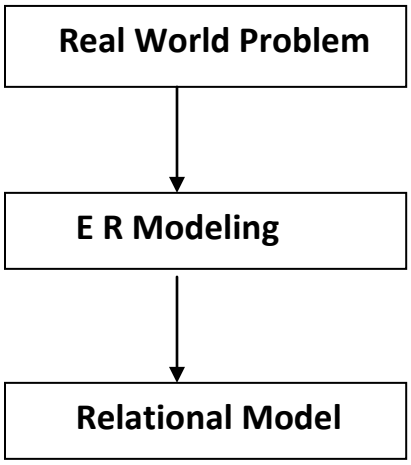
CS 344 - Database Management Systems

Lecture Notes – 12/09/2011



Q. Why it takes two steps to convert a real world problem to relational model?

Ans. Because ER Modeling is more intuitive and using ER modeling it is easy to analyze the relations among entity sets.

ER Model:

The *entity-relationship (ER) data model* allows us to describe the data involved in a real-world enterprise in terms of objects and their relationships and is widely used to develop an initial database design.
As the name suggests entity relationship model deals with entities and the relationships between them so defining the terms.

Entity Sets

It is the set of entities of similar type and entity is any object in the real world distinguishable from other objects. For Example In a book store all books from the same publisher is an entity set while a particular book is an entity.

- Putting all the similar entities altogether makes job easy for programmer and also it is scalable.

Attributes

An entity is described using a set of attributes. All entities in a given entity set have the same attributes. Attributes are of various types as follows.

Simple Attribute: Atomic value. Means can't be further divided. Ex Age, Color

Composite Attribute: Can be further divisible. Ex Roll Number of IITG student, Date of Birth, Course Name in IITG

Multi Valued Attribute: Attribute that can have multiple values of same type. Ex phone number, Account Number, Pat Name.

Derived Attribute: The value of attribute can be derived from the value of other attribute. Ex. Age from the DOB, Country from IP, place from PINCODE.

** There may be some examples which can't be represented using any of the above attribute. eg Video frames.

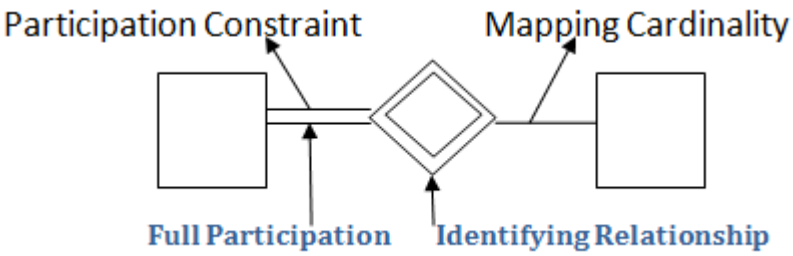
Q. Why do we require Derived Attributes?

Ans. Because Using derived attribute we can find the required value. We don't have to specifically store the other attribute separate in the table so it reduces the storage.

Weak Entity Sets and Strong Entity Sets: Weak entity sets are those which can't generate primary key on their own. While Strong entity sets have their primary keys.

Relationship Sets: A relationship gives the association between two or more entities. Grouping all similar relations what we get is a relationship set. Relationships have descriptive attributes i.e. they give additional information like if the leave date and duration is known then joining date can be estimated. There are four types of relationships as follows.

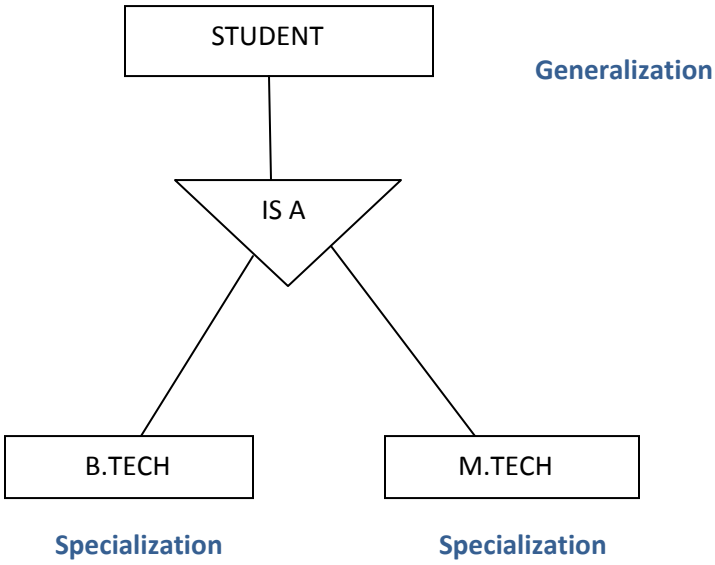
- One to One
- Many to One
- Many to Many
- One to Many



Specialization: An entity set may include subgroups of entities within it which are distinct in some way from other entities. The process of designating subgroupings within an entity set is called Specialization.

Generalization: Similar to the above, two specialized entity set can be represented as a super entity having the similar characteristics as the specialized entity sets.

Ex:



Aggregation: Its an abstraction through which relationships are treated as higher level entities.

Relational Model:

The *Relational data model* consists of a collection of tables, each of which is assigned a unique name. A row in a table represents a relationship among a set of values. Informally, a table is an entity set and a row is an entity.

Database Schema: It's the logical design of the database.

Keys:

Super Key: A super key is a set of one or more attributes that, taken collectively, allows us to indentify uniquely a tuple in the relation.

Candidate Key: Minimal superkeys are called Candidate Keys.

Primary Key: Key chosen by the database designer as the principle means of identifying tuples within a relation.

Relational-Algebra Operations:

Select: The *select* operation selects tuples that satisfy a given predicate. We use Greek letter sigma (σ) to denote selection operation.

Project: The *project* operation allows to select only a particular set of attributes. We use Geek letter pi (π) to denote projection operation.

Union: The *union* operation is used to "merge" two relations. It is denoted by (U) For two relations r and s, for r U s to be valid, we require that:

- The relations r and s must be having the same number of attributes
- The domains of the *i*th attribute of r and the *i*th attribute of s must be the same, for all *i*.

Set-Difference: The *set-difference* operation, denoted by minus sign (**-**) , allows us to find tuples that are in one relation but are not in another.

Cartesian-Product: It allows us to combine information from any two relations. It is denoted by cross (X)

Rename: This lets us rename a relation to an another name. It is denoted by Greek letter rho (ρ). To rename a relation/expression E as X, we write it as $\rho_x(E)$

Additional Relational-Algebra Operations:

Set-Intersection: This operation is used to select the tuples from two relations which are common to both. Set-Intersection operation is denoted by (∩)

Natural-Join: Consider two relations r(R) and s(S). The natural join of two relations r and s, denoted by (⋈), is a relation on schema R U S formally defined as follows:

$r \bowtie s = \pi_{RUS}(\sigma_{r.A1 = s.A1 \wedge r.A2 = s.A2 \wedge \ldots \wedge r.An = s.An} (r \times s))$

Division: Let r(R) and s(S) be relations, and let S ⊆ R; i.e, every attribute of schema S is also in schema R. The relation r ÷ s is a relation on schma R – S (i.e, on the schema containing all attributes of schema R that are not in schema S). A tuple t is in r ÷ s iff both of two conditions hold:

- t is in $\pi_{R-S}(r)$
- For every tuple t_s in s, there is a tuple t_r in r satisfying both of the following:
    - $t_r[S] = t_s[S]$
    - $t_r[R – S] = t$

Alternatively, $r \div s = \pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - \pi_{R-S,S}(r))$

Left outer Join: This operation, denoted by (⟕) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation and adds them to the result of the natural join.

Right outer join: This operation, denoted by (⟖) is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join.