INTRODUCTION TO THE RELATIONAL MODEL

- The main construct for representing data in the relational model is a **relation**

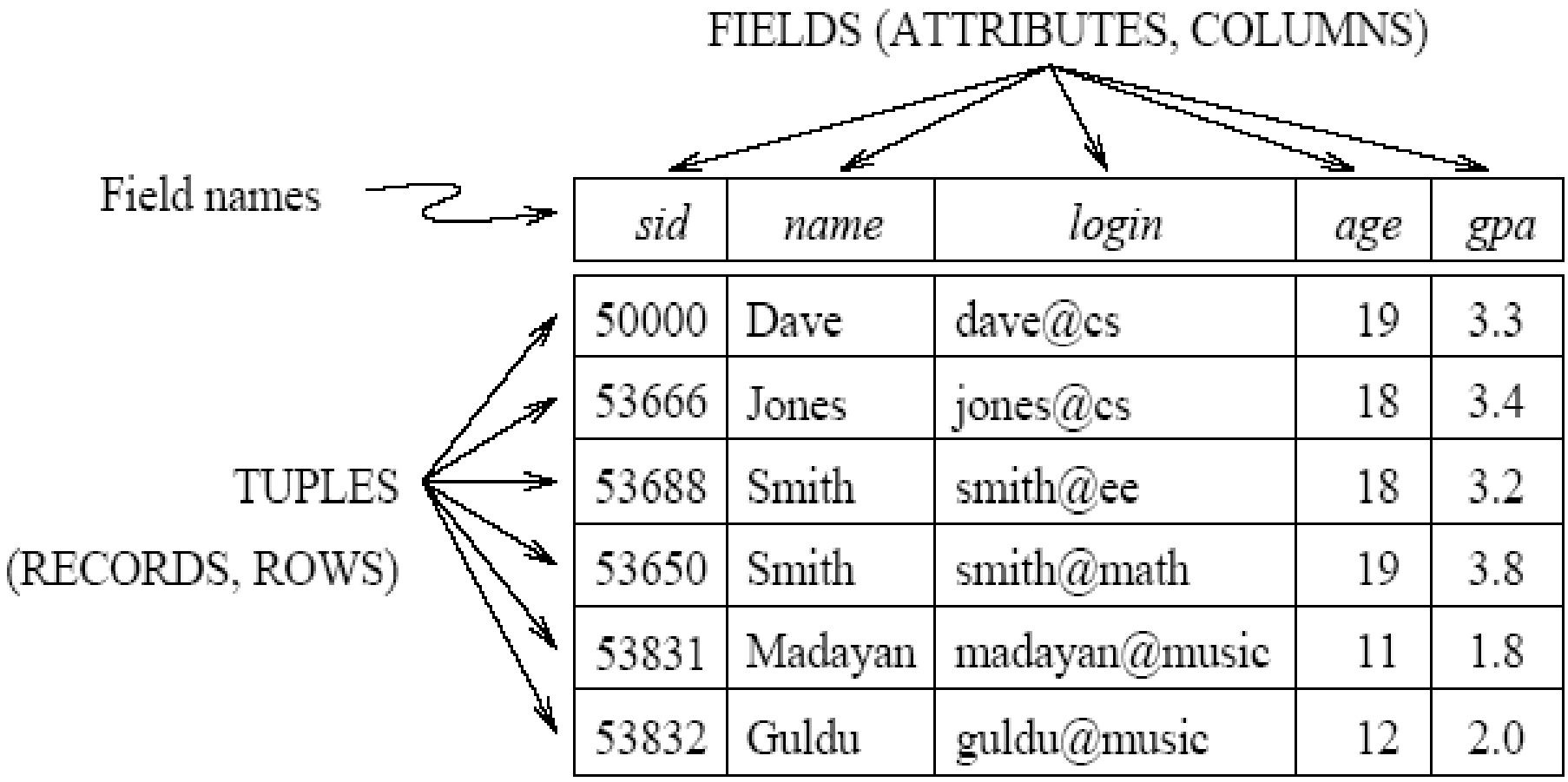
- A relation consists of a **relation schema** and a **relation instance**

- The schema specifies the relation's name, the name of each **field** (or **column**, or **attribute**), and the **type** of each field

Example:

Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)

- An **instance** of the Students relation appears in Figure 3.1



sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure 3.1 An Instance $S1$ of the Students Relation

- The **degree**, also called **arity**, of a relation is the number of fields

- The **cardinality** of a relation instance is the number of tuples in it

- In Figure 3.1, the degree of the relation (the number of columns) is **five**, and the cardinality of this instance is **six**

- A **relational database** is a collection of relations with distinct relation names

- The **relational database schema** is the collection of schemas for the relations in the database

Creating and Modifying Relations Using SQL

- To create the Students relation, use the following statement:

```
CREATE TABLE Students
(
   sid NUMBER(5),
   name VARCHAR2(10),
   login VARCHAR2(20),
   age NUMBER(2),
   gpa NUMBER(2,1)
)
```

- Tuples are inserted using the INSERT command

- We can insert a single tuple into the Students table as follows:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- We can delete tuples using the DELETE command

- We can delete all Students tuples with *name* equal to Smith using the command:

```
DELETE FROM Students
WHERE name = 'Smith'
```

- We can modify the column values in an existing row using the UPDATE command

- For example, we can increment the age and decrement the gpa of the student with *sid* 53688:

```
UPDATE Students
SET age = age + 1, gpa = gpa - 1
WHERE sid = 53688
```

INTEGRITY CONSTRAINTS OVER RELATIONS

- An **integrity constraint (IC)** is a condition that is specified on a database schema, and verified on the data that is to be stored in an instance of the database

- If a database instance satisfies all the integrity constraints specified on the database schema, it is a **legal** instance

- A **DBMS enforces** integrity constraints, in that it permits only legal instances to be stored in the database

- Integrity constraints are **specified** and **enforced** at different times:

1.When the DBA or end user defines a database schema

2.When a database application is run, the DBMS checks for integrity constraints

Key Constraints

- Consider the Students relation and the constraint that no two students have the same sid

- This Integrity Constraint is an example of a key constraint

A **key constraint** is a *minimal* subset of the fields of a relation that uniquely identifies a tuple

- A set of fields that uniquely identifies a tuple is called a **candidate key** for the relation

- A relation may have several candidate keys

- In the Students relation, **candidate keys** are *login* and *sid*

- Out of all the available candidate keys, a database designer can identify a **primary key**

- The set *{sid, name}* is an example of a **superkey**, which is a set of fields that contains a key

Specifying Key Constraints

SQL UNIQUE Constraint

- The UNIQUE constraint uniquely identifies each record in a database table

- Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table

Example:

```
CREATE TABLE Students
(
   sid NUMBER(5),
   name VARCHAR2(10) UNIQUE,
   login VARCHAR2(20),
   age NUMBER(2),
   gpa NUMBER(2,1)
)
```

SQL PRIMARY KEY Constraint

▪The PRIMARY KEY constraint uniquely identifies each record in a database table

Example:

```
CREATE TABLE Students
(
   sid NUMBER(5) PRIMARY KEY,
   name VARCHAR2(10),
   login VARCHAR2(20),
   age NUMBER(2),
   gpa NUMBER(2,1)
)
```

SQL FOREIGN KEY Constraint

- A FOREIGN KEY in one table points to a PRIMARY KEY in another table

- If one of the relations is modified, the other must be checked, and perhaps modified, to keep the data consistent

- The ***sid*** field of **Enrolled** is called a **foreign key** and refers to Students

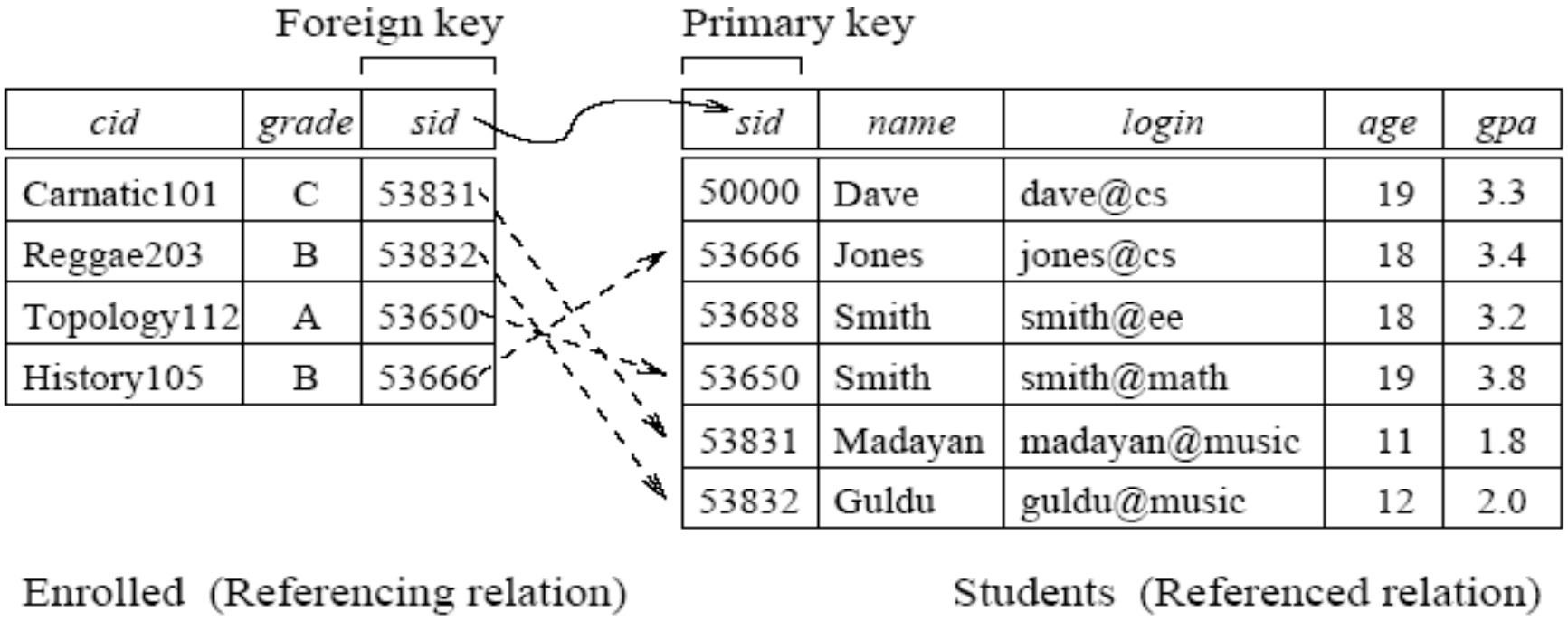


Figure 3.4 Referential Integrity

- **Foreign kay** value must be one of the values of the **Primary key**

- If we try to insert the tuple <*55555*, *Art104*, *A*> into *Enrolled*, the IC is violated because there is no tuple in *Students* with the id 55555

- Similarly, if we delete the tuple <*53666*, *Jones*, *jones@cs*, *18*, *3.4*> from *Students*, we violate the foreign key constraint because the tuple <*53666*, *History105*, *B*> in *Enrolled* contains *sid* value 53666, the *sid* of the deleted Students tuple

Example:

```
CREATE TABLE Enrolled
(
 cid VARCHAR2(20) PRIMARY KEY,
 grade VARCHAR2(2),
 sid NUMBER(5),
 FOREIGN KEY(sid) REFERENCES Students(sid)
 )
```

General Constraints

- CHECK constraint is used to limit the value range that can be placed in a column

Example: All students must be at least 16 years old (Fig 3.5)

```
CREATE TABLE Students
(
   sid NUMBER(5) PRIMARY KEY,
   name VARCHAR2(10),
   login VARCHAR2(20),
   age NUMBER(2) CHECK (age>16),
   gpa NUMBER(2,1)
)
```

sid	*name*	*login*	*age*	*gpa*
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8

Figure 3.5 An Instance $S2$ of the Students Relation

ENFORCING INTEGRITY CONSTRAINTS

- ICs are **specified** when a relation is **created** and **enforced** when a relation is **modified**

- Potential **IC violation** is generally checked at the end of each **SQL statement execution**

- Consider the Students relation shown in Figure 3.1

- The following insertion violates the primary key constraint because there is already a tuple with the *sid* 53688, and it will be rejected by the DBMS:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Mike', 'mike@ee', 17, 3.4)
```

- The following insertion violates the constraint that the primary key cannot contain *null:*

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (null, 'Mike', 'mike@ee', 17, 3.4)
```

- An update can cause violations, similar to an insertion:

UPDATE Students SET sid = 50000 WHERE sid = 53688

- This update violates the primary key constraint because there is already a tuple with *sid* 50000

referential integrity enforcement(on foreign key)

- Deletions of Enrolled tuples do not violate referential integrity, but insertions of Enrolled tuples could

- The following insertion is illegal because there is no student with *sid* 51111:

INSERT INTO Enrolled (cid, grade, sid)
VALUES ('Hindi101', 'B', 51111)

- On the other hand, insertions of Students tuples do not violate referential integrity although deletions could

- Further, updates on either Enrolled or Students that change the *sid* value could potentially violate referential integrity

```
CREATE TABLE Enrolled
(
 cid VARCHAR2(20) PRIMARY KEY,
 grade VARCHAR2(2),
 sid NUMBER(5),
 FOREIGN KEY(sid) REFERENCES Students(sid)
```

```
    ON DELETE CASCADE
    ON UPDATE NO ACTION
)
```

- The options are specified as part of the foreign key declaration

- The default option is NO ACTION, which means that the action (DELETE or UPDATE) is to be rejected

- The CASCADE keyword says that if a Students row is deleted, all Enrolled rows that refer to it are to be deleted as well

- If the UPDATE clause specified CASCADE, and the *sid*

column of a Students row is updated, this update is also carried out in each Enrolled row that refers to the updated Students row

- If a Students row is deleted, we can switch the enrollment to a 'default' student by using ON DELETE SET DEFAULT

- The default student is specified as part of the definition of the *sid* field in Enrolled; for example, *sid* NUMBER(5) DEFAULT *'53666'*

- It is really not appropriate to switch enrollments to a default student

- The correct solution in this example is to also delete all

enrollment tuples for the deleted student (that is, CASCADE), or to reject the update

- SQL also allows the use of *null* as the default value by specifying ON DELETE SET NULL

QUERYING RELATIONAL DATA

- A **relational database query** (query, for short) is a question about the data, and the answer consists of a new relation containing the result

- For example, find all students younger than 18 or all students enrolled in History105

- A **query language** is a language for writing queries

- SQL is the most popular commercial query language for a relational DBMS

- We can retrieve rows corresponding to students who are younger than 18 with the following SQL query:

```
SELECT * FROM Students S WHERE S.age < 18
```

- The symbol * means that we retain all fields of selected tuples in the result

- To understand this query, think of S as a variable that takes

on the value of each tuple in Students, one tuple after the other

- The condition *S.age < 18* in the WHERE clause specifies that we want to select only tuples in which the age field has a value less than 18

- Result is shown in Fig 3.6

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure 3.6 Students with $age < 18$ on Instance $S1$

- In addition to selecting a subset of tuples, a query can extract a subset of the fields of each selected tuple

- We can compute the names and logins of students who are younger than 18 with the following query:

```
SELECT S.name, S.login FROM Students S
WHERE S.age < 18
```

- Figure 3.7 shows the result

name	*login*
Madayan	madayan@music
Guldu	guldu@music

Figure 3.7 Names and Logins of Students under 18

- We can also combine information in the **Students** and **Enrolled** relations

- If we want to obtain the names of all students who obtained grade A and the cid in which they got grade A, we could write the following query:

```
SELECT S.name, E.cid FROM Students S, Enrolled E
WHERE S.sid = E.sid  AND E.grade = 'A'
```

- Answer is <Smith, Topology112>

LOGICAL DATABASE DESIGN: ER TO RELATIONAL

- The ER model is convenient for representing an initial, high-level database design

Entity Sets to Tables

- Each attribute of the entity set becomes an attribute of the table

- Note that we know both the domain of each attribute and the (primary) key of an entity set

- Consider the Employees entity set with attributes *ssn*, *name*, and *lot* shown in Figure 3.8

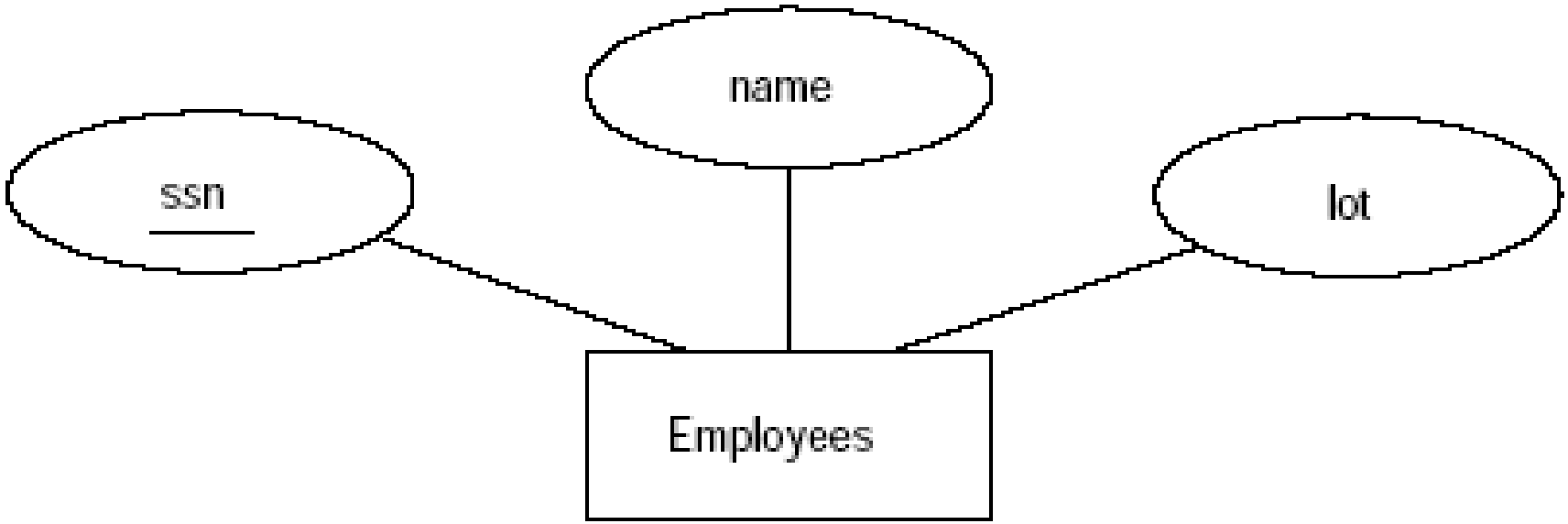


Figure 3.8 The Employees Entity Set

- A possible instance of the Employees entity set, containing three Employees entities, is shown in Figure 3.9 in a tabular format

ssn	*name*	*lot*
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

Figure 3.9 An Instance of the Employees Entity Set

- The following SQL statement captures the preceding information, including the domain constraints and key information:

```
CREATE TABLE Employees
(
 ssn VARCHAR2(11) PRIMARY KEY,
 name VARCHAR2(30),
 lot NUMBER(2)
)
```

Relationship Sets (without Constraints) to Tables

- A relationship set, like an entity set, is mapped to a relation in the relational model

- Thus, the attributes of the relation include:

  --The primary key attributes of each participating entity set, as foreign key fields
  --The descriptive attributes of the relationship set

- Consider the Works_In2 relationship set shown in Figure 3.10

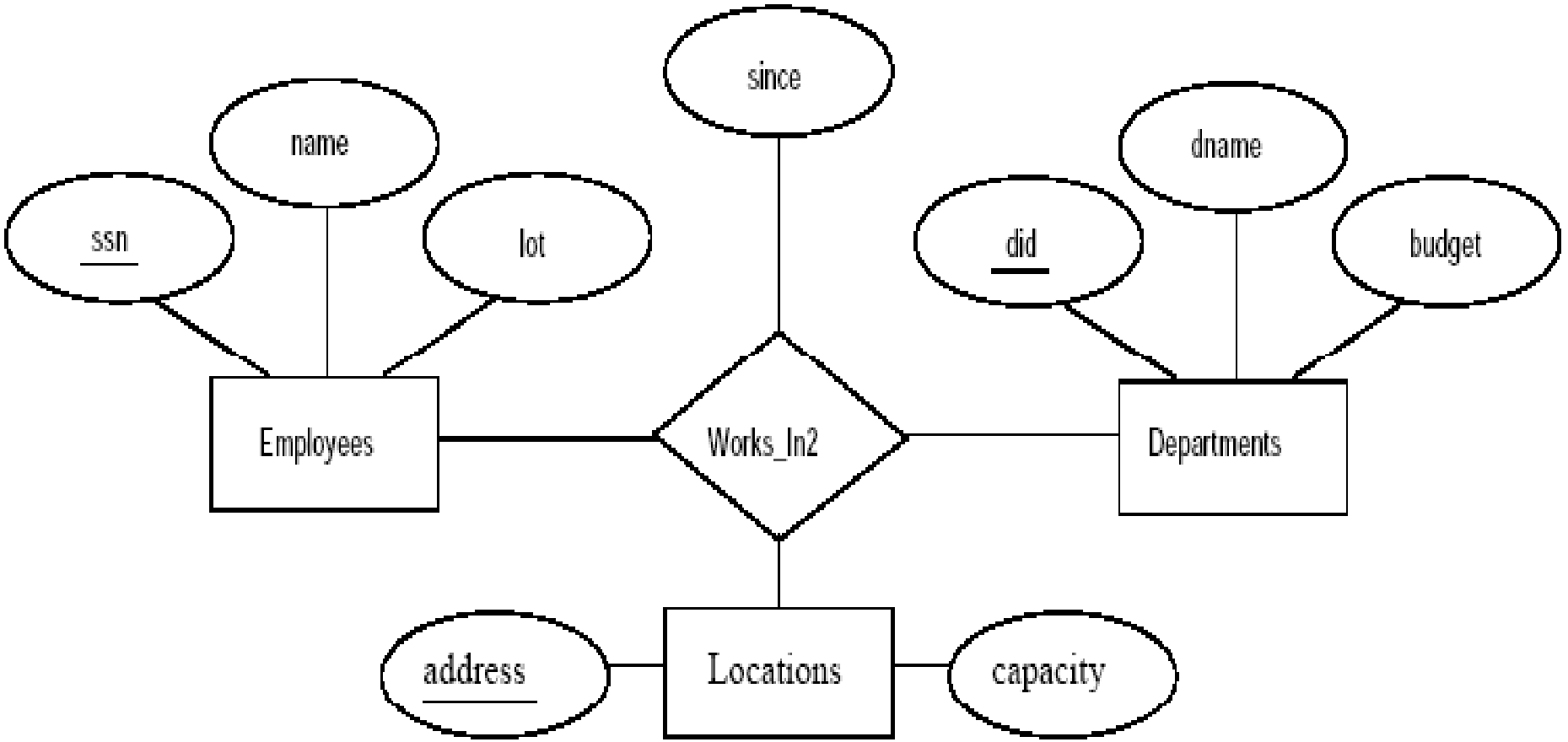


Figure 3.10 A Ternary Relationship Set

- All the available information about the Works_In2 table is captured by the following SQL definition:

```
CREATE TABLE Works_In2
(
  ssn VARCHAR2(11),
  did NUMBER(2),
  address VARCHAR2(20),
  since DATE,
  PRIMARY KEY(ssn, did, address),
  FOREIGN KEY(ssn) REFERENCES Employees(ssn),
  FOREIGN KEY(did) REFERENCES Departments(did),
  FOREIGN KEY(address) REFERENCES
  Locations(address) )
```

▪Consider the Reports_To relationship set shown in Figure 3.11

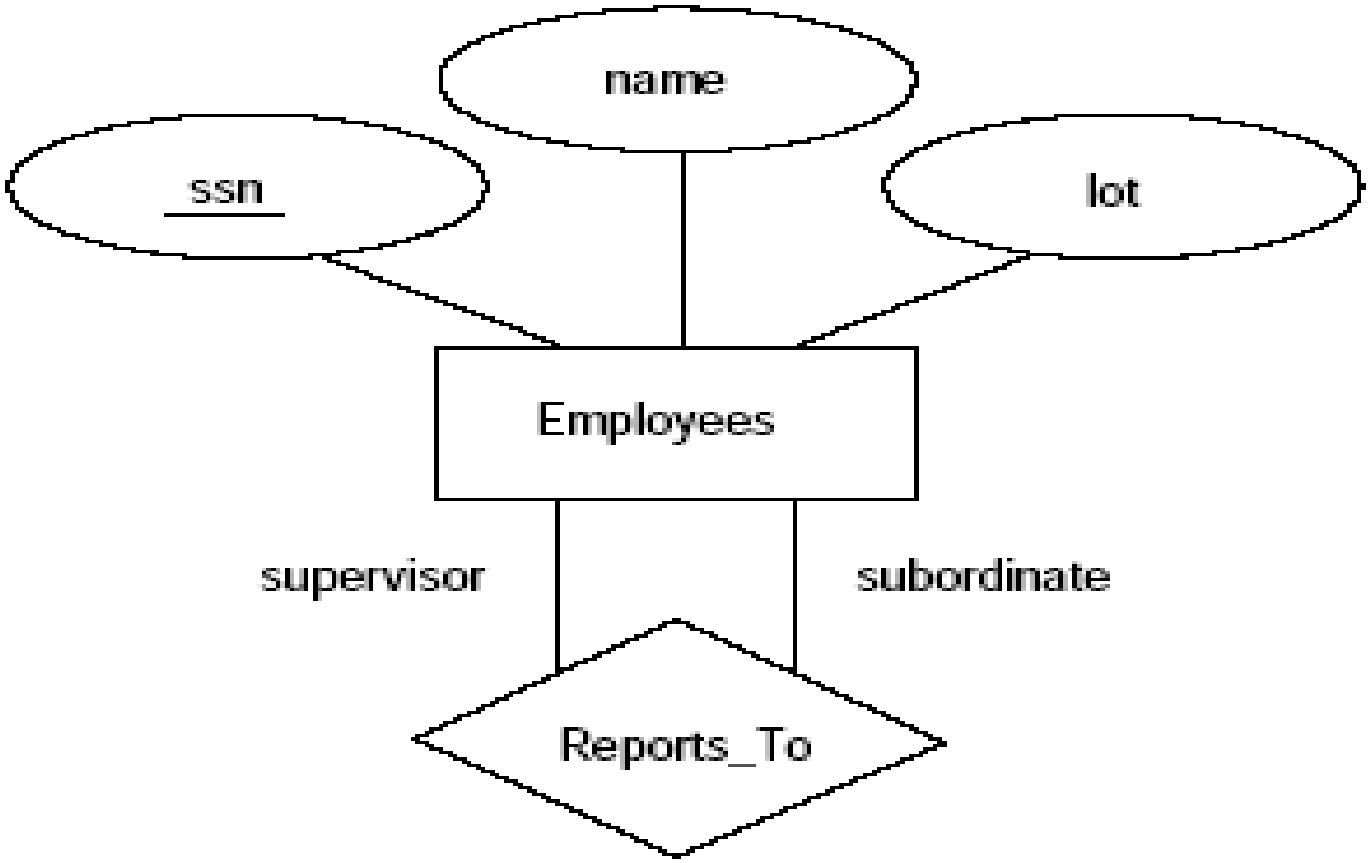


Figure 3.11 The Reports_To Relationship Set

- The role indicators *supervisor* and *subordinate* are used to create meaningful field names in the Reports_To table:

```
CREATE TABLE Reports_To
(
 supervisor_ssn VARCHAR2(11),
 subordinate_ssn VARCHAR2(11),
 PRIMARY KEY(supervisor_ssn, subordinate_ssn),
 FOREIGN KEY(supervisor_ssn) REFERENCES
 Employees(ssn),
 FOREIGN KEY(subordinate_ssn) REFERENCES
 Employees(ssn)
)
```

Translating Relationship Sets with Key Constraints

- If a relationship set involves *n* entity sets and some m of them are linked via arrows in the ER diagram, the key for any one of these *m* entity sets constitutes a key for the relation to which the relationship set is mapped

- Thus we have *m* candidate keys, and one of these should be designated as the primary key

- Consider the relationship set Manages shown in Figure3.12

- The table corresponding to Manages has the attributes *ssn*, *did*, *since* with *did* as *primary key*

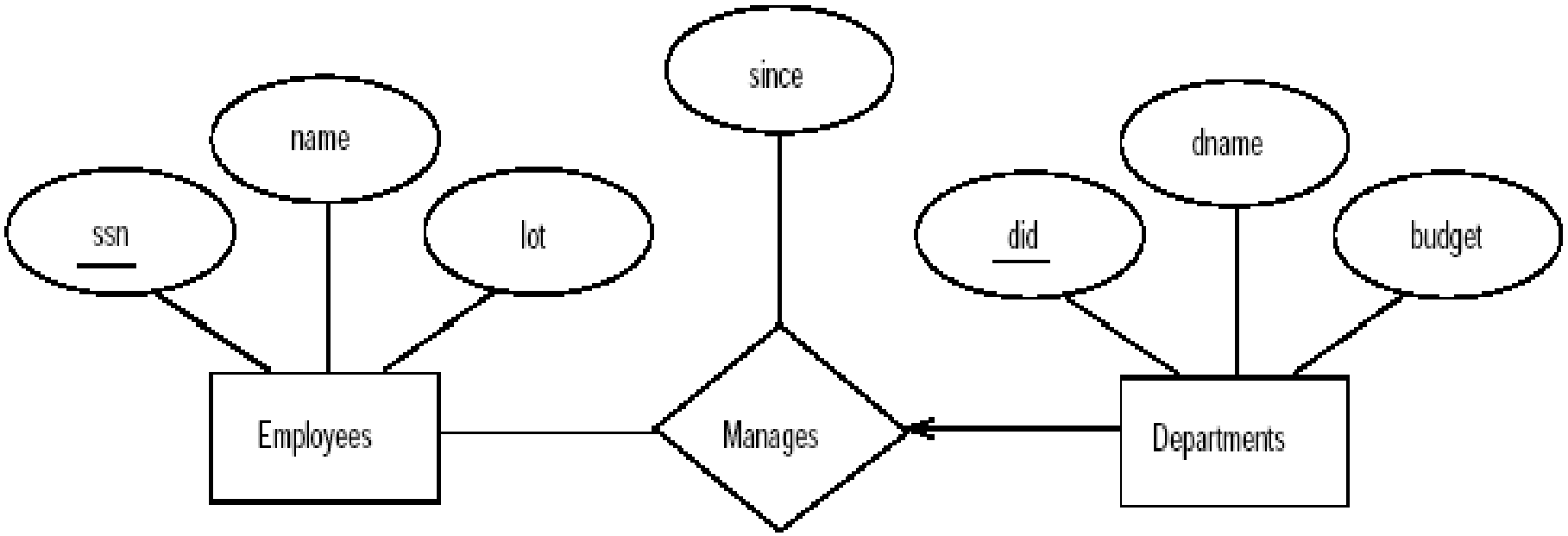


Figure 3.12 Key Constraint on Manages

First Method

```
CREATE TABLE Manages
(
  did NUMBER(2),
  ssn VARCHAR2(11),
  since DATE,
  PRIMARY KEY(did),
  FOREIGN KEY(did) REFERENCES Departments(did),
  FOREIGN KEY(ssn) REFERENCES Employees(ssn)
)
```

Second Method

```
CREATE TABLE Dept_Mgr
(
 did NUMBER(2) PRIMARY KEY,
```

```
 dname VARCHAR2(20),
 budget NUMBER(10,2),
 since DATE,
 ssn VARCHAR2(11),
 FOREIGN KEY(ssn) REFERENCES Employees(ssn)
)
```

- Second method eliminates the need for a separate Manages relation

Translating Relationship Sets with Participation Constraints

- Consider the ER diagram in Figure 3.13, which shows two relationship sets, Manages and Works_In

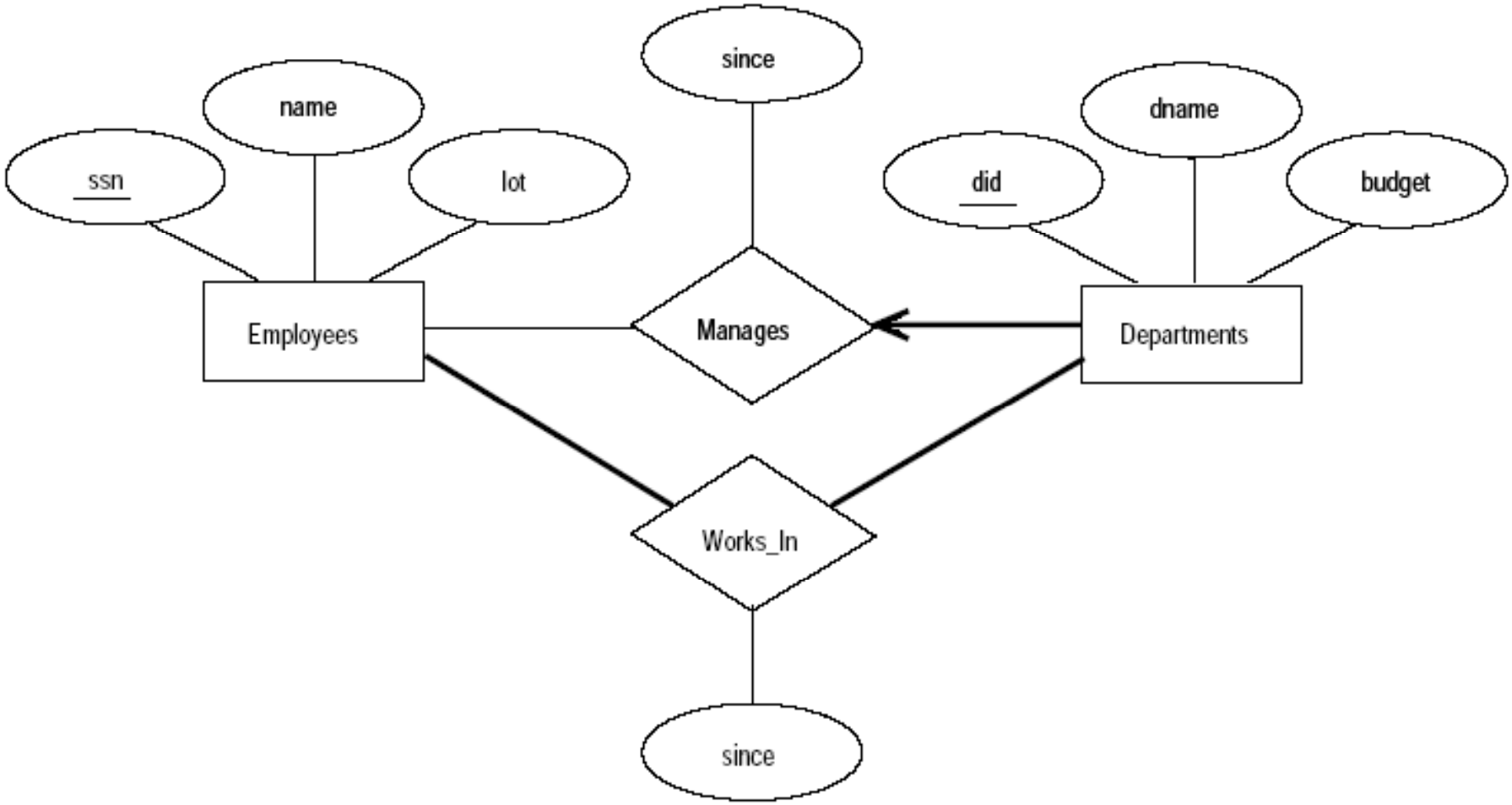


Figure 3.13 Manages and Works_In

```
CREATE TABLE Dept_Mgr
(
 did NUMBER(2) PRIMARY KEY,
 dname VARCHAR2(20),
 budget NUMBER(10,2),
 since DATE,
 ssn VARCHAR2(11) NOT NULL,
 FOREIGN KEY(ssn) REFERENCES Employees(ssn) ON
 DELETE NO ACTION
)
```

Translating Weak Entity Sets

- Consider the Dependents weak entity set shown in Figure 3.14, with partial key *pname*

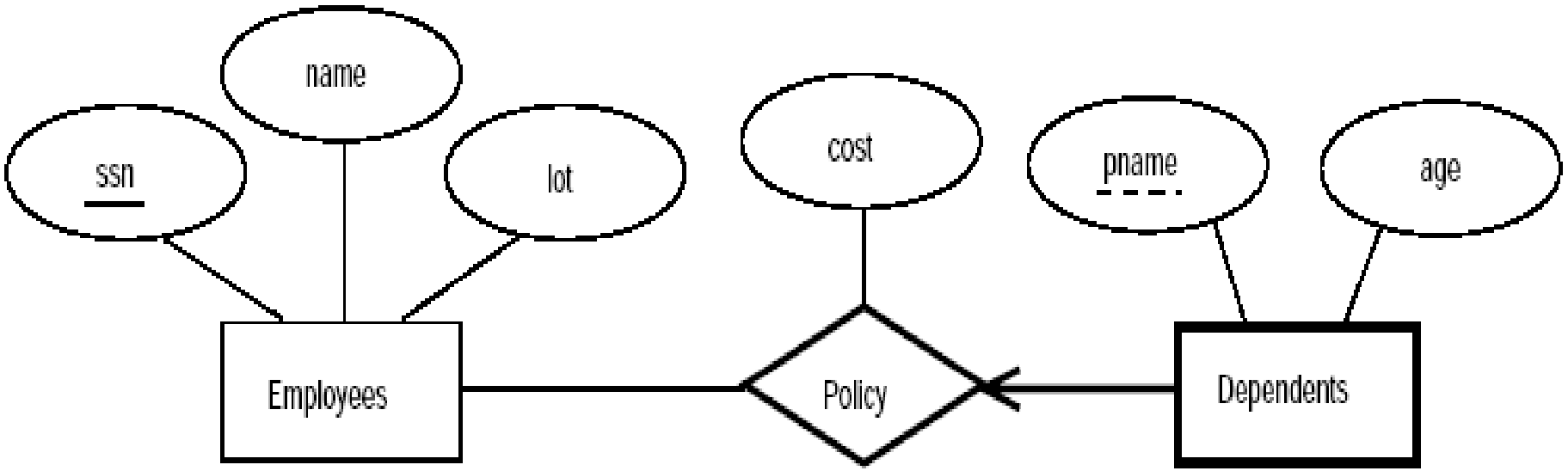


Figure 3.14 The Dependents Weak Entity Set

```
CREATE TABLE Dep_Policy
(
  pname VARCHAR2(20),
  age NUMBER(2),
  cost NUMBER(10,2),
  ssn VARCHAR2(11),
  PRIMARY KEY(ssn,pname)
  FOREIGN KEY(ssn) REFERENCES Employees(ssn) ON
  DELETE CASCADE
)
```

Translating Class Hierarchies

- The two basic approaches to handling ISA hierarchies by applying them to the ER diagram shown in Figure 3.15

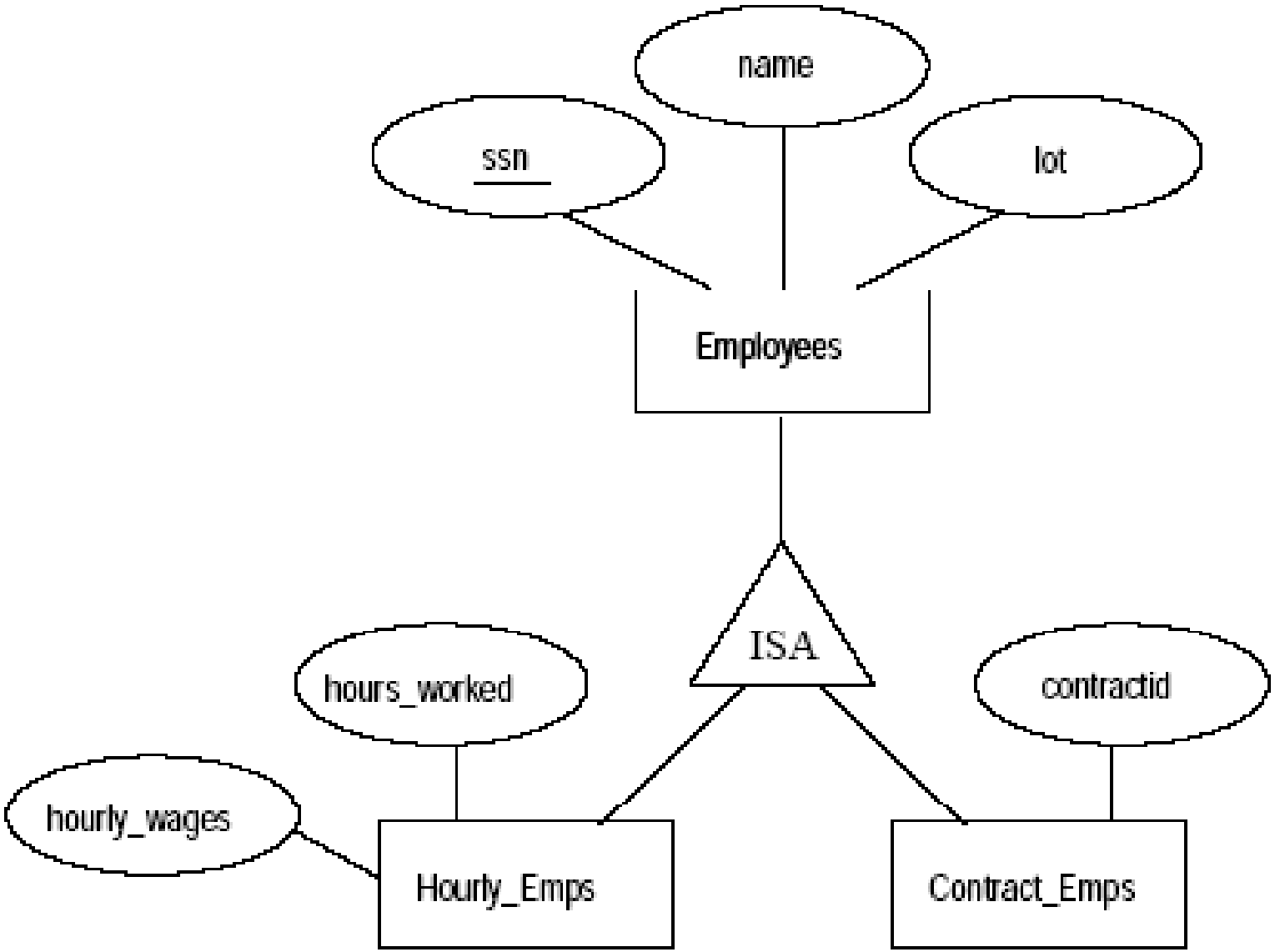


Figure 3.15 Class Hierarchy

Approach1:

- Employees relation contain fields *ssn*, *name* and *lot* with *ssn* as primary key

- Hourly_Emps relation has fileds *hourly_wages*, *hours_worked* and *ssn* with *ssn* as *primary key* as well as *foreign key* referencing the Emplyoyees

- Contract_Emps relation has fileds *contractid* and *ssn* with *ssn* as primary key as well as *foreign key* referencing the Emplyoyees

Approach2:

- Hourly_Emps includes all the attributes of Hourly_Emps as well as all the attributes of Employees (i.e., *ssn*, *name*, *lot*, *hourly_wages*, *hours_worked* with *ssn* as primary key as well as *foreign key* referencing the Emplyoyees)

- Contract_Emps includes all the attributes of Contract_Emps as well as all the attributes of Employees (i.e., *ssn*, *name*, *lot*, *contractid* with *ssn* as primary key as well as *foreign key* referencing the Emplyoyees)

Translating ER Diagrams with Aggregation

- Consider the ER diagram shown in Figure 3.16

- The Employees relation contain fields *ssn*, *name* and *lot* with *ssn* as *primary key*

- The Projects relation contain fields *pid*, *started_on* and *pbudget* with *pid* as *primary key*

- The Departments relation contain fields *did*, *dname* and *budget* with *did* as *primary key*

- The Sponsors relation contain fields *did*, *pid* and *since* with (*pid*,*did*) as *primary key* and *pid* as *foreign key* referencing Projects and *did* as *foreign key* referencing Departments

- The Monitors relation contain attributes *ssn*, *did*, *pid* and *until* with (*ssn*,*pid*,*did*) as *primary key* and *pid* as *foreign key* referencing Projects and *did* as *foreign key* referencing Departments and *ssn* as *foreign key* referencing Employees

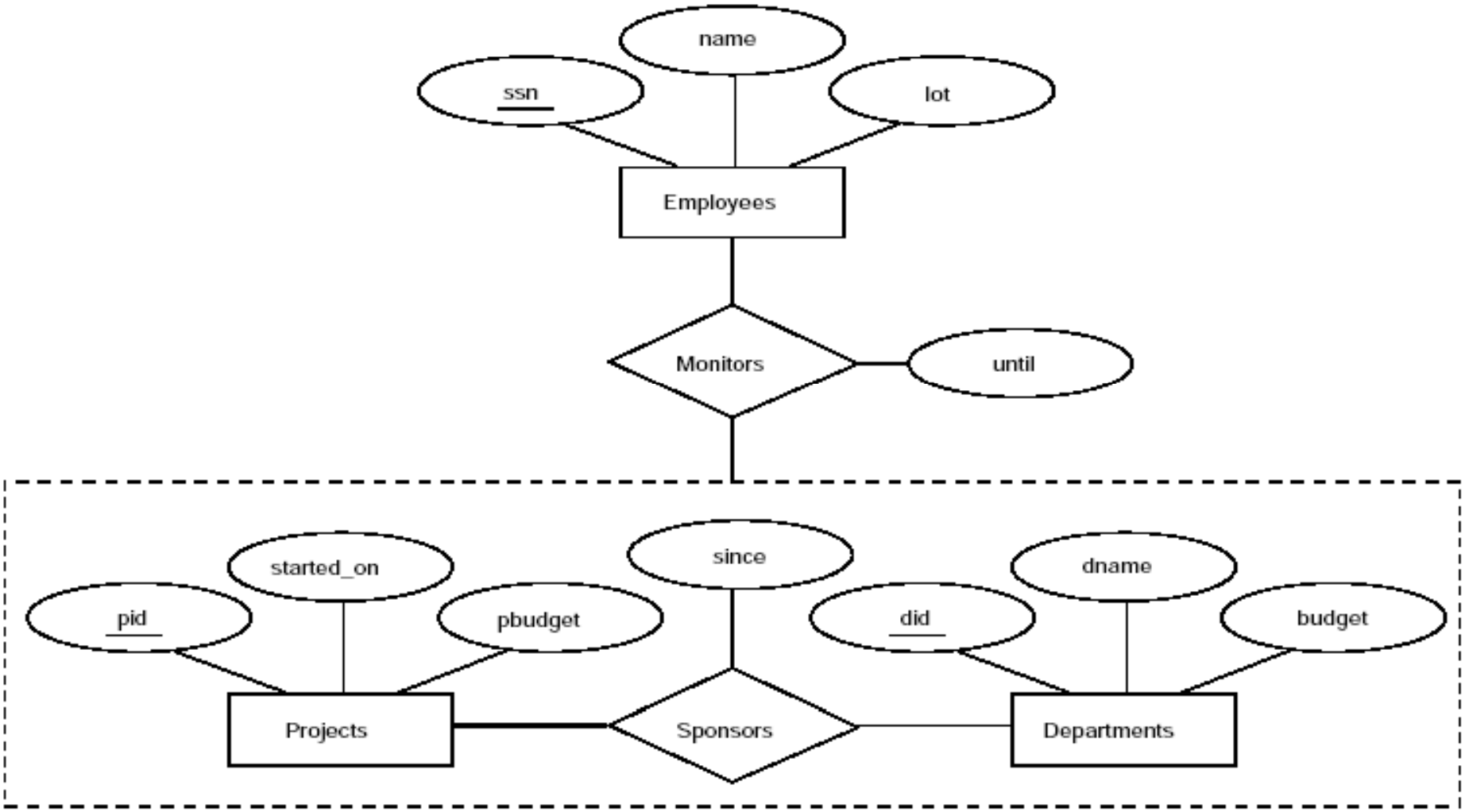


Figure 3.16 Aggregation

INTRODUCTION TO VIEWS

- A **view** is a virtual table whose rows are not explicitly stored in the database

- A view contains no data of it's own

Syntax: **create view *v as <query expression>***

Ex: CREATE VIEW Sailorsv(sid, sname)
AS SELECT sid, sname FROM Sailors

- The Sailorsv has two fields called *sid* and *sname* with the same domains as the fields *sid* and *sname* in *Sailors* table

▪This view can be used just like a **base table** in defining new queries or views

Ex: Retrieve all records from the sailorsv

Query: select * from sailorsv

Output:

SID	SNAME
32	Andy
85	Art
95	Bob
29	Brutus
22	Dustin
64	Horatio
74	Horatio
31	Lubber
58	Rusty
71	Zorba

- Although views are useful, they present serious problems if we express **updates, insertions, or deletions** with them.

- The difficulty is that a **modification** to the database expressed in terms of a **view** must be translated to a **modification** to the **actual relations** in the database.

Views, Data Independence, Security

- The ***physical schema*** for a relational database describes how the relations in the conceptual schema are stored, in terms of the file organizations and indexes used

- The view mechanism provides the support for ***logical data independence*** in the relational model

- For example, if the schema of a stored relation is changed, we can define a view with the old schema, and applications that expect to see the old schema can now use this view

- Views are also valuable in the context of ***security***

- We can define views that give a group of users access to just the information they are allowed to see

- For example, we can define a view that allows students to access other students name and age but not their gpa, and not allowed to access underlying Students table

RELATIONAL ALGEBRA

- Queries in algebra are composed using a collection of operators

- Basic operators (selection, projection, union, cross-product, and difference)

Selection and Projection

- The selection operator (σ) is used to specify the condition
- Consider the instance of the Sailors relation shown in Figure 4.2, denoted as *S2*

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Figure 4.1 Instance $S1$ of Sailors

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance $S2$ of Sailors

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

Figure 4.3 Instance $R1$ of Reserves

- Ex: Retrieve sailors whose rating is above 8

$$\sigma_{rating>8}(S2)$$

- evaluates to the relation shown in Figure 4.4

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
58	Rusty	10	35.0

Figure 4.4 $\sigma_{rating>8}(S2)$

- The projection operator (π) is used to specify the fields of a relation

- Ex: Find sailor names and ratings

$$\pi_{sname,rating}(S2)$$

- evaluates to the relation shown in Figure 4.5

sname	*rating*
yuppy	9
Lubber	8
guppy	5
Rusty	10

Figure 4.5 $\pi_{sname,rating}(S2)$

- Ex: Find ages of sailors

$$\pi_{age}(S2)$$

- evaluates to the relation shown in Figure 4.6

age
35.0
55.5

Figure 4.6 $\pi_{age}(S2)$

- Ex: Find sailor names and ratings, whose rating is above 8

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

- produces the result shown in Figure 4.7

sname	*rating*
yuppy	9
Rusty	10

Figure 4.7 $\pi_{sname,rating}(\sigma_{rating>8}(S2))$

Set Operations

- Union
- Intersection
- Set-difference
- Cross-product

Union Operation($\cup$)

- $R \cup S$ returns a relation instance containing all tuples that occur in either relation instance R or relation instance S (or both)

- The relations R and S must be compatible

Intersection Operation($\cap$)

- $R \cap S$ returns a relation instance containing all tuples that occur in both R and S

- The relations R and S must be compatible

Set-difference Operation(—)

- $R - S$ returns a relation instance containing all tuples that occur in R but not in S

- The relations R and S must be compatible

Cross-product Operation($\times$)

- $R \times S$ returns a relation instance contains all the fields of R (in the same order as they appear in R) followed by all the fields of S (in the same order as they appear in S)

- The cross-product operation is also called Cartesian product

Examples

1. Find the sid's who are present in S1 and S2

$\pi_{sid}(S1) \cup \pi_{sid}(S2)$

result:

sid
22
28
31
44
58

2. Find the sid's who are present in both S1 and S2

$$\pi_{sid}(S1) \cap \pi_{sid}(S2)$$

result:

sid
31
58

3. Find the sid's who are present in S1 but not in S2

$$\pi_{sid}(S1) - \pi_{sid}(S2)$$

result:

sid
22

4. The result of the cross-product $S1 \times R1$ is shown in Figure 4.11

(*sid*)	*sname*	*rating*	*age*	(*sid*)	*bid*	*day*
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

Figure 4.11 $S1 \times R1$

Renaming

- Field name conflicts can arise in some cases; for example, sid in $S1 \times R1$

- renaming operator(ρ) is used to rename the field

- No two fields in the result table must have the same name

- For example, the expression $\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$ returns a relation that contains the tuples shown in Figure 4.11 and has the following schema:

C(sid1:integer, sname: string, rating: integer, age: real, sid2: integer, bid: integer, day: dates)

- Note: **Renaming** is also used for naming intermediate relations

Joins

- Joins are used to combine two or more relations

Condition Joins

- Join operation accepts a join condition c and a pair of relations as arguments, and returns a relation

- The operation is defined as follows:

$$R \bowtie_c S = \sigma_c(R \times S)$$

- Ex: $S1 \bowtie_{S1.sid<R1.sid} R1$ is shown in Figure 4.12

(sid)	sname	rating	age	(sid)	bid	day
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

Figure 4.12 $S1 \bowtie_{S1.sid<R1.sid} R1$

Equijoin

- In Equijoin, condition consists of equalities of the form R.name1 = S.name2, that is, equalities between two fields in R and S

- The result $S1 \bowtie_{R.sid=S.sid} R1$ is shown in Figure 4.13

sid	*sname*	*rating*	*age*	*bid*	*day*
22	Dustin	7	45.0	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

Figure 4.13 $S1 \bowtie_{R.sid=S.sid} R1$

- Notice that only one *sid* appears in the result

Natural Join

- The equijoin expression $S1 \bowtie_{R.sid=S.sid} R1$ is actually a **natural join** and can simply be denoted as $S1 \bowtie R1$

- Relations must have at least one common attribute

▪Note: left outer join is denoted as ⟕ right outer join is denoted as ⟖ full outer join is denoted as ⟗

Division

▪Consider two relations A and B in which A has (exactly) two fields x and y and B has just one field y, with the same domain as in A

▪We define the division operation A/B as the set of all x values such that for every y value in B, there is a tuple $\langle x, y \rangle$ in A

A

sno	*pno*
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

Figure 4.14 Examples Illustrating Division

Examples of Relational Algebra Queries

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

(Q1) Find the names of sailors who have reserved boat 103

$$\pi_{sname}((\sigma_{bid=103}Reserves) \bowtie Sailors)$$

Result: *Dustin*, *Lubber* and *Horatio*

(OR)

- Using renaming operator

$$\rho(Temp1, \sigma_{bid=103}Reserves)$$

$$\rho(Temp2, Temp1 \bowtie Sailors)$$

$$\pi_{sname}(Temp2)$$

sid	*bid*	*day*
22	103	10/8/98
31	103	11/6/98
74	103	9/8/98

Figure 4.18 Instance of *Temp1*

sid	*sname*	*rating*	*age*	*bid*	*day*
22	Dustin	7	45.0	103	10/8/98
31	Lubber	8	55.5	103	11/6/98
74	Horatio	9	35.0	103	9/8/98

Figure 4.19 Instance of *Temp2*

- Here is another way to write this query:

$$\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$$

(Q2) Find the names of sailors who have reserved a red boat

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)$$

Result: *Dustin*, *Lubber* and *Horatio*

(Q3) Find the colors of boats reserved by Lubber

$$\pi_{color}((\sigma_{sname='Lubber'} Sailors) \bowtie Reserves \bowtie Boats)$$

(Q4) Find the names of sailors who have reserved at least one boat

$$\pi_{sname}(Sailors \bowtie Reserves)$$

(Q5) Find the names of sailors who have reserved a red or a green boat

$$\rho(Tempboats, (\sigma_{color='red'} Boats) \cup (\sigma_{color='green'} Boats))$$

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

(Q6) Find the sids of sailors with age over 20 who have not reserved a red boat

$$\pi_{sid}(\sigma_{age>20}Sailors) -$$

$$\pi_{sid}((\sigma_{color='red'}Boats) \bowtie Reserves \bowtie Sailors)$$

(Q7) Find the names of sailors who have reserved all boats

$$\rho(Tempsids, (\pi_{sid,bid}Reserves)/(\pi_{bid}Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

(Q8) Find the names of sailors who have reserved all boats called Interlake

$$\rho(Tempsids, (\pi_{sid,bid}Reserves)/(\pi_{bid}(\sigma_{bname='Interlake'}Boats)))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

RELATIONAL CALCULUS

- Relational calculus is an alternative to relational algebra
- Relational Algebra is procedural language
- Relational Calculus is nonprocedural language
- Relational calculus is of two types

1. Tuple Relational Calculus(TRC)
2. Domain Relational Calculus(DRC)

Tuple Relational Calculus

- A **tuple** variable is a variable that takes tuples of a relation as values

- A tuple relational calculus query has the form $\{ T \mid p(T) \}$ where ***T*** is a tuple variable and ***p(T)*** **is a formula** that describes *T*

- The result of this query is the set of all tuples *t* for which the formula *p(T)* evaluates to true with $T = t$

Ex:

(Q) Find all sailors with a rating above 7

$$\{S \mid S \in Sailors \wedge S.rating > 7\}$$

Syntax of TRC Queries

- Let ***Rel*** be a relation name, ***R*** and ***S*** be tuple variables, ***a*** an attribute of ***R***, and ***b*** an attribute of ***S***

- Let ***op*** denote an operator in the set $\{<, >, =, \leq, \geq, \neq\}$

- **Formula** contains the following:
  - $R \in Rel$
  - $R.a$ op $S.b$
  - $R.a$ op $constant$, or $constant$ op $R.a$

- any atomic formula
- $\neg p$, $p \wedge q$, $p \vee q$, or $p \Rightarrow q$
- $\exists R(p(R))$, where R is a tuple variable
- $\forall R(p(R))$, where R is a tuple variable

▪In the last two clauses above, the **quantifiers** $\exists$ and $\forall$ are said to **bind** the variable *R*

- A variable is said to be **free** in a formula if the formula does not contain an occurrence of a quantifier that binds it

- A **TRC query** is defined to be expression of the form $\{ T \mid p(T) \}$,where *T* is the only free variable in the formula *p*

Examples of TRC Queries

- Consider the instances *B1* of Boats, *R2* of Reserves, and *S3* of Sailors shown in Figures 4.15, 4.16, and 4.17

(Q)Find the names and ages of sailors with a rating above 7

$$\{P \mid \exists S \in Sailors(S.rating > 7 \wedge P.name = S.sname \wedge P.age = S.age)\}$$

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

(Q) Find the sailor name, boat id, and reservation date for each reservation

$$\{P \mid \exists R \in Reserves \;\; \exists S \in Sailors$$

$$(R.sid = S.sid \wedge P.bid = R.bid \wedge P.day = R.day \wedge P.sname = S.sname)\}$$

sname	*bid*	*day*
Dustin	101	10/10/98
Dustin	102	10/10/98
Dustin	103	10/8/98
Dustin	104	10/7/98
Lubber	102	11/10/98
Lubber	103	11/6/98
Lubber	104	11/12/98
Horatio	101	9/5/98
Horatio	102	9/8/98
Horatio	103	9/8/98

Figure 4.20 Answer to Query Q

(Q) Find the names of sailors who have reserved boat 103

$$\{P \mid \exists S \in Sailors\ \exists R \in Reserves(R.sid = S.sid \wedge R.bid = 103 \wedge P.sname = S.sname)\}$$

(Q) Find the names of sailors who have reserved a red boat

$$\{P \mid \exists S \in Sailors\ \ \exists R \in Reserves\ \ \exists B \in Boats$$
$$(R.sid = S.sid \wedge B.bid = R.bid \wedge B.color = 'red' \wedge P.sname = S.sname)\}$$

(Q) Find the names of sailors who have reserved all boats

$$\{P \mid \exists S \in Sailors\ \ \forall B \in Boats$$
$$(\exists R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid \wedge P.sname = S.sname))\}$$

(Q) Find sailors who have reserved all red boats

$$\{S \mid S \in Sailors \wedge \forall B \in Boats$$
$$(B.color = 'red' \Rightarrow (\exists R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid)))\}$$

Domain Relational Calculus

- A **domain variable** is a variable that ranges over the values in the domain of some attribute

- Ex: the variable can be assigned an integer if it appears in an attribute whose domain is the set of integers

- A DRC query has the form

$$\{\langle x_1, x_2, \ldots, x_n \rangle \mid p(\langle x_1, x_2, \ldots, x_n \rangle)\}$$

- where each x_i is either a *domain variable* or a constant and $p(\langle x_1, x_2, \ldots, x_n \rangle)\}$ denotes a **DRC formula**

- The result of this query is the set of all tuples $\langle x_1, x_2, \ldots, x_n \rangle$ for which the formula evaluates to true

- A DRC formula is defined similar to the definition of a TRC formula

- The main difference is that the variables are now domain variables

- Let *op* denote an operator in the set $\{<, >, =, \leq, \geq, \neq\}$ and let *X* and *Y* be domain variables

- **Formula** contains the following:

- $\langle x_1, x_2, \ldots, x_n \rangle \in Rel$, where Rel is a relation with n attributes; each x_i, $1 \leq i \leq n$ is either a variable or a constant.
- X op Y
- X op *constant*, or *constant* op X
- any atomic formula
- $\neg p$, $p \wedge q$, $p \vee q$, or $p \Rightarrow q$
- $\exists X(p(X))$, where X is a domain variable
- $\forall X(p(X))$, where X is a domain variable

Examples of DRC Queries

(Q) Find all sailors with a rating above 7

$$\{\langle I, N, T, A\rangle \mid \langle I, N, T, A\rangle \in Sailors \wedge T > 7\}$$

(Q) Find the names of sailors with a rating above 7

$$\{\langle N\rangle \mid \exists I, T, A(\langle I, N, T, A\rangle \in Sailors \wedge T > 7\}$$

(Q) Find the names of sailors who have reserved boat 103

$$\{\langle N\rangle \mid \exists I, T, A(\langle I, N, T, A\rangle \in Sailors$$
$$\wedge \exists Ir, Br, D(\langle Ir, Br, D\rangle \in Reserves \wedge Ir = I \wedge Br = 103))\}$$

(Or)

$$\{\langle N \rangle \mid \exists I, T, A(\langle I, N, T, A \rangle \in Sailors \wedge \exists D(\langle I, 103, D \rangle \in Reserves))\}$$