Module 2: The Relational Data Model

Relational Model Concepts

The principles of the relational model were first outlined by Dr. E.F. Codd in 1970 in a classic paper called "A relational Model of Data for Large Shared Data Banks". In this paper, Dr. Codd proposed the relational model for the database.

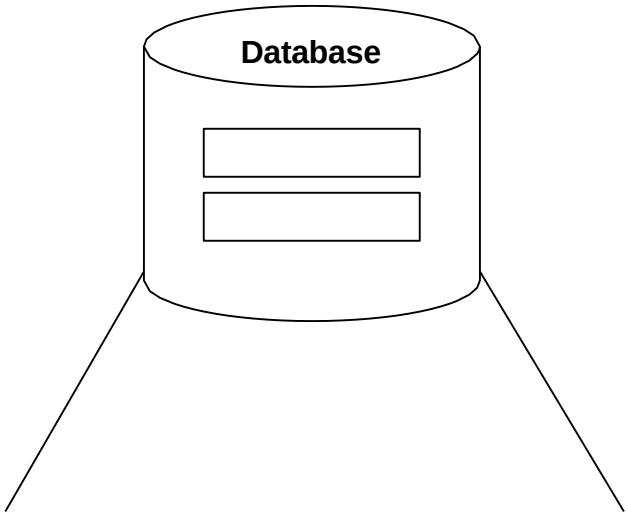
The more popular models used at the time were hierarchical and network, or even simple flat file data structures. Relational database management systems soon became very popular especially for their ease of use and flexibility in structure. In addition, a number of innovative vendors, such as Oracle, supplemented the RDBMS with a suite of powerful application development and user products, providing a total solution.

Components of the Relational Model

- Collections of objects or relations that store the data
- A set of operators that can act on the relations to produce other relations
- Data integrity for accuracy and consistency

Definition of a Relational Database

A relational database is a collection of relations or two-dimensional tables.



EMP

EMPNO	ENAME	DEPTNO
7839	King	10

DEPT

DEPTNO	DNAME	LOC
10	Accounting	New York
20	Research	Dallas

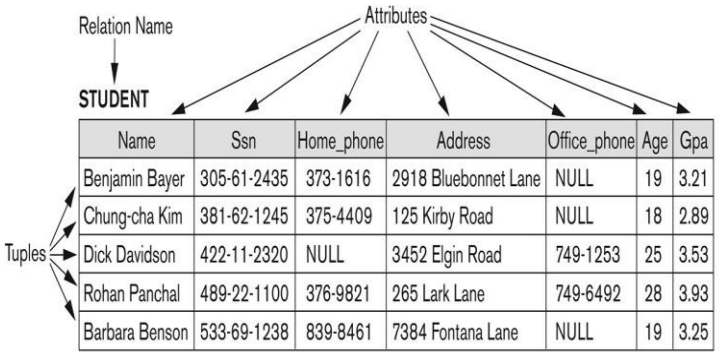
A relational database is a collection of relations or two-dimensional tables to store information.

For Example, you might want to store information about all the employees in your company. In a relational database, you create several tables to store different pieces of information about your employees, such as an employee table, a department table and a salary table

Informal Definitions

- Informally, a relation looks like a table of values.
- A relation typically contains a set of rows.
- The data elements in each row represent certain facts that correspond to a real-world entity or relationship.
    - In the formal model, rows are called tuples
- Each column has a column header that gives an indication of the meaning of the data items in that column.
    - In the formal model, the column header is called an attribute name (or just attribute).

Example of a Relation



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

- The data type describing the types of values that can appear in each columns is called a **domain**.

- Key of a Relation:
    - Each row has a value of a data item (or set of items) that uniquely identifies that row in the table called the **key**
      Eg:- In the STUDENT table, SSN is the key.

    - Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table called *artificial key* or *surrogate key*

Formal Definitions – Schema

- The **Schema** (or description) of a Relation:
  - Denoted by R(A_1, A_2,.....A_n)
  - R is the **name** of the relation
  - The **attributes** of the relation are A_1, A_2, ... A_n
  - The degree of the relation is n (the number of its attributes).
- Example:

    CUSTOMER (Cust-id, Cust-name, Address, Phone#)

  - CUSTOMER is the relation name.
  - Defined over the four attributes:Cust-id, Cust-name, Address, Phone#

- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.
- The **Schema** (or description) of a Relation:
  - Denoted by R(A_1, A_2,.....A_n)
  - R is the **name** of the relation
  - The **attributes** of the relation are A_1, A_2, ... A_n
  - The degree of the relation is n (the number of its attributes).
- Example:

    CUSTOMER (Cust-id, Cust-name, Address, Phone#)

  - CUSTOMER is the relation name.
  - Defined over the four attributes:Cust-id, Cust-name, Address, Phone#
- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.

- A **tuple** is an ordered set of values (enclosed in angled brackets '< … >').
- Each value is derived from an appropriate *domain*.
- A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
  - <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
  - This is called a 4-tuple as it has 4 values.
  - A tuple (row) in the CUSTOMER relation.
- A relation is a **set** of such tuples (rows).
- A **domain** has a logical definition:
  - Example: "University_Seat_Number" is the set of 10 characters valid in VTU university.
  - "Academic_Dept_Code" is a set of academic department codes such as 'CS', 'IS', 'EC', etc,.
- A domain also has a data-type or a format defined for it.
  - Dates have various formats such as year, month, day formatted as yyyy-mm-dd, or as dd-mmm-yyyy etc.
- The attribute name designates the role played by a domain in a relation:
  - Example: The domain Date may be used to define two attributes named "Invoice-date" and "Payment-date" with different meanings.

Characteristics of Relations

- **Ordering of tuples in a relation r(R):**
  - Tuples ordering is not part of a relation definition because it is defined as a set of tuples.
    - Many logical orders can be specified on the relation.
    - There is no preference for one logical ordering over another.
  - When a relation is implemented as a file, a physical ordering may be specified on the records of the file.
- **Ordering of values within each tuple:**
  - The tuple is an ordered list, so the ordering of values in a tuple is important.
  - At a logical level, the order is not important as long as the correspondence between the attribute and its value is mentioned.

Figure 5.2
The relation STUDENT from Figure 5.1 with a different order of tuples.

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21

- **Values and NULLs in the tuple:**
  - All values are considered atomic (indivisible).
  - Each value in a tuple must be from the domain of the attribute for that column.
    - If tuple t = $<v_1, v_2, \ldots, v_n>$ is a tuple (row) in the relation state r of $R(A_1, A_2, \ldots, A_n)$
    - Then each v_i must be a value from $dom(A_i)$
  - A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.
- **Interpretation (Meaning) of a Relation:**

  - The relational schema can be interpreted as a declaration or type of assertion.

  - For example, schema of the STUDENT relation interprets that, a student entity has Name,

    USN, Home_phone, Address, Office_phone, Age, Gpa.

  - Alternative interpretation of a relation schema is as a **predicate,** that values in each tuple are interpreted as values that satisfy the predicate.

Relational Model Notation

- $R(A_1, A_2, ..., A_n)$ denotes a relation schema R of degree n (n attributes).
- r(R) is a relation state of relation schema R.
- $t[A_i]$ and $t.A_i$ refer to the value v_i in t for attribute A_i.
- The letters Q, R, S denote relation names.
- The letters q, r, s denote relation states.
- The letters t, u, v denote tuples.
- The dot notation R.A can be used to identify the attributes (e.g. Student.Name or Student.Age).

Relational model constraints

- Constraints on databases can generally be divided into three main categories:

- Inherent model-based or Implicit constraints:- Constraints that are inherent in the data model.

- Schema-based or explicit constraints:- Constraints that can be directly expressed in schema of the data model using DDL.

- Application-based or semantic constraints or business rules:- Constraints expressed and enforced using application programs.

Schema-based constraints

- Schema based constraints in the relational model are:
  - **Domain** constraints
  - **Key** constraints and constraints on NULL values
  - **Entity** integrity constraints
  - **Referential** integrity constraints
- **Domain constraint**
  - Every value in a tuple must be from the *domain of its attribute* and must be atomic (or it could be **null**, if allowed for that attribute).
- **Key Constraints**
  - As a relation is defined as a set of tuple, thus no two tuples can have the same combination of values for all their attributes.
  - Usually, there are other subset of attributes (superkey SK) with this constraint: $t_i[SK] \neq t_j[SK]$ 6 i, j
  - It is called uniqueness constraint that no two distinct tuples in r can have the same values for SK.
  - A superkey can have redundant attributes.
  - A key K of R is a superkey of R with the additional property that removing any attribute A from K leaves a set of attributes K′ that is not a superkey of R.
  - A key satisfies two constraints:
  - - Two distinct tuples in any state of the relation cannot have identical values for (all) attributes in the key.

- - It is a ***minimal superkey*** – that is , a superkey from which we cannot remove any attributes and still have the uniqueness property in condition 1 hold.
- Key is determined from the meaning of the attributes, and the property is ***time-invariant***.
- It must continue to hold when we insert new tuples in the relation
- Example:
- The attribute {SSN} in the Student relation is a key.
- Any set includes {SSN} is a superkey of the relation student. Eg:- {SSN, Name, Age}
- In general:
- Any *key* is a *superkey* (but not vice versa).
- Any set of attributes that *includes a key* is a *superkey.*
- A *minimal* superkey is also a key.
- A relation schema may have more than one key.
- Such keys are called the candidate keys.
- The primary key is one of the candidate keys which is selected to identify tuples in the relation.
- The attributes that form the primary key are underlined in the schema.
- The primary key value is used to *uniquely identify* each tuple in a relation.

- **Null Constraint:**
  - Not Null constraint specifies that an attribute must have a valid value (e.g. Student name).

Relational Database Schema

- A relational database schema S is a set of relation schemas that belong to the same database.
  - S is the name of the whole **database schema.**
  - S = {R_1, R_2, …, R_m} and a set of integrity constraints IC.
  - R_1, R_2, …, R_m are the names of the individual **relation schemas** within the database S.
- The integrity constraints are specified on a DB schema and are expected to hold on every DB state.
- Example: 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 5.5
Schema diagram for the COMPANY relational database schema.

Entity Integrity Constraints

- The primary key attributes PK of each relation schema R in S cannot have null values in any tuple of r(R).
  - This is because primary key values are used to *identify* the individual tuples.
  - t[PK] s null for any tuple t in r(R).
  - If PK has several attributes, null is not allowed in any of these attributes.
- Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity Constraints

- A constraint involving two relations
  - The previous constraints involve a single relation.
- Used to specify a relationship among tuples in two relations:
  - The referencing relation and the referenced relation.
- Tuples in the referencing relation R_1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R_2.
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R_1.FK to R_2.PK.
- Referential integrity constraint is based on foreign key (FK) concept
- A set of attributes FK in relation schema R_1 is a foreign key of R_1 that references relation R_2 if it satisfies the following two rules:
  - The attributes in FK have the same domain(s) as the primary key attribute PK of R_2.
  - A value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ ($t_1[FK]=t_2[PK]$) or is null.
- A foreign key can refer to its own relation.
- Referential integrity constraints typically arise from the *relationship among the entities*.

Other Types of Constraints

- **State constraints**

  - define the constraints that a valid state of the database must satisfy.

  Ex: Domain constraints, Key Constraints, Entity Integrity constraints, Referential Integrity constraints

Transition constraints

- defined to deal with state changes in the database.

Ex: the salary of an employee can only increase. Such constraints typically enforced by the application programs or specified using active rules and triggers.

Update Operations on Relations

- The basic update operations of the relational model are:
  - Insert
  - Delete
  - Update (or Modify)
- All integrity constraints specified on the database schema should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may propagate to cause other updates automatically.
  - This may be necessary to maintain integrity constraints.
- The basic update operations of the relational model are:
  - Insert
  - Delete
  - Update (or Modify)
- All integrity constraints specified on the database schema should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may propagate to cause other updates automatically.
  - This may be necessary to maintain integrity constraints.

The Insert Operation

- INSERT may violate any of the four types of constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain.
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation.
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does

not exist in the referenced relation.

- Entity integrity:
  - if the primary key value is null in the new tuple.
- Example:-
- Operation

Insert <'Vinod', 'S', 'Joseph', NULL, '1986-04-05', '#123, Bangalore', F,

28000, NULL, 4> into EMPLOYEE

Result: Violates Entity IC , so it is rejected.

- Operation

Insert <'Alice', 'J', 'Zelaya', '999887777', '1986-04-05', '#334, Bangalore', F,

28000, '98764321', 4> into EMPLOYEE

Result: Violates Key IC , so it is rejected.

- Operation

Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1986-04-05', '#454, Bangalore',

F, 28000, '99876436', 7> into EMPLOYEE

Result: Violates Referential IC , so it is rejected

The Delete Operation

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database.
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL .
      - RESTRICT option: reject the deletion.
      - CASCADE option: propagate deletion by deleting tuples that reference the tuple that is deleted.
      - SET NULL option: set the foreign keys of the referencing tuples to NULL.
  - One of the above options must be specified during database design for each foreign key constraint.
  - Example:
    - **Operation**

      Delete the WORKS_ON tuple with ESSN-'999887777' and Pno=10.

      Result: Acceptable. Deletes exactly one tuple

    - **Operation**

      Delete the EMPLOYEE tuple with SSN= '999887777'.

      Result: Not Acceptable. Violates Referential IC.

- **Operation**
  Delete the EMPLOYEE tuple with SSN= '333445555'.

  Result: Not Acceptable. Violates Referential IC.

The Update Operation

- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified.
- Any of the other constraints may also be violated, depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT.
    - Need to specify similar options to DELETE.
  - Updating a foreign key (FK):
    - May violate referential integrity.
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain constraints.

Operation

- Update the salary of an EMPLOYEE tuple with SSN ='999887777' to 28000.

Result: Acceptable

Operation

- Update the Dno of the EMPLOYEE tuple with SSN='999887777' to 7.
  Result: Violates Referential IC

- Operation
  Update the SSN of the EMPLOYEE tuple with SSN='999887777' to '987654321'.

Result: Violates Primary key constraints

Relational Algebra

- Procedural language
- Queries in relational algebra are applied to relation instances, result of a query is again a relation instance
- Six basic operators in relational algebra:

select	σ	selects a subset of tuples from reln
project	π	deletes unwanted columns from reln
Cartesian Product	$\times$	allows to combine two relations
Set-difference	$-$	tuples in reln. 1, but not in reln. 2
Union	$\cup$	tuples in reln 1 plus tuples in reln 2
Rename	ρ	renames attribute(s) and relation

- The operators take one or two relations as input and give a new relation as a result (relational algebra is "closed").

Select Operation

- Notation: $\sigma_P(r)$

  Defined as

  $$\sigma_P(r) := \{t \mid t \in r \text{ and } P(t)\}$$

  where
  - r is a relation (name),
  - P is a formula in propositional calculus, composed of conditions of the form

    $$<\text{attribute}> = <\text{attribute}> \text{ or } <\text{constant}>$$

    Instead of "$=$" any other comparison predicate is allowed ($\neq, <, >$ etc).
    Conditions can be composed through $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

- Example: given the relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \wedge D>5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Project Operation

- Notation: $\pi_{A_1,A_2,\ldots,A_k}(r)$

  where $A_1, \ldots, A_k$ are attribute names and
  r is a relation (name).

- The result of the projection operation is defined as the relation that has k columns obtained by erasing all columns from r that are not listed.

- Duplicate rows are removed from result because relations are sets.

- Example: given the relations r

r

A	B	C
α	10	2
α	20	2
β	30	2
β	40	4

$\pi_{A,C}(r)$

A	C
α	2
β	2
β	4

Cartesian Product

- Notation: $r \times s$ where both r and s are relations

  Defined as $r \times s := \{tq \mid t \in r \textbf{ and } q \in s\}$

- Assume that attributes of $r(R)$ and $s(S)$ are disjoint, i.e., $R \cap S = \emptyset$.

  If attributes of $r(R)$ and $s(S)$ are not disjoint, then the rename operation must be applied first.

- Example: relations r, s:

r

A	B
α	1
β	2

s

C	D	E
α	10	$+$
β	10	$+$
β	20	$-$
γ	10	$-$

$r \times s$

A	B	C	D	E
α	1	α	10	$+$
α	1	β	10	$+$
α	1	β	20	$-$
α	1	γ	10	$-$
β	2	α	10	$+$
β	2	β	10	$+$
β	2	β	20	$-$
β	2	γ	10	$-$

Union Operator

- Notation: $r \cup s$ where both r and s are relations

  Defined as $r \cup s := \{t \mid t \in r \textbf{ or } t \in s\}$

- For $r \cup s$ to be applicable,

  1. r, s must have the same number of attributes
  2. Attribute domains must be compatible (e.g., 3rd column of r has a data type matching the data type of the 3rd column of s)

- Example: given the relations r and s

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cup s$

A	B
α	1
α	2
β	1
β	3

Set Difference Operator

- Notation: $r - s$ where both r and s are relations

  Defined as $r - s := \{t \mid t \in r \textbf{ and } t \notin s\}$

- For $r - s$ to be applicable,

  1. r and s must have the same arity
  2. Attribute domains must be compatible

- Example: given the relations r and s

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r - s$

A	B
α	1
β	1

Rename Operation

- Allows to name and therefore to refer to the result of relational algebra expression.
- Allows to refer to a relation by more than one name (e.g., if the same relation is used twice in a relational algebra expression).
- Example:

  $$\rho_x(E)$$

  returns the relational algebra expression E under the name x

  If a relational algebra expression E (which is a relation) has the arity k, then

  $$\rho_{x(A_1,A_2,\ldots,A_k)}(E)$$

  returns the expression E under the name x, and with the attribute names $A_1, A_2, \ldots, A_k$.

Composition of Operations

- It is possible to build relational algebra expressions using multiple operators similar to the use of arithmetic operators (nesting of operators)
- Example: $\sigma_{A=C}(r \times s)$

$r \times s$

A	B	C	D	E
α	1	α	10	+
α	1	β	10	+
α	1	β	20	−
α	1	γ	10	−
β	2	α	10	+
β	2	β	10	+
β	2	β	20	−
β	2	γ	10	−

$\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	+
β	2	β	10	+
β	2	β	20	−

Example Queries

Assume the following relations:

```
BOOKS(DocId, Title, Publisher, Year)
STUDENTS(StId, StName, Major, Age)
AUTHORS(AName, Address)
borrows(DocId, StId, Date)
has-written(DocId, AName)
describes(DocId, Keyword)
```

- *List the year and title of each book.*

  $\pi_{\text{Year, Title}}(\text{BOOKS})$

- *List all information about students whose major is CS.*

  $\sigma_{\text{Major} = \text{'CS'}}(\text{STUDENTS})$

- *List all students with the books they can borrow.*

  $\text{STUDENTS} \times \text{BOOKS}$

- *List all books published by McGraw-Hill before 1990.*

  $\sigma_{\text{Publisher} = \text{'McGraw-Hill'} \wedge \text{Year} < 1990}(\text{BOOKS})$

- *List the name of those authors who are living in Davis.*

  $\pi_{\text{AName}}(\sigma_{\text{Address like '\%Davis\%'}}(\text{AUTHORS}))$

- *List the name of students who are older than 30 and who are not studying CS.*

  $\pi_{\text{StName}}(\sigma_{\text{Age}>30}(\text{STUDENTS})) -$
  $\pi_{\text{StName}}(\sigma_{\text{Major}=\text{'CS'}}(\text{STUDENTS}))$

- *Rename* `AName` *in the relation* `AUTHORS` *to* `Name`.

  $\rho_{\text{AUTHORS(Name, Address)}}(\text{AUTHORS})$

Composed Queries (formal definition)

- A *basic expression* in the relational algebra consists of either of the following:
  - A relation in the database
  - A constant relation
    (fixed set of tuples, e.g., $\{(1, 2), (1, 3), (2, 3)\}$)

- If E_1 and E_2 are expressions of the relational algebra, then the following expressions are relational algebra expressions, too:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_P(E_1)$ where P is a predicate on attributes in E_1
  - $\pi_A(E_1)$ where A is a list of some of the attributes in E_1
  - $\rho_x(E_1)$ where x is the new name for the result relation
    [and its attributes] determined by E_1

Examples of Composed Queries

1. *List the names of all students who have borrowed a book and who are CS majors.*

   $$\pi_{\text{StName}}(\sigma_{\text{STUDENTS.StId}=\text{borrows.StId}} (\sigma_{\text{Major}=\text{'CS'}}(\text{STUDENTS}) \times \text{borrows}))$$

2. *List the title of books written by the author 'Silberschatz'.*

   $$\pi_{\text{Title}}(\sigma_{\text{AName}=\text{'Silberschatz'}} (\sigma_{\text{has-written.DocId}=\text{BOOKS.DocID}}(\text{has-written} \times \text{BOOKS})))$$

   or

   $$\pi_{\text{Title}}(\sigma_{\text{has-written.DocId}=\text{BOOKS.DocID}} (\sigma_{\text{AName}=\text{'Silberschatz'}}(\text{has-written}) \times \text{BOOKS}))$$

3. *As 2., but not books that have the keyword 'database'.*

   . . . as for 2. . . .

   $$- \ \pi_{\text{Title}}(\sigma_{\text{describes.DocId}=\text{BOOKS.DocId}} (\sigma_{\text{Keyword}=\text{'database'}}(\text{describes}) \times \text{BOOKS}))$$

4. *Find the name of the youngest student.*

   $$\pi_{\text{StName}}(\text{STUDENTS}) -$$
   $$\pi_{\text{S1.StName}}(\sigma_{\text{S1.Age}>\text{S2.Age}}(\rho_{\text{S1}}(\text{STUDENTS}) \times \rho_{\text{S2}}(\text{STUDENTS})))$$

5. *Find the title of the oldest book.*

   $$\pi_{\text{Title}}(\text{BOOKS}) -$$
   $$\pi_{\text{B1.Title}}(\sigma_{\text{B1.Year}>\text{B2.Year}}(\rho_{\text{B1}}(\text{BOOKS}) \times \rho_{\text{B2}}(\text{BOOKS})))$$

Additional Operators

These operators do not add any power (expressiveness) to the relational algebra but simplify common (often complex and lengthy) queries.

Set-Intersection	$\cap$	
Natural Join	$\bowtie$	
Condition Join	$\bowtie_C$	(also called Theta-Join)
Division	$\div$	
Assignment	$\longleftarrow$	

Set-Intersection

- Notation: $r \cap s$
  Defined as $r \cap s := \{t \mid t \in r \textbf{ and } t \in s\}$
- For $r \cap s$ to be applicable,
  1. r and s must have the same arity
  2. Attribute domains must be compatible
- Derivation: $r \cap s = r - (r - s)$
- Example: given the relations r and s

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cap s$

A	B
α	2

Natural Join

- Notation: $r \bowtie s$
- Let r, s be relations on schemas R and S, respectively. The result is a relation on schema $R \cup S$. The result tuples are obtained by considering each pair of tuples $t_r \in r$ and $t_s \in s$.
- If t_r and t_s have the same value for each of the attributes in $R \cap S$ ("same name attributes"), a tuple t is added to the result such that
  - t has the same value as t_r on r
  - t has the same value as t_s on s
- Example: Given the relations $R(A, B, C, D)$ and $S(B, D, E)$
  - Join can be applied because $R \cap S \neq \emptyset$
  - the result schema is (A, B, C, D, E)
  - and the result of $r \bowtie s$ is defined as

$$\pi_{r.A,r.B,r.C,r.D,s.E}(\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$$

- Example: given the relations r and s

r

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

s

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	τ

$r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Condition Join

- Notation: $r \bowtie_C s$

  C is a condition on attributes in $R \cup S$, result schema is the same as that of Cartesian Product. If $R \cap S \neq \emptyset$ and condition C refers to these attributes, some of these attributes must be renamed.
  Sometimes also called *Theta Join* ($r \bowtie_\theta s$).

- Derivation: $r \bowtie_C s = \sigma_C(r \times s)$

- Note that C is a condition on attributes from both r and s

- Example: given two relations r, s

r

A	B	C
1	2	3
4	5	6
7	8	9

s

D	E
3	1
6	2

$r \bowtie_{B<D} s$

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

If C involves only the comparison operator "$=$", the condition join is also called *Equi-Join.*

- Example 2:

r

A	B	C
4	5	6
7	8	9

s

C	D
6	8
10	12

$r \bowtie_{\mathsf{C=SC}} (\rho_{\mathsf{S(SC,D)}}(s))$

A	B	C	SC	D
4	5	6	6	8

Division

- Notation: $r \div s$
- Precondition: attributes in S must be a subset of attributes in R, i.e., $S \subseteq R$. Let r, s be relations on schemas R and S, respectively, where

  - $R(A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S(B_1, \ldots, B_n)$

  The result of $r \div s$ is a relation on schema $R - S = (A_1, \ldots, A_m)$

- Suited for queries that include the phrase "for all".

  The result of the division operator consists of the set of tuples from r defined over the attributes $R - S$ that match the combination of **every** tuple in s.

  $$r \div s := \{t \mid t \in \pi_{R-S}(r) \land \forall u \in s : tu \in r\}$$

- Example: given the relations r, s:

r

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

s

D	E
a	1
b	1

$r \div s$

A	B	C
α	a	γ
γ	a	γ

Assignment

- Operation ($\longleftarrow$) that provides a convenient way to express complex queries.

  Idea: write query as sequential program consisting of a series of assignments followed by an expression whose value is "displayed" as the result of the query.

- Assignment must always be made to a temporary relation variable.

  The result to the right of $\longleftarrow$ is assigned to the relation variable on the left of the $\longleftarrow$. This variable may be used in subsequent expressions.

Example Queries

1. *List each book with its keywords.*

   BOOKS $\bowtie$ Descriptions

   Note that books having no keyword are not in the result.

2. *List each student with the books s/he has borrowed.*

   BOOKS $\bowtie$ (borrows $\bowtie$ STUDENTS)

3. *List the title of books written by the author 'Ullman'.*

   $\pi_{\text{Title}}(\sigma_{\text{AName='Ullman'}}(\text{BOOKS} \bowtie \text{has-written}))$

   or

   $\pi_{\text{Title}}(\text{BOOKS} \bowtie \sigma_{\text{AName='Ullman'}}(\text{has-written}))$

4. *List the authors of the books the student 'Smith' has borrowed.*

   $\pi_{\text{AName}}(\sigma_{\text{StName='Smith'}}(\text{has-written} \bowtie (\text{borrows} \bowtie \text{STUDENTS}))$

5. *Which books have both keywords 'database' and 'programming'?*

   $\text{BOOKS} \bowtie (\pi_{\text{DocId}}(\sigma_{\text{Keyword='database'}}(\text{Descriptions})) \cap \pi_{\text{DocId}}(\sigma_{\text{Keyword='programming'}}(\text{Descriptions})))$

   or

   $\text{BOOKS} \bowtie (\text{Descriptions} \div \{(\text{'database'}), (\text{'programming'})\})$

   with $\{(\text{'database'}), (\text{'programming'})\})$ being a constant relation.

6. Query 4 using assignments.

   $\text{temp1} \longleftarrow \text{borrows} \bowtie \text{STUDENTS}$

   $\text{temp2} \longleftarrow \text{has-written} \bowtie \text{temp1}$

   $\text{result} \longleftarrow \pi_{\text{AName}}(\sigma_{\text{StName='Smith'}}(\text{temp2}))$

Modifications of the Database

- The content of the database may be modified using the operations *insert*, *delete* or *update*.
- Operations can be expressed using the assignment operator. $r_{new} \longleftarrow$ operations on(r_{old})

Insert

- Either specify tuple(s) to be inserted, or write a query whose result is a set of tuples to be inserted.
- $r \longleftarrow r \cup E$, where r is a relation and E is a relational algebra expression.
- STUDENTS $\longleftarrow$ STUDENTS$\cup\{(1024, \text{'Clark'}, \text{'CSE'}, 26)\}$

Delete

- Analogous to insert, but $-$ operator instead of $\cup$ operator.
- Can only delete whole tuples, cannot delete values of particular attributes.
- STUDENTS $\longleftarrow$ STUDENTS $-$ $(\sigma_{\text{major='CS'}}(\text{STUDENTS}))$

Update

- Can be expressed as sequence of delete and insert operations. Delete operation deletes tuples with their old value(s) and insert operation inserts tuples with their new value(s).

ER-to-Relational Mapping

Step 1

» For each regular entity type E in the ER schema, create a relation R that includes all the simple attributes of E

» For composite attributes, use the simple component attributes.

» Choose one of the key attributes of E as the primary key for R.

» If the chosen key was a composite, the set of simple attributes that form it will together be the primary key of R.

Step 2

» For each weak entity type W in the ER schema with owner entity type E, create a relation R that includes all simple attributes (or simple components of composites) of W.

» Include as foreign key attributes of R the primary key attribute of the relation that corresponds to the owner entity type E.

» The primary key of R is the combination of the primary key of the owner and the partial key of the weak entity type, if any.

Step 3

» For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.

» Choose one of the relations (say S). Include as a foreign key in S the primary key of T.

» It is better to choose an entity type with total participation in R for the role of S.

» Include the simple attributes of R as attributes of S.

Step 4

» For each binary 1:N relationship type R, identify the relation S that represents the entity type participating at the N-side of the relationship

» Include as a foreign key in S the primary key of the relation T that represents the other entity type participating in R.

» Include the simple attributes of R as attributes of S.

Step 5

» For each binary M:N relationship type R, create a new relation S to represent R.

» Include as foreign key attributes in S the primary keys of the participating entity types.

» Their combination will form the primary key.

» Include any attributes of R as attributes of S.

Step 6

» For each multi-valued attribute A, create a new relation R.

» R will include an attribute corresponding to A, plus the primary key attribute K of the relation that has A as an attribute.

» The primary key of R is the combination of A and K.

Step 7

» For each n-ary relationship type R, where n>2, create a new relation S.

» Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types

» Include any attributes of R.

» The primary key of S is usually a combination of all the foreign keys in S.

ER TO RELATIONAL MAPPING Example

The ER model is convenient for representing an initial, high-level database design. Given an ER diagram describing a databa'3e, a standard approach is taken to generating a relational database schema that closely approximatesthe ER design. We now describe how to translate an ER diagram into a collection of tables with associated constraints, that is, a relational database schema.

3.5.1 Entity Sets to Tables

An entity set is mapped to a relation in a straightforward way: 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. A possible instance of the Employees entity set, containing three

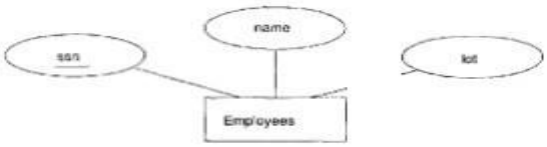


Figure 3.8 The Employees Entity Set

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      CHAR(11),
                         name     CHAR(30),
                         lot      INTEGER,
                         PRIMARY KEY (ssn) )
```

3.5.2 Relationship Sets (without Constraints) to Tables

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

To represent a relationship, we must be able to identify each participating entity and give values to the descriptive attributes of the relationship. 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.

The set of nondescriptive attributes is a superkey for the relation. If there are no key constraints, this set of attributes is a candidate key.

The set of nondescriptive attributes is a superkey for the relation. If there are no key constraints (see Section 2.4.1), this set of attributes is a candidate key.

Consider the Works_In2 relationship set shown in Figure 3.10. Each department has offices in several locations and we want to record the locations at which each employee works.

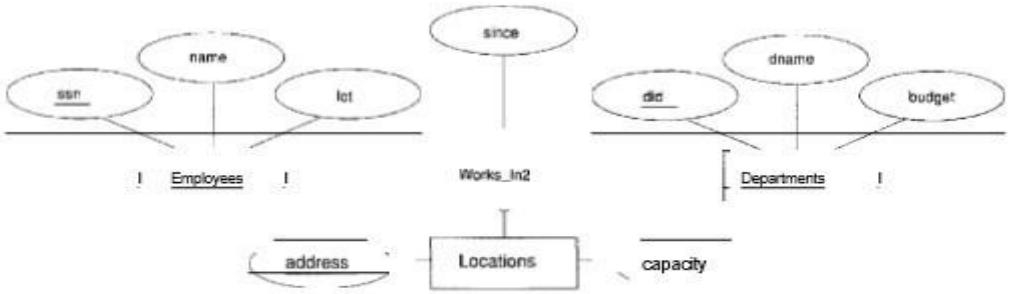


Figure 3.10 A Ternary Relationship Set

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

CREATE TABLE Works_In2 (ssn CHAR(11), did INTEGER, address CHAR(20) , since DATE, PRIMARY KEY (8sn, did, address), FOREIGN KEY (ssn) REFERENCES Employees, FOREIGN KEY (address) REFERENCES Locations, FOREIGN KEY (did) REFERENCES Departments);

Finally, consider the Reports_To relationship set shown in Figure 3.11. The

Figure 3.11 The Reports_To Relationship Set

role indicators *supervisor* and *subordinate* are used to create meaningful field names in the CREATE statement for the Reports_To table:

```
CREATE TABLE Reports_To (
        supervisor...ssn    CHAR(11),
        subordinate...ssn   CHAR(11),
        PRIMARY KEY (supervisor_ssn, subordinate_ssn),
        FOREIGN KEY (supervisor...ssn) REFERENCES Employees(ssn),
        FOREIGN KEY (subordinate...ssn) REFERENCES Employees(ssn) )
```

3.5.3 Translating Relationship Sets with Key Constraints

If a relationship set involves n entity sets and somem of them are linked via arrows in the ER diagTam, the key for anyone of these m entity sets constitutes a key for the relation to which the relationship set is mapped.

Consider the relationship set Manages shown in Figure 3.12. The table cor-

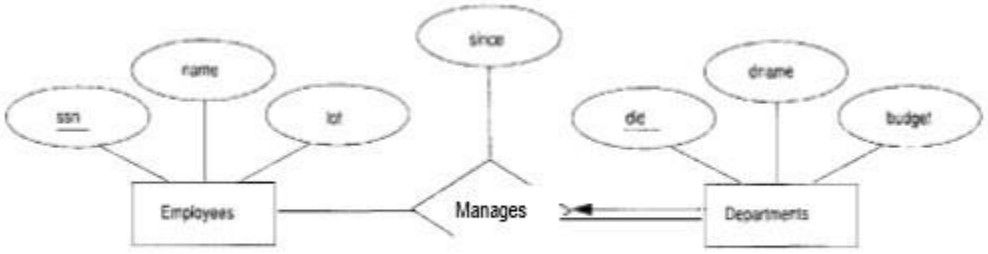


Figure 3.12 Key Constraint on Manages

The following SQL statement, defining a DepLMgr relation that captures the information in both Departments and Manages, illustrates the approach to translating relationship sets with key constraints:

CREATE TABLE DepLMgr(did INTEGER, dname CHAR(20), budget REAL, ssn CHAR (11) , since DATE, PRIMARY KEY (did), FOREIGN KEY (ssn) REFERENCES Employees)

Note that ssn can take on null values.

3.5.4 Translating Relationship Sets with Participation Constraints

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

Every department is required to have a manager, due to the participation constraint, and at most one manager, due to the key constraint.

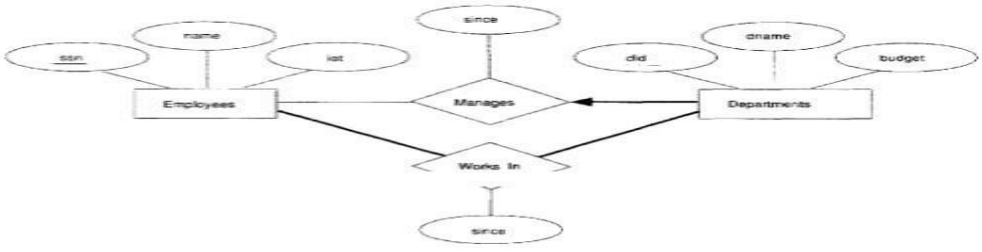


Figure 3.13 Manages and WorksJn

```
CREATE TABLE Dept_Mgr ( did       INTEGER,
                        dname     CHAR(20),
                        budget    REAL,
                        ssn       CHAR(11) NOT NULL,
                        since     DATE,
                        PRIMARY KEY (did),
                        FOREIGN KEY (ssn) REFERENCES Employees
                                  ON DELETE NO ACTION)
```

It also captures the participation constraint that every department must have a manager: Because ssn cannot take on null values, each tuple of Dep-Mgr identifies a tuple in Employees (who is the manager). The NO ACTION specification, which is the default and need not be explicitly specified, ensures that an Employees tuple cannot be deleted while it is pointed to by a Dept-Mgr tuple.

3.5.5 Translating Weak Entity Sets

A weak entity set always participates in a one-to-many binary relationship and has a key constraint and total participation.we must take into account that the weak entity has only a partial key. Also, when an owner entity is deleted, we want all owned weak entities to be deleted.

Consider the Dependents weak entity set shown in Figure 3.14, with partial key *pname*. A Dependents entity can be identified uniquely only if we take the key of the *owning* Employees entity and the *pname* of the Dependents entity, and the Dependents entity must be deleted if the owning Employees entity is deleted.

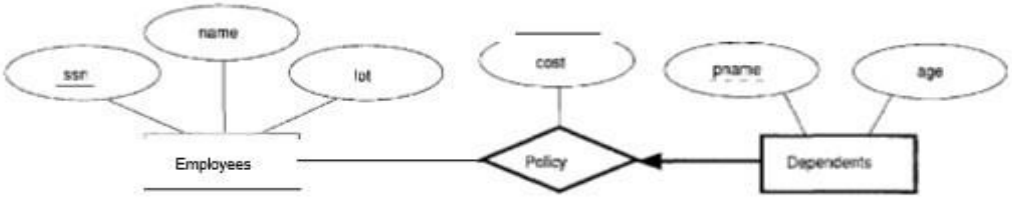


Figure 3.14 The Dependents Weak Entity Set

We can capture the desired semantics with the following definition of the Dep_Policy relation:

```
CREATE TABLE Dep_Policy (pname   CHAR(20),
                         age     INTEGER,
                         cost    REAL,
                         ssn     CHAR(11),
                         PRIMARY KEY (pname, ssn),
                         FOREIGN KEY (ssn) REFERENCES Employees
                                 ON DELETE CASCADE )
```

3.5.6 Translating Class Hierarchies

We present the two basic approaches to handling ISA hierarchies by applying them to the ER diagram shown in Figure 3.15:

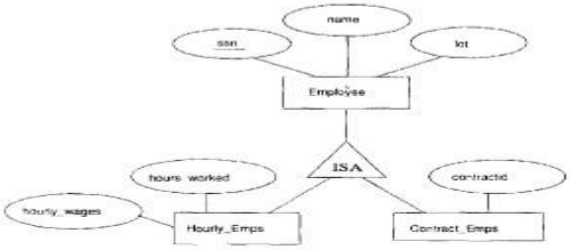


Figure 3.15 Class Hierarchy

1. We can map each of the entity sets Employees, Hourly_Emps, and ContracLEmps to a distinct relation. The Employees relation is created as in Section 2.2. We discuss Hourly_Emps here; ContracLEmps is handled similarly. The relation for Hourly_Emps includes the *hourly_wages* and *hours_worked* attributes of Hourly_Emps. It also contains the key attributes of the superclass *(ssn,* in this example), which serve as the primary key for Hourly_Emps, as well as a foreign key referencing the superclass (Employees). For each Hourly_Emps entity, the value of the *name* and *lot* attributes are stored in the corresponding row of the superclass (Employees). Note that if the superclass tuple is deleted, the delete must be cascaded to Hourly_Emps.
2. Alternatively, we can create just two relations, corresponding to Hourly_Emps and ContracLEmps. The relation for 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*).

3.5.7 Translating ER Diagrams with Aggregation

Consider the ER diagram shown in Figure 3.16. The Employees, Projects,

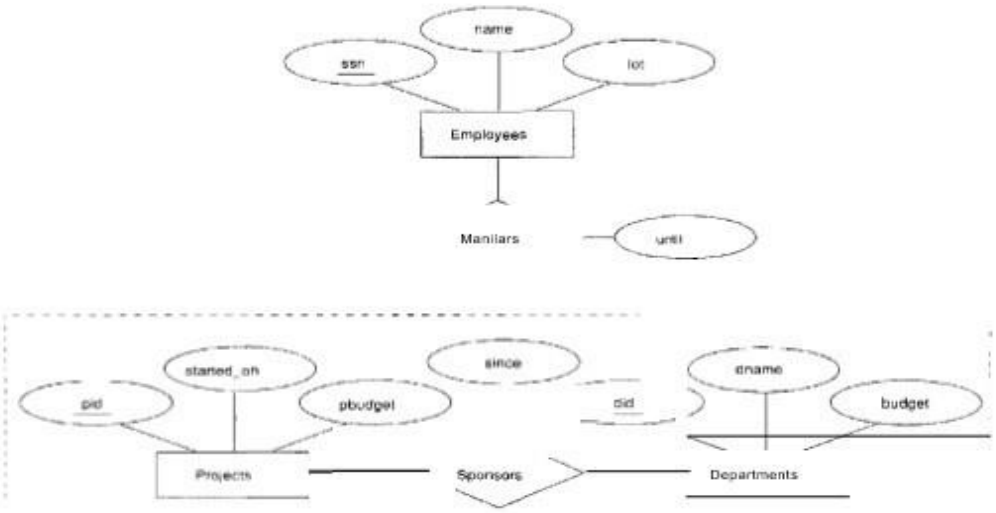


Figure 3.16 Aggregation

and Departments entity sets and the Sponsors relationship set are mapped as described in previous sections. For the Monitors relationship set, we create a relation with the following attributes: the key attributes of Employees *(88n),* the key attributes of Sponsors (*did, pid*), and the descriptive attributes of Monitors (*until*). This translation is essentially the standard mapping for a relationship set, as described in Section 3.5.2.

3.5.8 ER to Relational: Additional Examples

Consider the ER diagram shown in Figure 3.17. We can use the key constraints

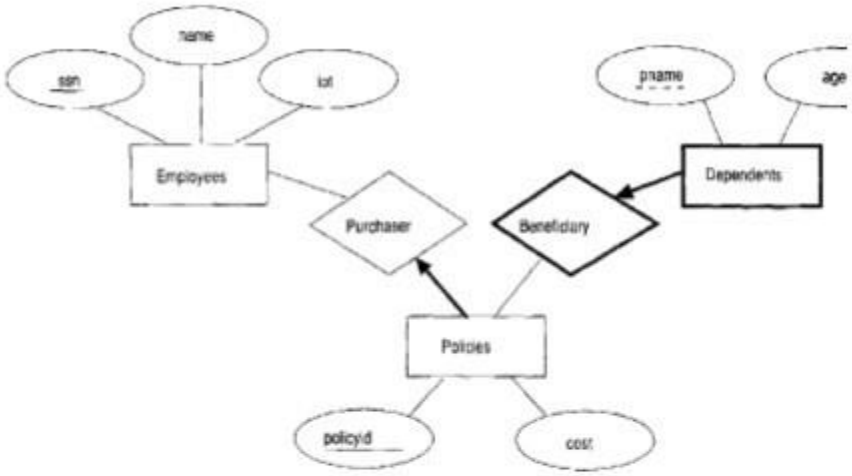


Figure 3.17 Policy Revisited

to combine Purchaser information with Policies and Beneficiary information with Dependents, and translate it into the relational model as follows:

```
CREATE TABLE Policies ( policyid  INTEGER,
                        cost      REAL,
                        ssn       CHAR(11) NOT NULL,
                        PRIMARY KEY (policyid),
                        FOREIGN KEY (ssn) REFERENCES Employees
                                ON DELETE CASCADE )
```

CREATE TABLE Dependents (pnameCHAR(20) , age INTEGER, policyid INTEGER, PRIMARY KEY (pname, policyid), FOREIGN KEY (policyid) REFERENCES Policies ON DELETE CASCADE);