Lecture Notes

Date: 23rd August 2011
Topic: Introduction to Normal Forms
Submitted By: Praveen A and Pratik Sarda

Database Design Comparison:

How do we compare 2 database designs and know which one is better??

Example

If a Department has multiple buildings how will you represent it in ER model:

ER Model 1: Buildings as multi valued attribute in entity set Department.

ER Model 2: Separate entity set Building with many to many relationships with entity set Department.

Comparisons between the two ER Models are:

1. In ER Model 1, there can be invalid building names in the Department table, while in ER Model 2 the relationship table will only have building names from entity set Building.

2. In ER Model 1, new building in campus which doesn't have any department as of now, can't be represented, while in ER Model 2, such a building can be entered in the entity set building even though no departments are related to it.

3. If the name of a building is changed, multiple changes will have to be made in ER Model 1, while in ER Model 2, the change will have to be made only once in the Building entity set, and it will reflect in all the relations.

The metric to compare two ER models should consider factors like:

1. Does it capture all entities

2. Does it capture all domains

3. Partial / Full participation of entity sets in relations

4. Cardinality of Attributes

Introduction to Schema Normalization:

One of the biggest problems in Database design is **redundancy**. Redundancy causes problems in updation, insertion and deletion of entries in databases. During updation all redundant copies have to be updated. During insertion it may not be possible to store certain information until some other unrelated information is stored as well. And during deletion it may not be possible to delete some information without losing some unrelated information as well. The previous example of building and department shows redundancy in first solution.

We should make sure that if single value changes, then it should change only one cell in the DBMS. To achieve this decompositions are performed. A **Decomposition** of a relation schema R is replacing it with two relation schemas such that each contain a subset of attributes of R and together contain all attributes in R.

Example 1: Instructor and Department:

Schema 1:

InstructorAndDepartment
InstructorID
InstructorName
DepartmentID
DepartmentName

In this schema DepartmentID and name is repeated every time an instructor belongs to the same department. Any other attribute of department would also be repeated. Any change in department name will lead to multiple changes in the table.

Schema 2:

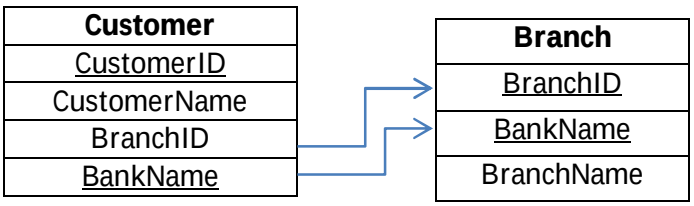
Here Department ID becomes a foreign key in the schema of Instructor referencing Department Schema. And only department ID is repeated. Any change in department name will lead to change in one place itself. Schema 2 is a decomposition of Schema 1 and is a better option than schema 1.Schema2 involves multiple table lookups.

Example 2: Bank Customer And Branch

Schema 1:

CustomerAndBranch
CustomerID
CustomerName
BranchID
BranchName
BankName

Schema 2:



This ensures

1. One table for each entity set and one table for each relationship.

2. Ease in maintaining Consistency.

The **goal of schema normalization** is to maintain right balance between reducing redundancy and multiple table lookups.

How do we know how much a schema has to be decomposed?? This is solved by introducing normal forms. Normal forms for relations make sure that certain kind of problems cannot arise in the schema. There are various kinds of Normal Forms like first normal form, second normal form, BCNF etc.

First Normal Form:

The first normal form is the most basic among the normal forms. It imposes a very basic requirement on relations. A domain is **atomic** if elements of the domain are considered to be indivisible units. Examples of non-atomic domains are address (consists of street name, pin code etc), roll numbers in our institute (first two digits tell the year of joining).Integers are assumed to be atomic as they cannot be divisible. A schema is in **First normal form** if the domain of every attribute is atomic.

Example 1

Relation Student= (StudentID, Department, YearOfJoining, SName) is in first normal form as every attribute is indivisible.

However the relation Student= (StudentID, Department, Sname) with student id of the form 07 012345 or 08 123456 is not in first normal form as student ID is used to find the year of joining.

Example 2

Relation Courses= (CourseID, Instructor, Department)

Here every attribute is indivisible and hence is in First normal form

Relation Courses= (CourseID, Instructor)

Here Course ID is of the form CS344 or CS345 which is used to find the department name and hence CourseID is divisible and hence is not in First Normal Form

Functional Dependencies:

Functional Dependencies are very important in distinguishing good database designs from bad database designs. They are constraints on the set of Legal relations.

A legal instance r(R) of Schema R satisfies **Functional Dependency** $\alpha \rightarrow \beta$ if

1. Both α and β are subsets of R

2. For all pairs of tuples t1 and t2 in the instance if t1[α]=t2[α] then t1[β]=t2[β]

In schema Courses=(CourseID, Instructor, Credits) if α=CourseID and β=Credits then α ->β forms a functional dependency.

Trivial Case: If β is a subset of α then α ->β is a functional dependency.

Super, Primary and Candidate Keys are types of functional dependencies. That is if any of these keys are present in alpha then α ->β holds for any β in the relation.

A single functional dependency on a relation R will not be very useful but if we have all the functional dependencies they are very useful in checking normal forms. Given a set of functional dependencies F we can find a larger set of functional dependencies using that set.

Axioms allow us to reason about functional dependencies with ease. The following 3 axioms known as **Armstrong's axioms** are used repeatedly:

α, β and γ are set of attributes.

1. Reflexivity: if $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ holds
2. Augmentation: If $\alpha \rightarrow \beta$ then $\gamma\alpha \rightarrow \gamma\beta$ holds
3. Transitivity: If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.

References:

1. Database System Concepts: Silberschatz, Sudarshan and Korth
2. Database Management Systems: Ramakrishnan