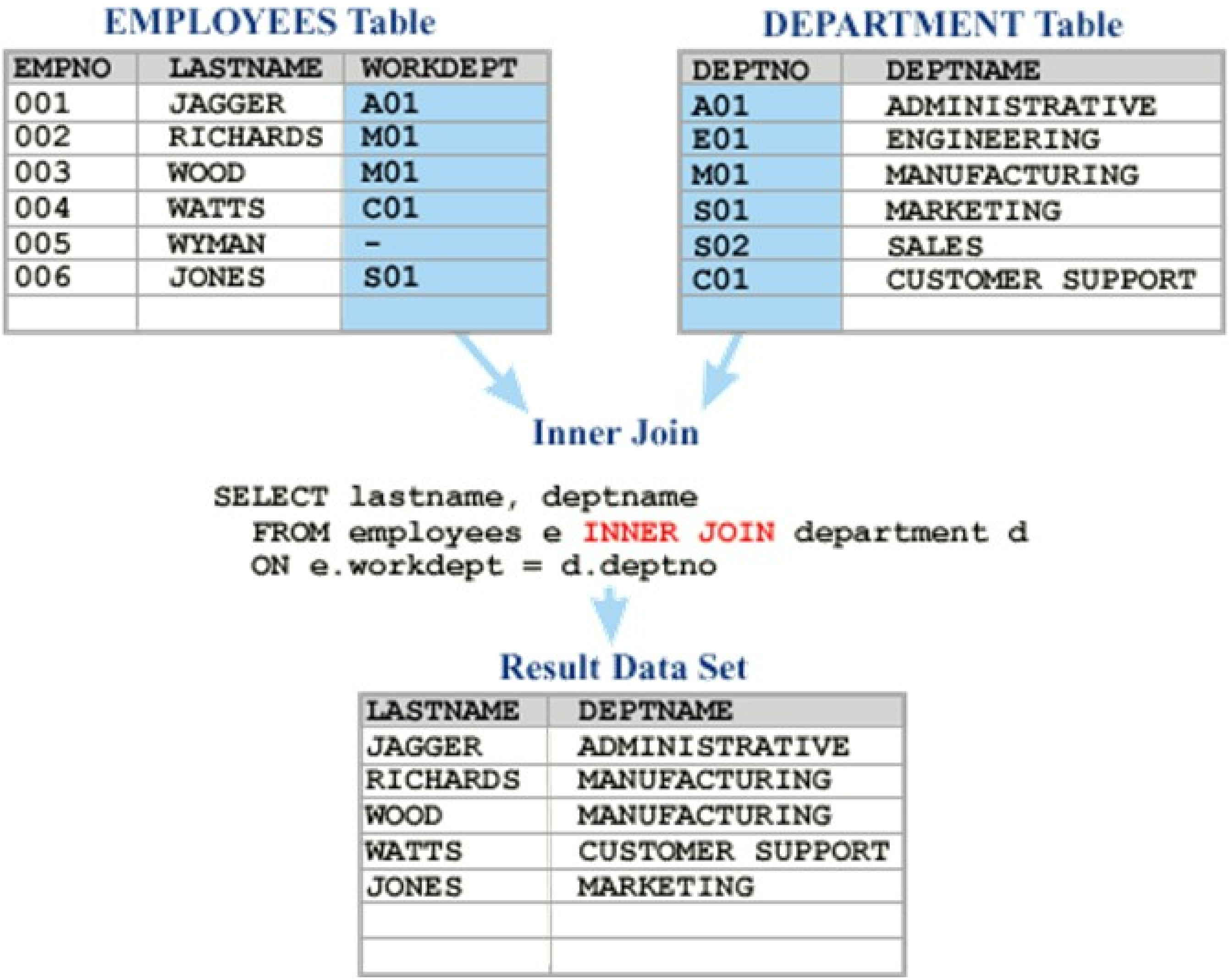
Loan ∞ Borrower

Loan_no	Branch_name	amount	Custumer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

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

- **Inner join**
  - Most frequently used type of join
  - Referred to as an EQUIJOIN.
  - query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate.

Inner Join Example



EMPLOYEES Table

EMPNO	LASTNAME	WORKDEPT
001	JAGGER	A01
002	RICHARDS	M01
003	WOOD	M01
004	WATTS	C01
005	WYMAN	-
006	JONES	S01

DEPARTMENT Table

DEPTNO	DEPTNAME
A01	ADMINISTRATIVE
E01	ENGINEERING
M01	MANUFACTURING
S01	MARKETING
S02	SALES
C01	CUSTOMER SUPPORT

Inner Join

```
SELECT lastname, deptname
  FROM employees e INNER JOIN department d
  ON e.workdept = d.deptno
```

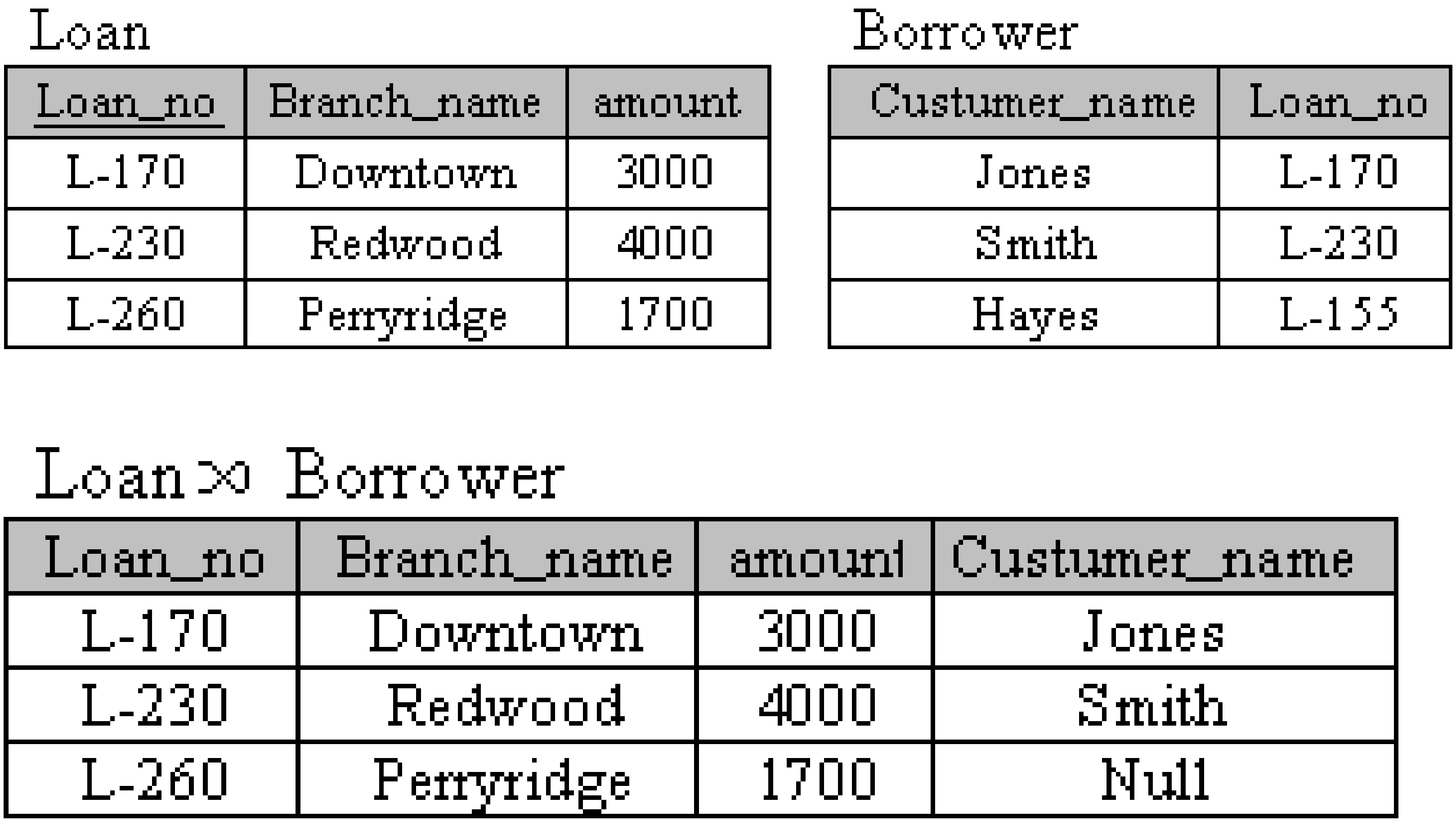
Result Data Set

LASTNAME	DEPTNAME
JAGGER	ADMINISTRATIVE
RICHARDS	MANUFACTURING
WOOD	MANUFACTURING
WATTS	CUSTOMER SUPPORT
JONES	MARKETING

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

- **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

Left Outer join



Loan

Loan_no	Branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

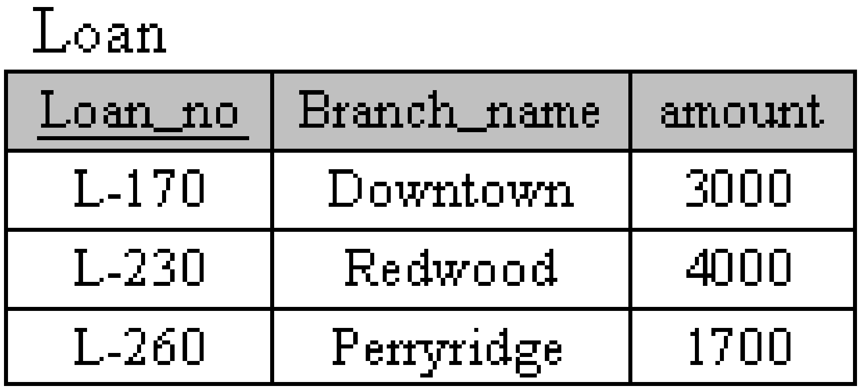
Borrower

Custumer_name	Loan_no
Jones	L-170
Smith	L-230
Hayes	L-155

Loan ⟕ Borrower

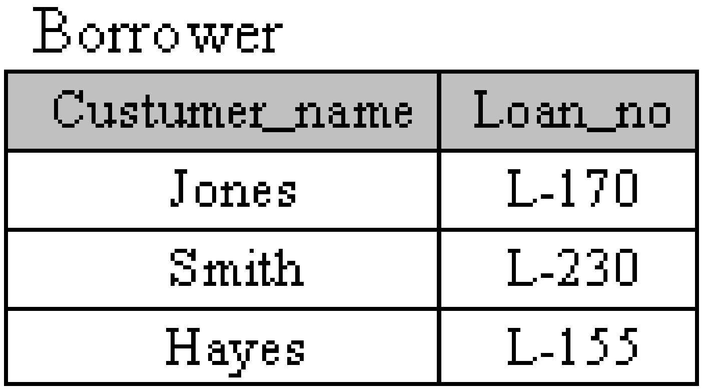
Loan_no	Branch_name	amount	Custumer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	Null

Right Outer Join



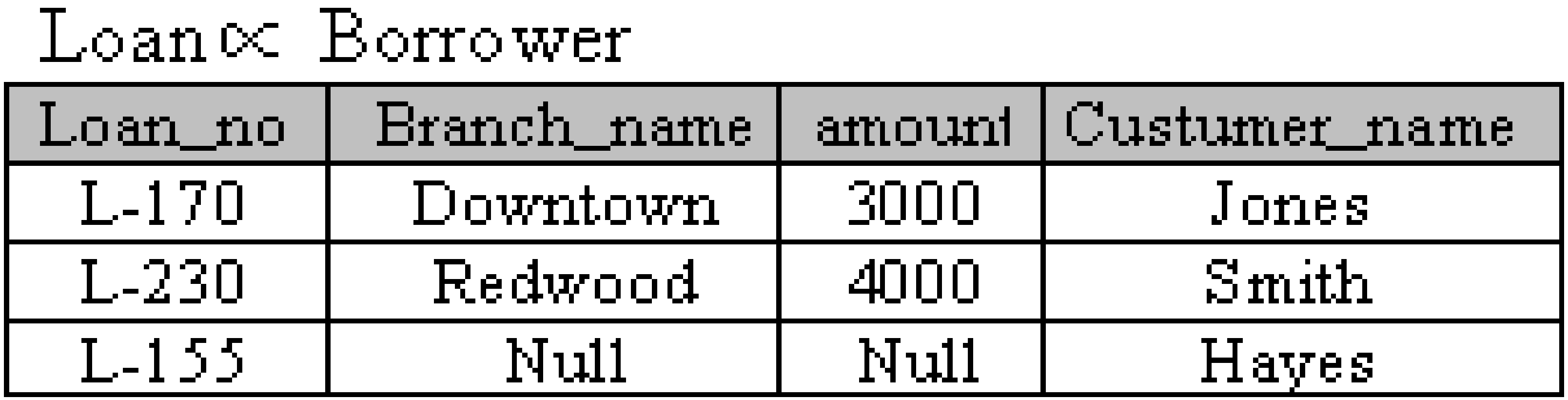
Loan

Loan_no	Branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700



Borrower

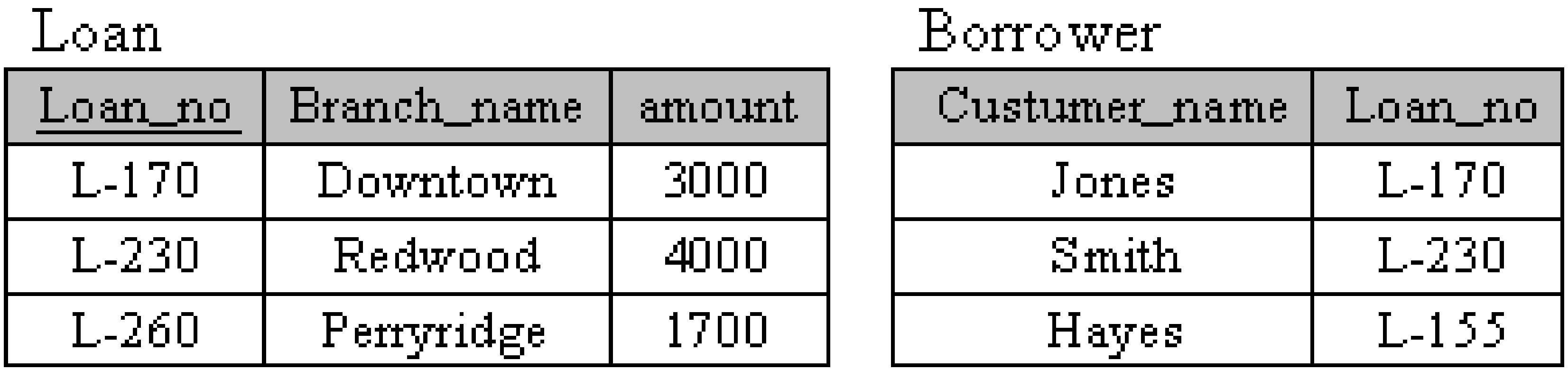
Custumer_name	Loan_no
Jones	L-170
Smith	L-230
Hayes	L-155



Loan∝ Borrower

Loan_no	Branch_name	amount	Custumer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	Null	Null	Hayes

Full Outer Join

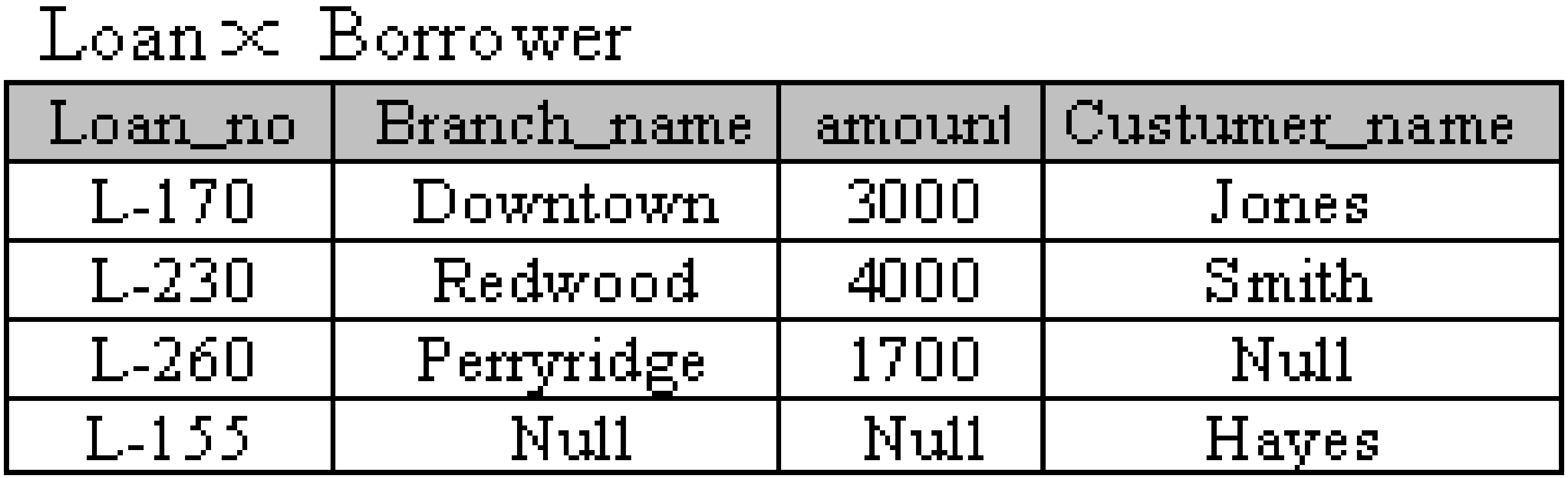


Loan

Loan_no	Branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

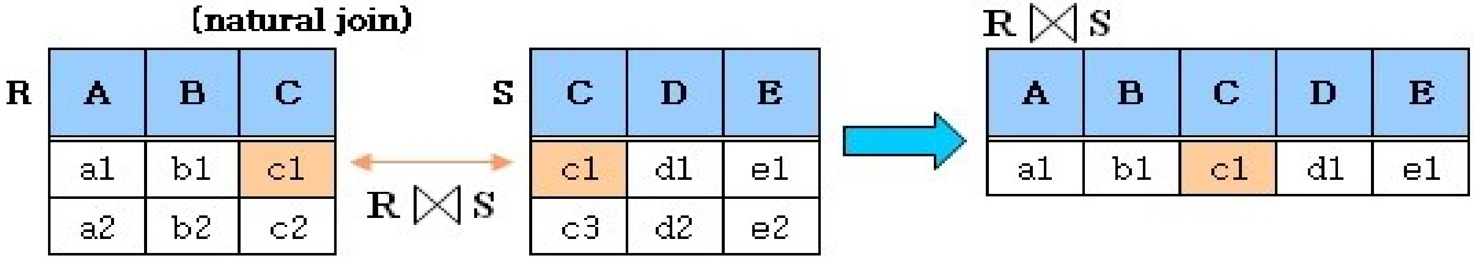
Borrower

Custumer_name	Loan_no
Jones	L-170
Smith	L-230
Hayes	L-155



Loan ⟗ Borrower

Loan_no	Branch_name	amount	Custumer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	Null
L-155	Null	Null	Hayes



(natural join)

R

A	B	C
a1	b1	c1
a2	b2	c2

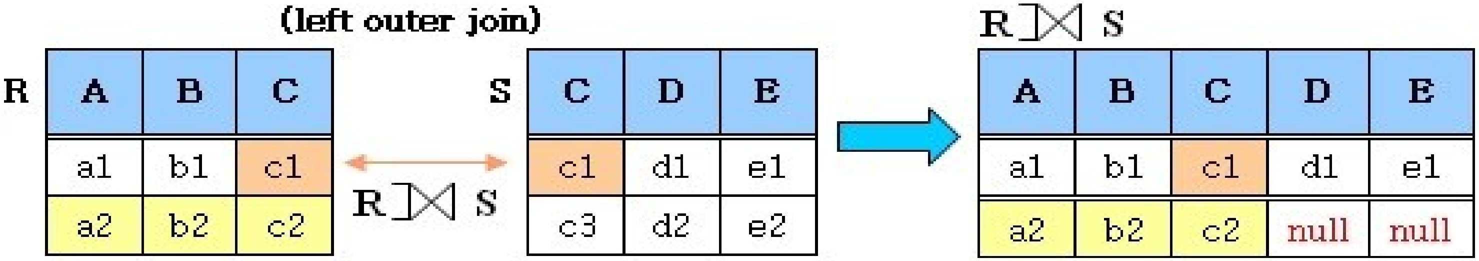
R ⋈ S

S

C	D	E
c1	d1	e1
c3	d2	e2

R ⋈ S

A	B	C	D	E
a1	b1	c1	d1	e1



(left outer join)

R

A	B	C
a1	b1	c1
a2	b2	c2

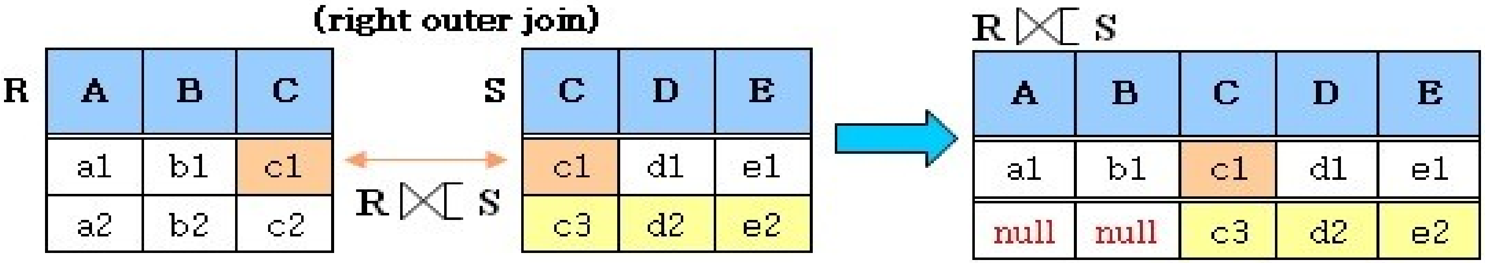
R ⟕ S

S

C	D	E
c1	d1	e1
c3	d2	e2

R ⟕ S

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b2	c2	null	null



(right outer join)

R

A	B	C
a1	b1	c1
a2	b2	c2

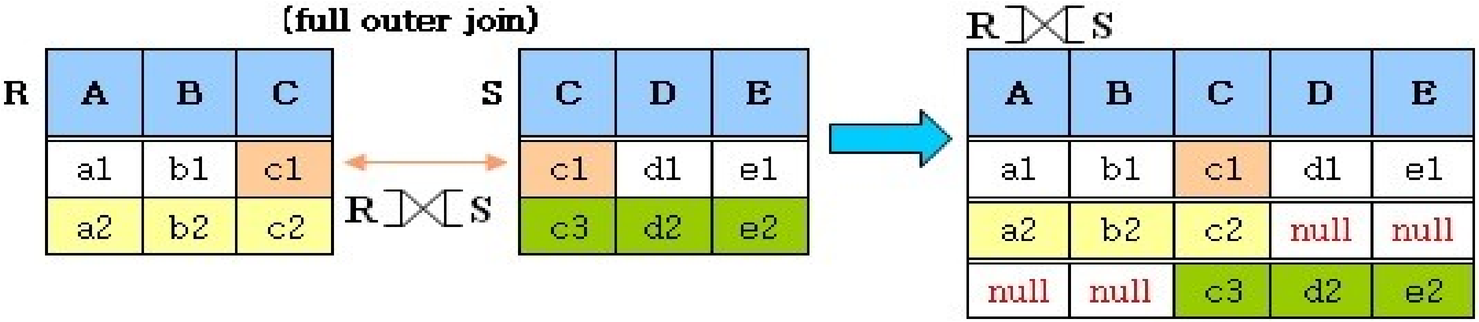
R ⟖ S

S

C	D	E
c1	d1	e1
c3	d2	e2

R ⟖ S

A	B	C	D	E
a1	b1	c1	d1	e1
null	null	c3	d2	e2



(full outer join)

R

A	B	C
a1	b1	c1
a2	b2	c2

R ⟗ S

S

C	D	E
c1	d1	e1
c3	d2	e2

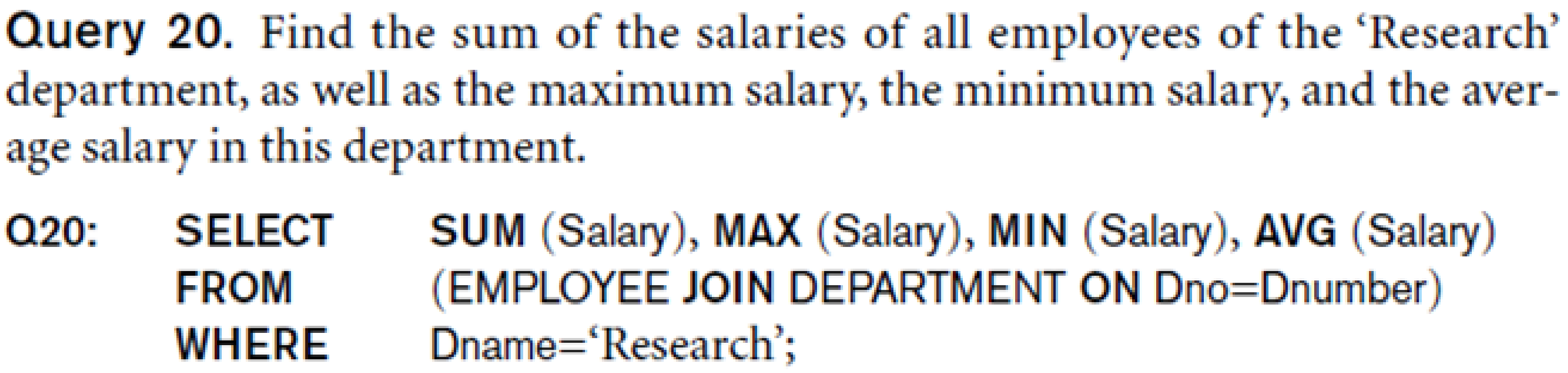
R ⟗ S

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

Aggregate Functions in SQL

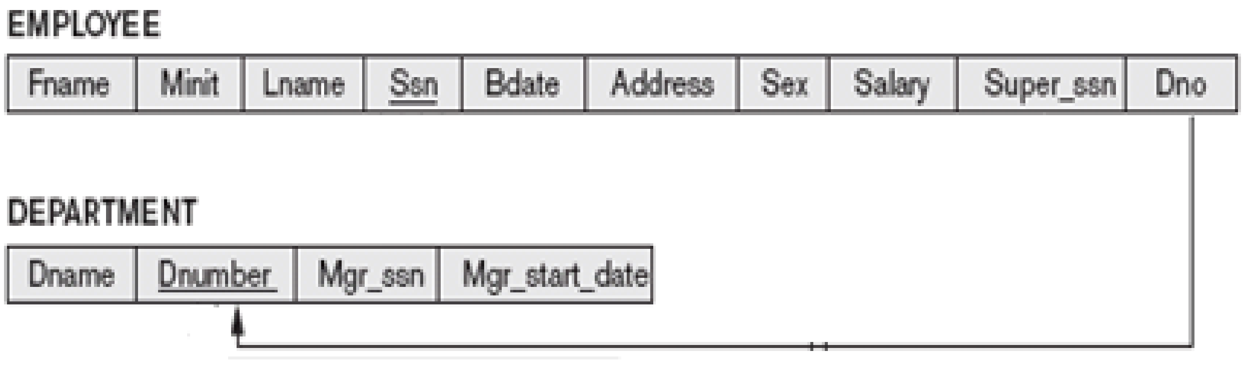
- Used to summarize information from multiple tuples into a single-tuple summary
- **Grouping**
  - Create subgroups of tuples before summarizing
- Built-in aggregate functions
  - **`COUNT`**, **`SUM`**, **`MAX`**, **`MIN`**, and **`AVG`**
- Functions can be used in the `SELECT` clause or in a `HAVING`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

Aggregate Functions in SQL



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';
```



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

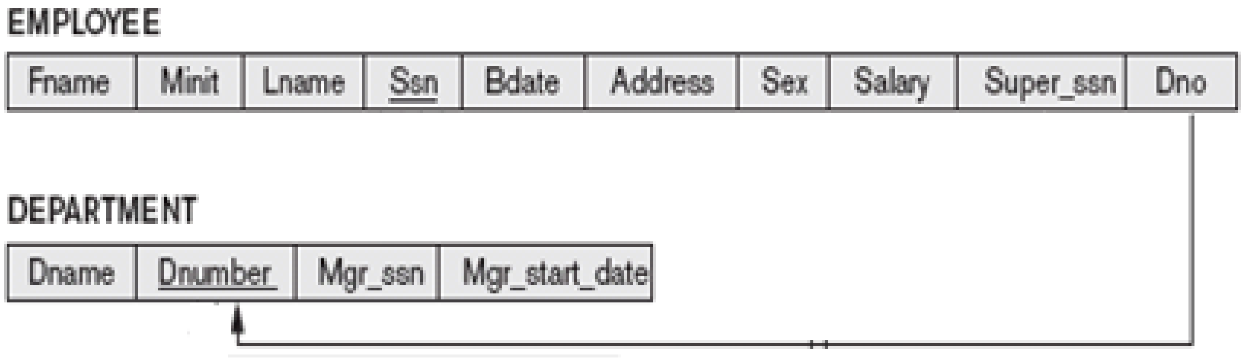
Dname	Dnumber	Mgr_ssn	Mgr_start_date

Aggregate Functions in SQL

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

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

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



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

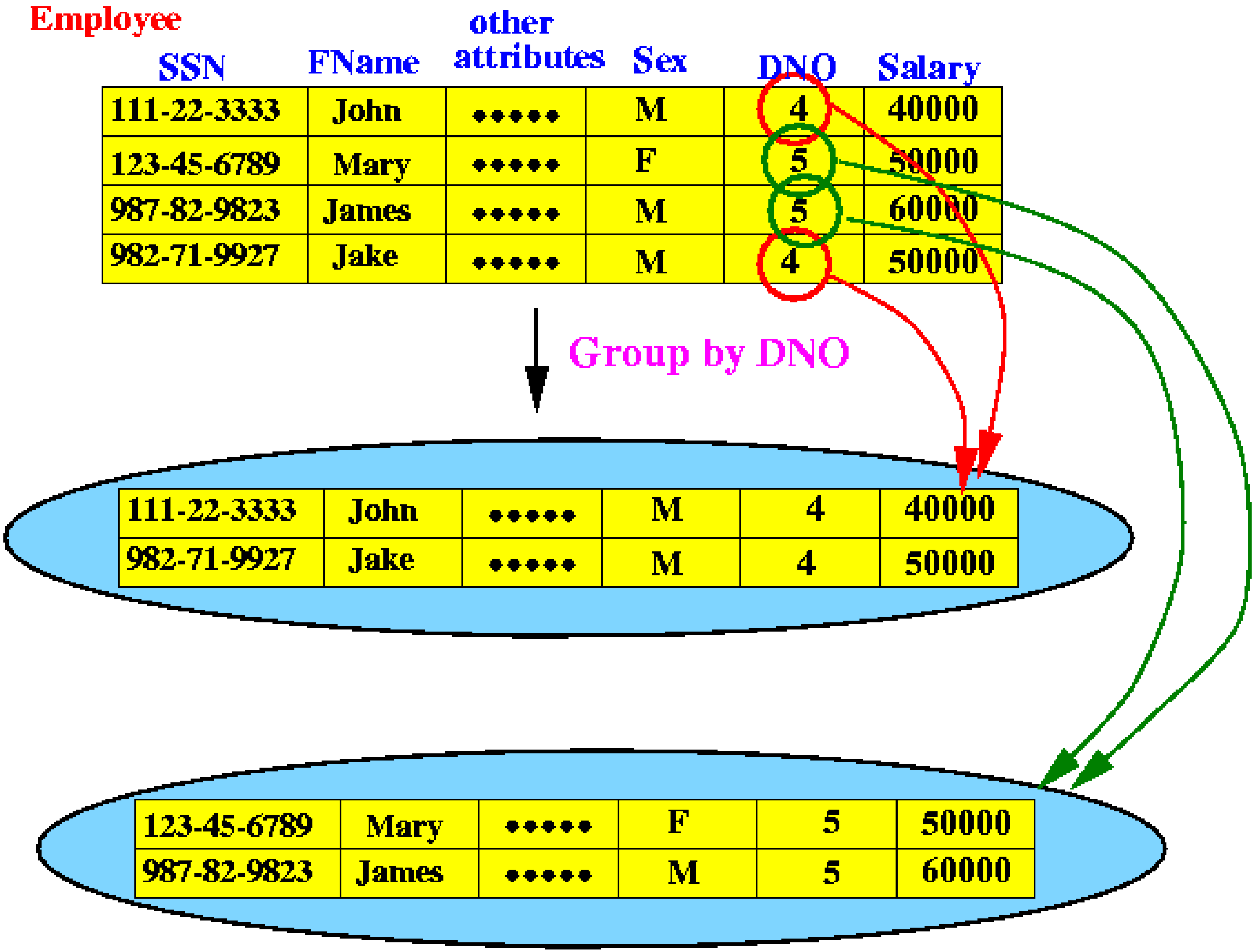
DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

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

“Group By” Example→ 1 attribute



Employee

SSN	FName	other attributes	Sex	DNO	Salary
111-22-3333	John	●●●●●	M	4	40000
123-45-6789	Mary	●●●●●	F	5	50000
987-82-9823	James	●●●●●	M	5	60000
982-71-9927	Jake	●●●●●	M	4	50000

Group by DNO

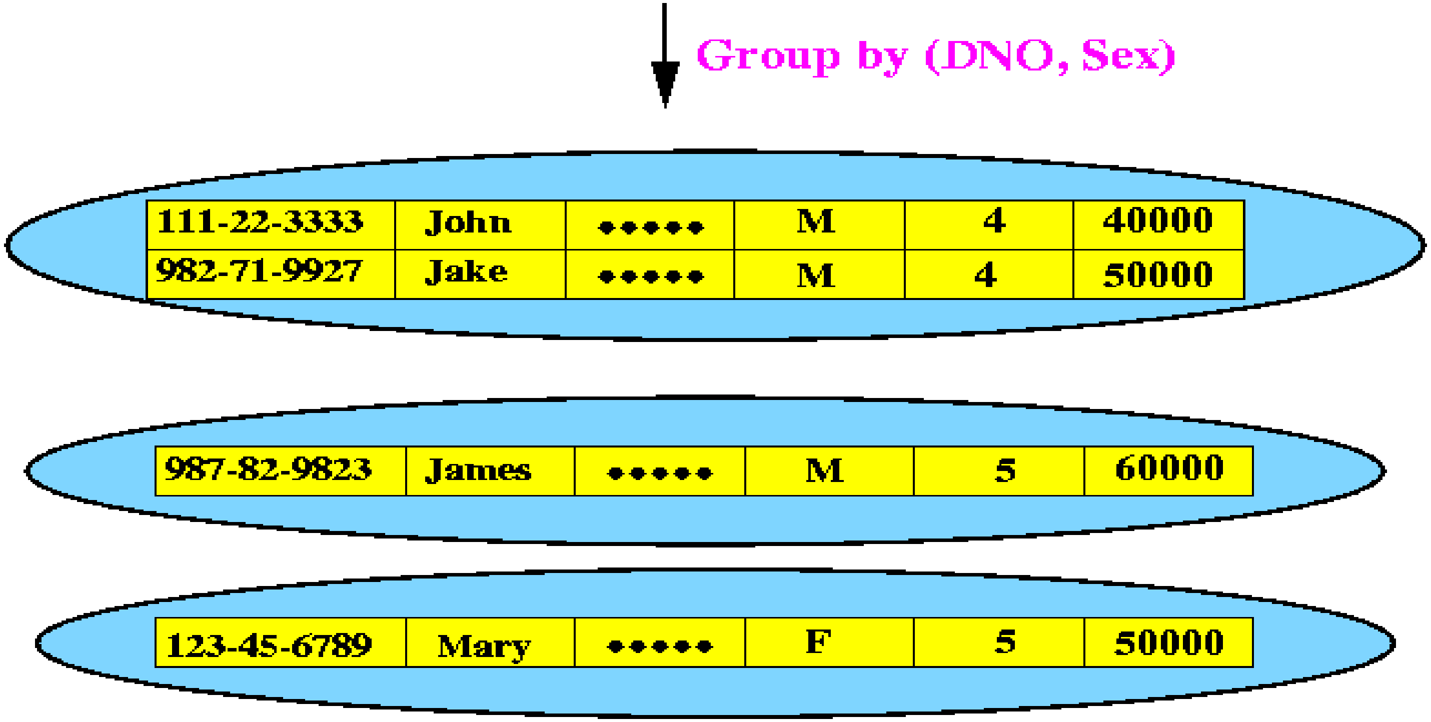
111-22-3333	John	●●●●●	M	4	40000
982-71-9927	Jake	●●●●●	M	4	50000

123-45-6789	Mary	●●●●●	F	5	50000
987-82-9823	James	●●●●●	M	5	60000

"Group By" Example→ more than 1 attribute

Employee

SSN	FName	other attributes	Sex	DNO	Salary
111-22-3333	John	•••••	M	4	40000
123-45-6789	Mary	•••••	F	5	50000
987-82-9823	James	•••••	M	5	60000
982-71-9927	Jake	•••••	M	4	50000



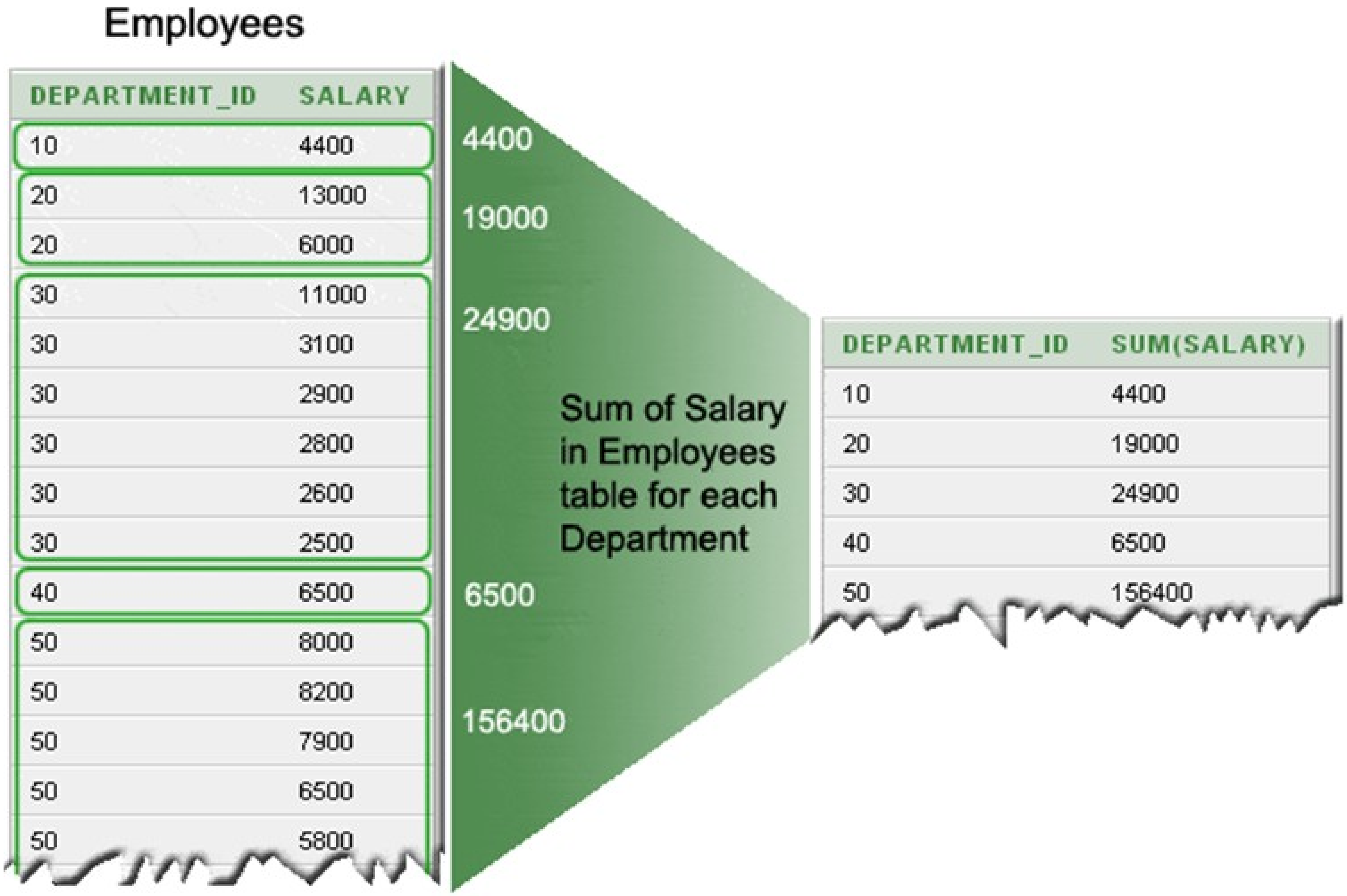
Group by (DNO, Sex)

111-22-3333	John	•••••	M	4	40000
982-71-9927	Jake	•••••	M	4	50000

987-82-9823	James	•••••	M	5	60000

123-45-6789	Mary	•••••	F	5	50000

Group By and Aggregate Function



Employees

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
30	11000
30	3100
30	2900
30	2800
30	2600
30	2500
40	6500
50	8000
50	8200
50	7900
50	6500
50	5800

4400

19000

24900

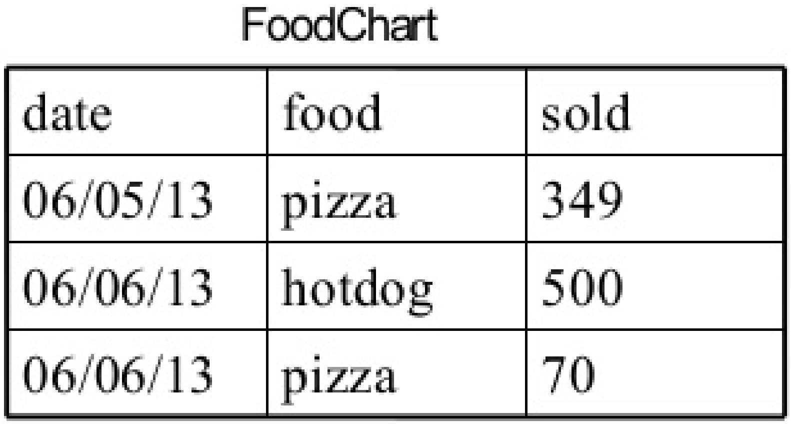
6500

156400

Sum of Salary in Employees table for each Department

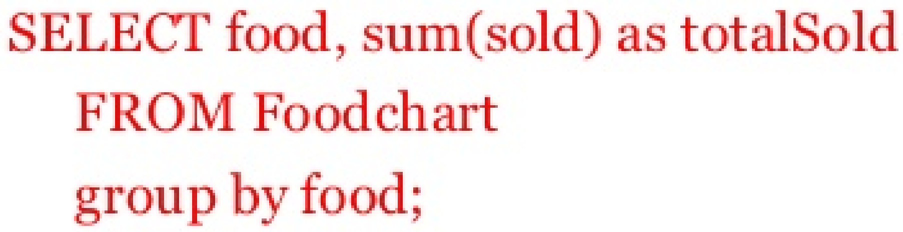
DEPARTMENT_ID	SUM(SALARY)
10	4400
20	19000
30	24900
40	6500
50	156400

Group By and Aggregate Function

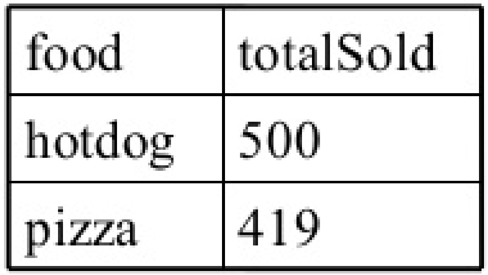


FoodChart

date	food	sold
06/05/13	pizza	349
06/06/13	hotdog	500
06/06/13	pizza	70



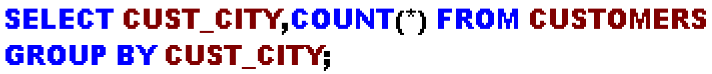
```
SELECT food, sum(sold) as totalSold
    FROM Foodchart
    group by food;
```



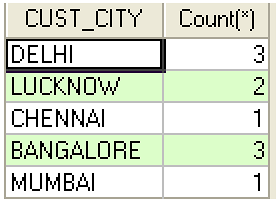
food	totalSold
hotdog	500
pizza	419

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

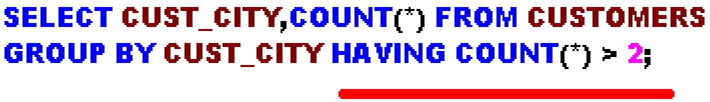
- **HAVING** clause
  - Provides a condition on the summary information



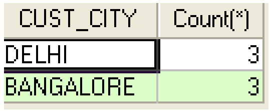
```sql
SELECT CUST_CITY,COUNT(*) FROM CUSTOMERS
GROUP BY CUST_CITY;
```



CUST_CITY	Count(*)
DELHI	3
LUCKNOW	2
CHENNAI	1
BANGALORE	3
MUMBAI	1

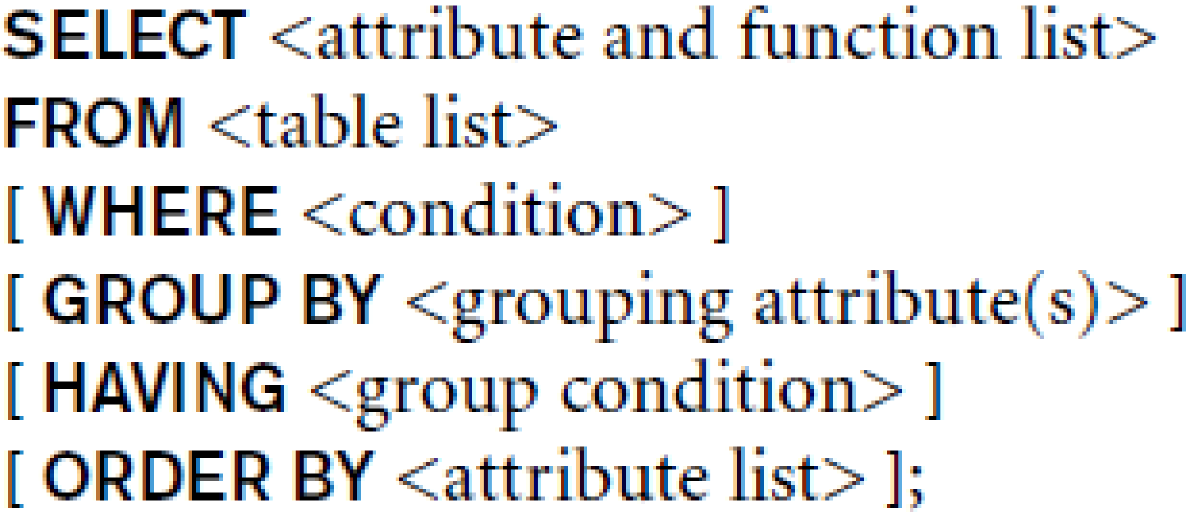


```sql
SELECT CUST_CITY,COUNT(*) FROM CUSTOMERS
GROUP BY CUST_CITY HAVING COUNT(*) > 2;
```



CUST_CITY	Count(*)
DELHI	3
BANGALORE	3

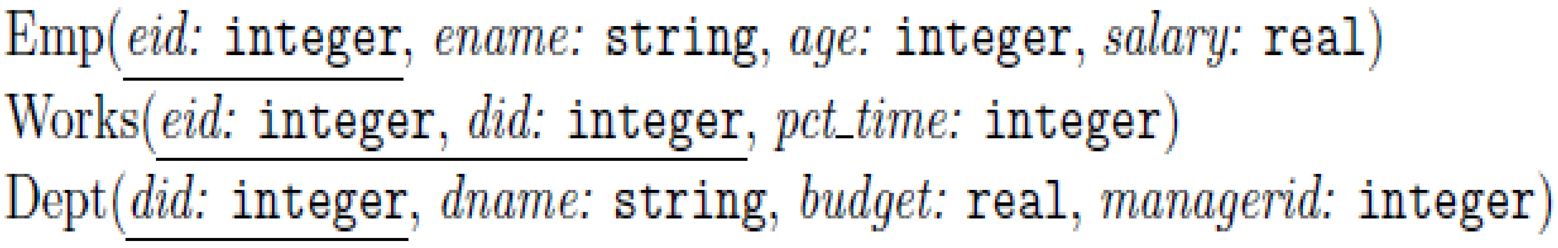
Discussion and Summary of SQL Queries



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

Quiz 1

- Consider the following relational schema. An employee can work in more than one department; the *pct time* field of the Works relation shows the percentage of time that a given employee works in a given department.



Emp(*eid:* integer, *ename:* string, *age:* integer, *salary:* real)
Works(*eid:* integer, *did:* integer, *pct_time:* integer)
Dept(*did:* integer, *dname:* string, *budget:* real, *managerid:* integer)

Quiz 1

- Write the following queries in SQL:
  - Print the names and ages of each employee who works in both the Hardware department and the Software department.
  - For each department with more than 20 full-time-equivalent employees (i.e., where the part-time and full-time employees add up to at least that many full-time employees), print the *did* together with the number of employees that work in that department.

Quiz 1

- – Print the name of each employee whose salary exceeds the budget of all of the departments that he or she works in.
- – Find the *ename*s of managers who manage only departments with budgets larger than $1 million, but at least one department with budget less than $5 million.

SQL Advanced

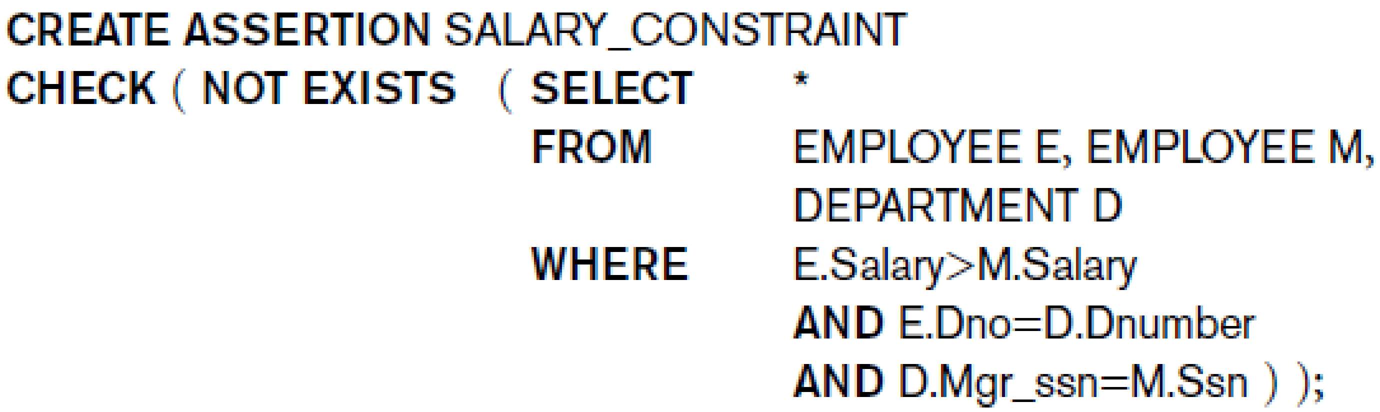
- More Complex SQL Retrieval Queries
- Specifying Constraints as Assertions and Actions as Triggers
- Views (Virtual Tables) in SQL
- Schema Change Statements in SQL

Specifying Constraints as Assertions and Actions as Triggers

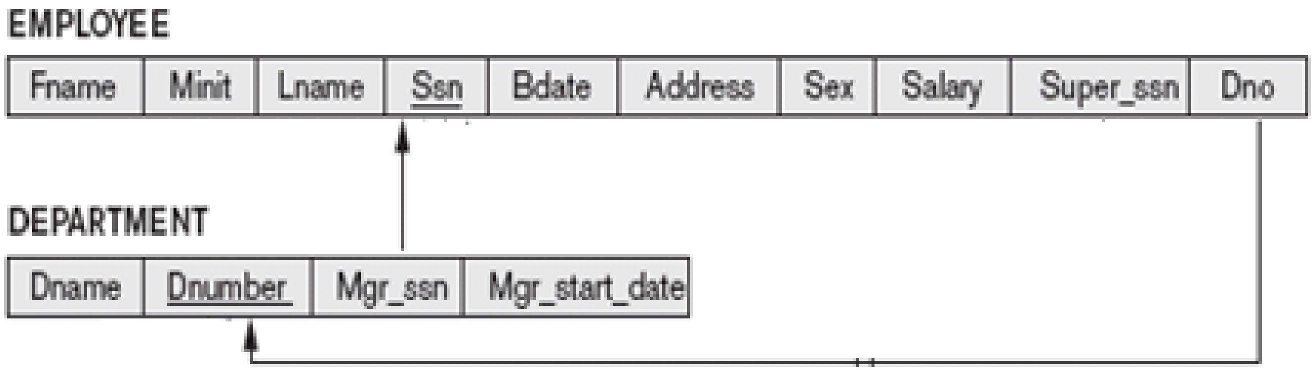
- **`CREATE ASSERTION`**
  - Specify additional types of constraints outside scope of built-in relational model constraints
- **`CREATE TRIGGER`**
  - Specify automatic actions that database system will perform when certain events and conditions occur

Specifying General Constraints as Assertions in SQL

- `CREATE ASSERTION`



```
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 ) );
```



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

Introduction to Triggers in SQL

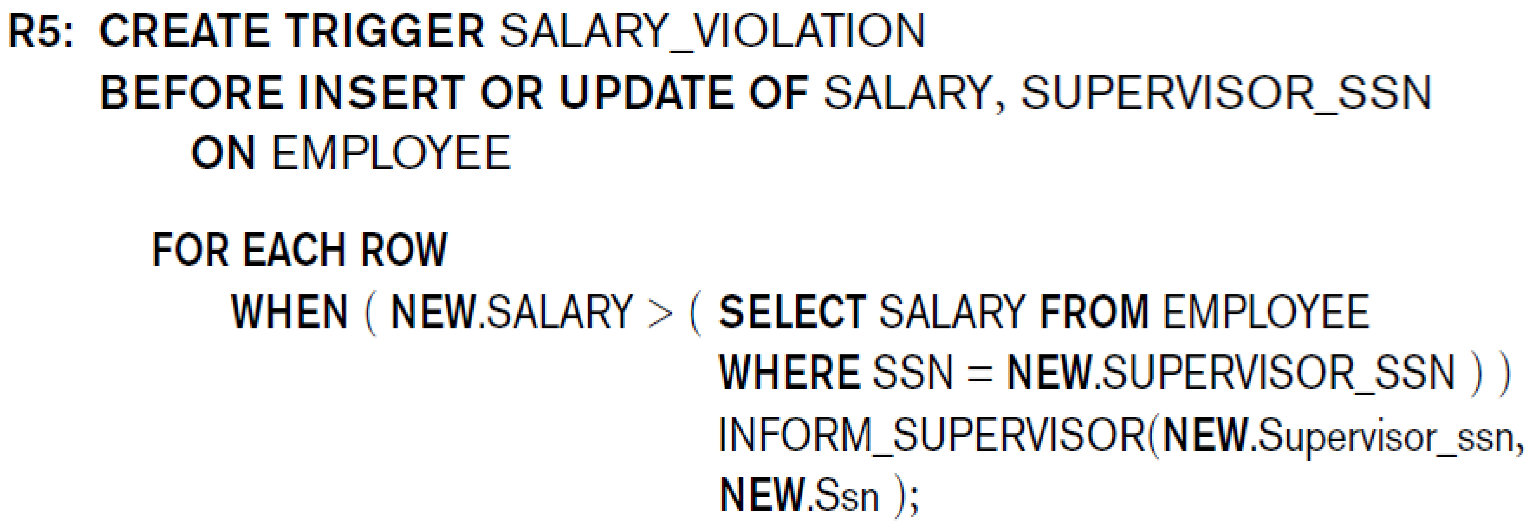
- `CREATE TRIGGER` statement
  - Used to monitor the database
- Typical trigger has three components:
  - **Event(s)**
  - **Condition**
  - **Action**

Triggers in SQL

- Check whenever an employee’s salary is greater than the salary of his or her direct supervisor in the COMPANY database
- Event:
  - Inserting a new employee record,
  - Changing an employee’s salary,
  - Changing an employee’s supervisor

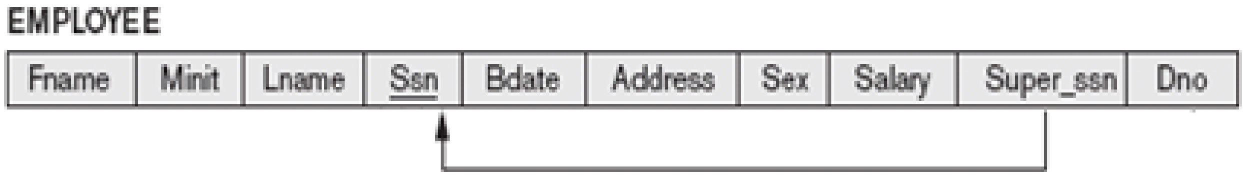
Triggers in SQL

- Action:
  - Execute the stored procedure INFORM_SUPERVISOR.
- Following trigger in oracle:



```
R5:  CREATE TRIGGER SALARY_VIOLATION
     BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
         ON EMPLOYEE

       FOR EACH ROW
          WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
                                WHERE SSN = NEW.SUPERVISOR_SSN ) )
                                INFORM_SUPERVISOR(NEW.Supervisor_ssn,
                                NEW.Ssn );
```



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

SQL Advanced

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

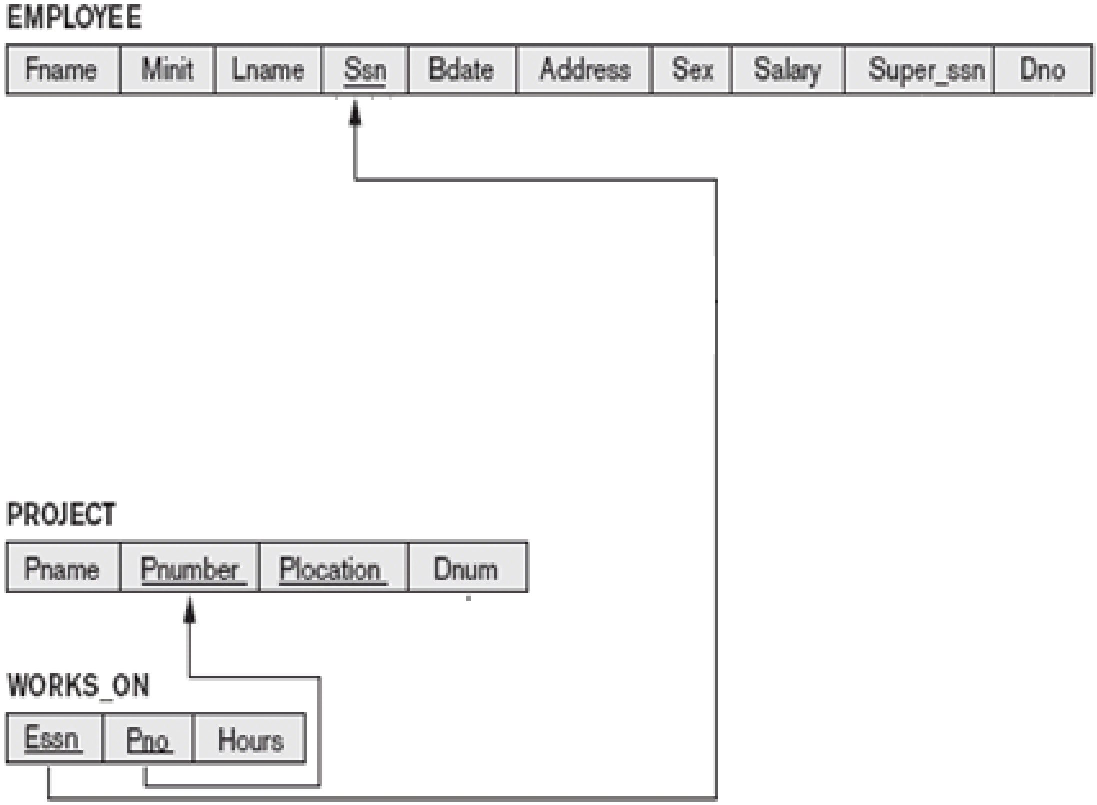
Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables
  - Considered to be a virtual table

Specification of Views in SQL

- **`CREATE VIEW`** command
  - Give table name, list of attribute names, and a query to specify the contents of the view

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



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

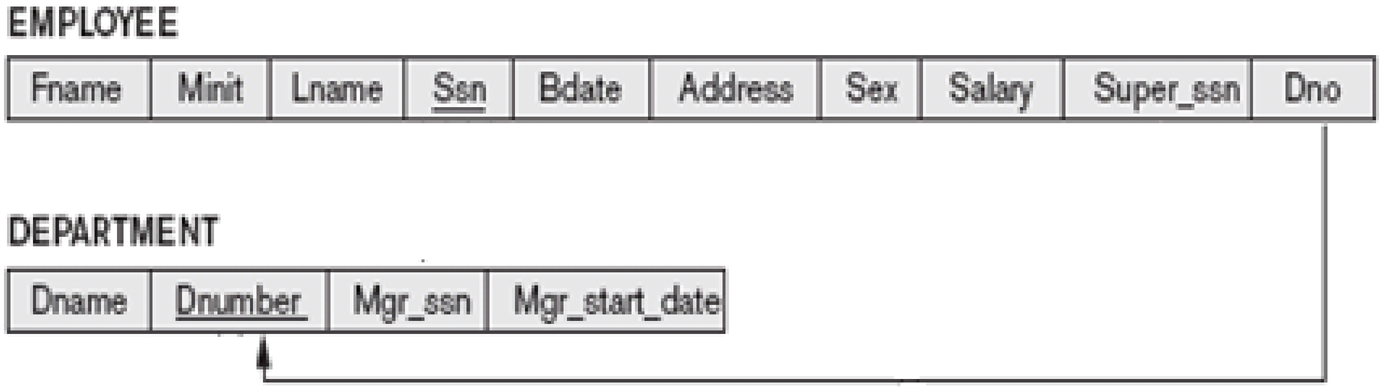
PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

```
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;
```



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

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

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

View Implementation (cont’d.)

- **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

SQL Advanced

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

Schema Change Statements in SQL

- **Schema evolution commands**
  - Can be done while the database is operational
  - Does not require recompilation of the database schema

The DROP Command

- `DROP` command
  - Used to drop named schema elements, such as tables, domains, or constraint
- Drop behavior options:
  - `CASCADE` and `RESTRICT`
- Example:
  - `DROP SCHEMA COMPANY CASCADE;`
    - Automatically drop objects (tables, functions, etc.) that are contained in the schema.
-

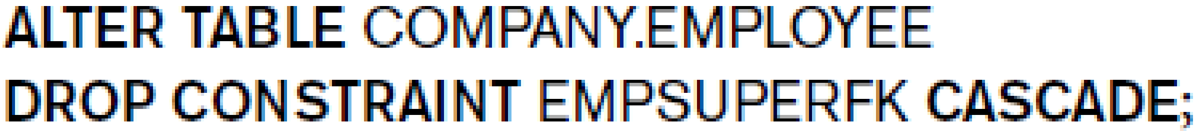
The ALTER Command

- **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`

- When you remove a column from a table, will automatically remove all of the indexes and constraints that involved the dropped column.
- If the column that you want to remove is used in other database objects such views, triggers, stored procedures, etc., you cannot drop the column because other objects are depending on it.
  - In this case, you need to add the CASCADE option to the DROP COLUMN clause to drop the column and all of its dependent objects:

The ALTER Command (cont'd.)

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



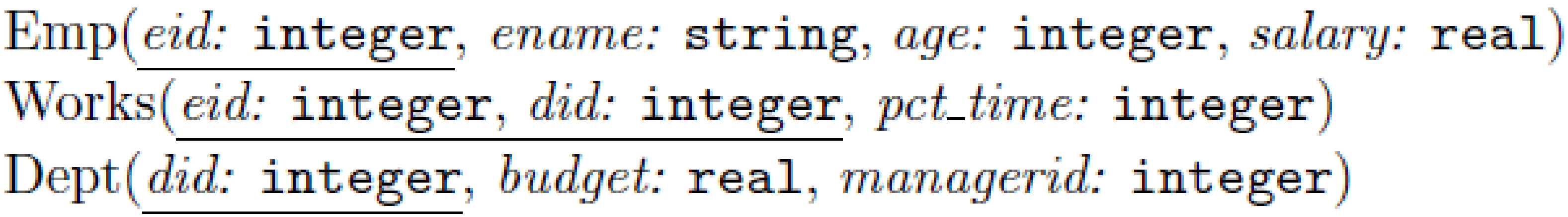
```
ALTER TABLE COMPANY.EMPLOYEE
DROP CONSTRAINT EMPSUPERFK CASCADE;
```

Summary

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

Quiz 2

- Consider the following relational schema and briefly answer the questions that follow:



Emp(*eid:* `integer`, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(*eid:* `integer`, *did:* `integer`, *pct_time:* `integer`)
Dept(*did:* `integer`, *budget:* `real`, *managerid:* `integer`)

- Next Page->

Quiz 2

- 1. Define a table constraint on Emp that will ensure that every employee makes at least $10,000.
- 2. Define a table constraint on Dept that will ensure that all managers have *age* > 30.
- 3. Define an assertion on Dept that will ensure that all managers have *age* > 30. Compare this assertion with the equivalent table constraint. Explain which is better.

- CREATE TABLE Emp (eid INTEGER, ename CHAR(10), age INTEGER , salary REAL, PRIMARY KEY (eid), CHECK (*salary* >= 10000))