Guidelines For Validating OLTP Database Design

DOCUMENT HISTORY

Version	Date	Author	Comments
0.01	25 May 2007	Anirudhan Velur	Initial Draft
1.0	30 May 2007	Anirudhan Velur	Baselined
1.1	15 Oct 2007	Anirudhan Velur	Updated Generalization Hierarchies

1 INTRODUCTION

Experts in E-R modeling will readily admit that data modeling is more an art than a science. Although the transformation of an E-R diagram to a relational schema follows a set of well-defined, straight-forward rules, errors in an E-R diagram can lead to normalization problems which the transformation rules fail to capture.

The modern approach to structured systems analysis and design involves the use of systems analyst workbench technology for computer-aided software engineering (CASE). Active-in-development CASE tools provide systems analysts a graphical user interface for building process and data model diagrams. Ultimately, the conceptual data model diagrams are converted to physical relational database schemas that incorporate data and referential integrity constraints. The process model diagrams are used to generate screens, reports, menus, and computer programming code to implement business processes that maintain the data that are stored in the relational tables.

Many CASE tools incorporate a data modeling approach based on Chen's (1976) Entity-Relationship (E-R) model. The automatic conversion of an E-R diagram to a relational database schema through the use of CASE technology provides an extremely productive systems development environment. Most CASE tools advertise the ability to produce a normalized schema, at least to third normal form; however, the extent to which the generated schema is actually normalized is a function of the accuracy of the E-R diagram. As Ling (1985) points out, "it is very difficult to determine whether an E-R diagram is the best representation for a given database."

2 GENERAL TYPES OF ERRORS

2.1 Non Identification Of Entity

The major problem of the OLTP design is missed entities. The implementation rules for identifying entities, relationships, and attributes include an English language sentence structure analogy where the nouns of a descriptive sentence identify entities, verbs identify relationships, and adjectives identify attributes. These rules have been defined by
Chen (1983) as follows:

- Rule 1: A common noun (such as person, chair), in English corresponds to an entity type on an E-R diagram.
- Rule 2: A transitive verb in English corresponds to a relationship type in an E-R diagram.
- Rule 3: An adjective in English corresponds to an attribute of an entity in an E-R diagram.

English statement: A person may own a car and may belong to a political party. Analysis: “person,” “car,” and “political party” are nouns and therefore correspond to entity types. … “own” and “belong to” are transitive verbs (or verb phases) and therefore define relationships.

Suppose the given requirements is the Employee has the following information to be stored employee id , first name, last name , date of birth and department . Each employee works in department . To perform the data modeling task, you begin by defining your entities, the significant objects of interest. Entities are the things about which you want to store information. For example, you might want to define an entity for employees called EMPLOYEE because you need to store information about everyone who works for your organization. Suppose the requirement document does not mention anything about the department and department is given as attribute in employee table. Normally the designer create a department as attribute in

employee table and finishes of his task. But eventually he misses an entity department . You wont able to find out the missed entity until you feeds the data into the table.

You might also define an entity, called DEPARTMENT, for departments.

2.2 Primary key definition

Next, you define primary keys for your entities. A primary key is a unique identifier for an entity. Choosing a primary key is one of the most important steps in good database design. A primary key is a table column that serves a special purpose. Each database table needs a primary key because it ensures row-level accessibility. If you choose an appropriate primary key, you can specify a primary key value, which lets you query each table row individually and modify each row without altering other rows in the same table. The values that compose a primary key column are unique; no two values are the same.

Each table has one and only one primary key, which can consist of one or many columns. A concatenated primary key comprises two or more columns. In a single table, you might find several columns, or groups of columns, that might serve as a primary key and are called candidate keys. A table can have more than one candidate key, but only one candidate key can become the primary key for that table. For the non-existence dependent (sometimes called prime) entities, select the appropriate primary key attribute(s). From the defined attributes for the entity, choose the minimal set that guarantees uniqueness for all instances, that always has a required value, and whose value will never change. If no such attribute or set of attributes can be found, consider adding a system-assigned attribute as the primary key. In the case of the EMPLOYEE entity, you probably need to store lots of information. However, most of this information (such as gender, birth date, address, and hire date) would not be a good choice for the primary key. In this case, you could choose a unique employee ID or number (EMPLOYEE_NUMBER) as the primary key. In the case of the DEPARTMENT entity, you could use a unique department number (DEPARTMENT_NUMBER) as the primary key.

In good old days Database Designer tends to use sequence objects only because of the following reasons

- No natural primary keys are available on the table .
- If the PK constitute multiple attributes and one of the migrated/ attribute can be nullable.
- If the PK is constitutes more than 4 attributes and has nested child tables under it (Performance / Space requirements for indexes)

By definition, a relational database must contain normalized tables, and to be properly normalized, a table must contain a primary key. Database developers often disagree about whether it's better to use naturally occurring data or meaningless values as a table's primary key

Primary keys are often best served by introducing an *artificial key* attribute if a natural primary key cannot be found. Perhaps simply a *record number*

2.2.1 Natural Vs Surrogate key

Now we discuss the pros and cons of Natural Vs Surrogate key in the current scenario . Before getting into so deep let us understand the basic definitions

What is a Primary key ?

- The primary key is an attribute or a set of attributes that uniquely identify a specific instance of an entity. Every entity in the data model must have a primary key whose values uniquely identify instances of the entity.
- To qualify as a primary key for an entity, an attribute must have the following properties:
    - It must have a non-null value for each instance of the entity.
    - The value must be unique for each instance of an entity.
    - The values must not change or become null during the life of each entity instance.
    - The primary key-value must exist when the record is created.
    - The primary key must be compact and contain the fewest possible attributes.
    - The primary key must remain stable—you can't change the primary-key field(s).

Finding Candidate Keys

- An instance of a relation cannot be used to prove that an attribute or combination of attributes is a candidate key.
- In other words, we cannot examine a relation and decide, for example, that *Postcode* is a candidate key simply because no two rows share the same post code.
- The fact that there are no duplicates at a particular moment in time does not guarantee that duplicates are not possible.
- However, the presence of duplicates may be used to show that a certain attribute or combination of attributes is not a candidate key.

Therefore, to correctly identify a candidate key, we need to be aware of the meaning of attributes in the real world and think about whether duplicates *could* arise for a given choice of key.

Finding Primary Keys

- For a given relation, the *primary key* is one of the candidate keys. It is chosen from the candidate keys in order to uniquely identify records within the relation.
- Since a relation cannot have duplicate rows (by definition), it is always possible to uniquely identify each row.
    - This means that every relation has at least one candidate key.
    - Hence, a primary key can always be found.
- In the worst case, the entire set of attributes could serve as the primary key, but usually some smaller subset is sufficient.

2.2.2 Why Natural Primary Key

In good old days we used to use only natural primary keys for all tables. As a data modeler I myself was a fan of Natural primary key and argued countless no of times. A *natural* key is ideally how the business would identify an entity instance**.** A key that is formed of attributes that already exist in the real world. For example, U.S. citizens are issued a Social Security Number (SSN) that is unique to them (this isn't guaranteed to be true, but it's pretty darn close in practice)

The following are some of the positive points of Natural Primary key

1. Easy to understand example SSN is unique and easily identify one person
2. Easily implement universally accepted codes example United states State codes
3. Cannot be easily duplicated unless there is typing mistake
4. Reduction of no of index required since most of the time the record is linked and accessed by primary keys

The advantage of natural keys is that they exist already, you don't need to introduce a new "unnatural" value to your data schema.

2.2.3 Why Surrogate key

An artificial or surrogate key is one that has no meaning to the business or organization. During good old days Artificial keys are permitted when:

- No attribute has all the primary key properties
- The primary key is large and complex.

Now during the past few years the situation/ requirement for database design changed. Now most of the applications are global in Nature and want to implement the DB design globally. Furthermore There are so many security laws prevails before disclosing any type of personal data to outside world like Social security no , Account numbers , Credit card numbers etc

There are many strict laws to prevent the disclosure of personal data like

- Payment Card Industry Data Security Standard (PCIDSS)
- Sarbanes-Oxley Act
- HIPAA (Health Insurance Portability and Accountability Act)
- European Union Data Protection Directive
- California's Database Security Breach Notification Act(California Senate Bill 1386)
- Gramm-Leach-Bliley Act
- Federal Information Security Management Act

If you use any of the personal identification details that are critical and can be misused by another person you need to keep it as secured as possible. Because of the security issue natural primary keys like SSN need to be encrypted and kept. If you encrypt a primary key, you will have to encrypt all referencing foreign keys. If you encrypt an indexed column, you may end up with slow queries when trying to use the encrypted value.

In Today's world SSN is bad choice of PK because of in many cases, people are not required to reveal their SSN — so they could opt out of that field, and leave your primary key NULL. For employment this is probably a non-issue, but you can imagine other systems where people are logged in a database but SSN is not a legal requirement. SSNs can be re-issued after death, so unless your system is constantly cleaning out data for people who are deceased, the chance of duplicates is there.

2.2.3.1 Scalability

Take another example of vehicle sales business in which the natural PK for identifying one vehicle is vehicle VIN number. Currently the company sells only newer vehicle. But when the company starts re-sales of vehicle and selling vehicles and equipment via land-based and online auction, and though VIN does not makes as a KEY. VIN is not unique since re-sales company can see the same vehicle over and over yet and because we need to maintain its state each time it arrive and leaves So need to create a Surrogate key to keep track of vehicle

Another example of a look up table is one that contains a row for each state, province, or territory in North America. For example there would be a row for California, a US state, and for Ontario, a Canadian province. The primary goal of this table is to provide an official list of these geographical entities, a list that is reasonably static over time A valid natural key for this table would be the state code, a unique two character code – e.g. CA for California and ON for Ontario. But Canada has same type of state codes "NW" if the application needs to be implemented globally the approach of unique state code wont work unless country is the part of PK in state

2.2.3.2 Easy for Correcting errors

Let's further complicate the example by supposing that you enter the SSN used as PK incorrectly. Initially, this doesn't seem like such a big problem; you simply correct the value before pressing save. But once you save, changing the PK is not allowed by application , if you change it often violates referential integrity if there are related records. With referential integrity features enabled like update cascade, you can usually update a primary-key value, and the data engine will update related values automatically. But just because you *can* update a primary-key value doesn't mean you *should*, and you definitely shouldn't allow the uneducated user to do so. A primary-key value shouldn't be subject to data entry errors, because changing the value violates a rule. Now you've got a problem that your application's referential integrity feature isn't equipped to resolve.

2.2.3.3 Not Attached to Business

Another disadvantage of natural keys is that because they have business meaning they are effectively coupled to your business: you may need to rework your key when your business requirements change. For example, if your users decide to make Customer Number alphanumeric instead of numeric then in addition to updating the schema for the Customer table (which is unavoidable) you would have to change every single table where Customer Number is used as a foreign key

2.2.3.4 Location independent

If the entity you are modeling contains data from multiple sources, there is usually value in creating a surrogate key. For example, assume you learn that both Henry Winkler the student and Hank Winkler the instructor are really the same person. You can add this person as a unique row to the person entity identified by a person identifier surrogate key. This person can then play the two roles of instructor and student.

2.2.3.5 Language independent

When an application is implemented globally and If you need to store the data centrally you need to use surrogate key (Number) as PK since Pk should not be depend on the local language, If the PK is stored as Natural key it will be difficult to show in different languages so that the users in other geography can understand it . So if you want to implement application globally its better to use surrogate key as PK

2.2.3.6 Easier to Map to Object model

Now most of the application are web based there is a need for object relation mapping. The tools like Hibernate can not use natural primary keys

2.2.3.7 Save Space on Foreign Keys

if you have any tables that are associated with the master table, and therefore have foreign key columns that refer to the primary key of your master, then having a separate primary key can actually save space overall, as the data for the master columns doesn't have to be duplicated across all the linked tables.

2.2.3.8 Easier to Join tables

Since its Simple one key join its simply the joins from multiple tables and improve the performance

2.2.4 Issues with Surrogate key

2.2.4.1.1 Additional Index and keys

The database need to additional unique index /Inversion index for constraints for alternate keys /inversion keys since its difficult to check duplicate records with artificial keys.

2.2.4.2 Difficulty in application coding –lookup tables

Since most of the look up table need to be pre populated before application coding and the look up codes mat be hardcoded within the application many times. While deploying the application at client site the look up needs to be pre-populated. There is no guarantee that same surrogate key will be generated at client site and can break the code

2.2.4.3 Difficulty Of understanding the data

Since all the keys contains numeric data once joined by multiple table (Many people give “Id” as PK column name without any prefix of suffix), can find it difficult to understand the data whether its customer id, employee id or department id

2.2.5 Summary

Criteria	Natural Key	Surrogate Key
Uniqueness	Subject to errors	Always unique
Null ability	Cannot enter data until value is known	Record Can be created
Non Changeable	Subjected to change in business requirements	Neutral to business- Never gets changed
Key Complexity	Can be complex with many fields	Always one field
Join operation	Complex	Simple
PK Index Size	Large	Small
Functional Dependency	Attached Business – Has functional dependency	Artificial – No functional dependency
Normalization	Strictly adhere to Normal form Since PK has functional dependency	Artificial Key- Not attached to business. Might not be adhere to normal form
Data Security	Might not be Adhere to Data Protection laws	Adhere to Data Protection laws
Data Presentation	Might be able to understand data without meta data	Might not be able to understand data without metadata like column names
Application domain	More suited for Local application	Suited for Global Application
Indexes	No additional index required	Additional indexes are required for AK
Keys	AK might not be Required	AK is a must
Design Scalability (Many location/Geography implementation)	Might not be scalable	Scalable

Its Apparent that the Database Designer must consider the various points as discussed above before deciding on the primary key choice. Final decision I am leaving to the Database designer's wisdom and customer's standards whether to keep natural key or an surrogate key.

While Using Artificial keys (may be required) but normally after creating the Table design the designer tend to forget alternate key for that. It creates Data inconsistency and instability.

2.3 Missed Requirements

One of the major problem in database design is missed/ messed up requirements. The Modeler may miss the requirement which may intern very important for data validity and integrity. Because a data model exposes many of the business rules that describe the area being modeled, reading the requirements and discussing with the user groups are very important. The designer has to understand the proper meaning of verbal phrase used to note down the requirements. The verbs like "must " , "may" can make impact on the data base design

In general, there are two classes of E-R modeling errors that lead to normalization problems: (1) the "incomplete data model" error, and (2) the "mis-modeled problem domain" error The incomplete data model error tends to occur in situations where the systems analysts are tasked to build a computer-based information system that is limited in scope. A key objective for successful information system project management is the definition of a limited, yet adequate project scope--a scope that enables the production of system deliverables within a reasonable time period. Limiting a project's scope often results in information systems that are based on limited data models. Limited information systems are fairly common throughout the IS world where dissimilar technologies prevent data sharing and work against the concept of a shared, enterprise-wide database.

The mis-modeled problem domain error is actually a class of errors including those that arise whenever systems analysts lack a complete understanding of the problem domain. These include errors such as depicting an attribute as single-valued when, in fact, the attribute is multi-valued, or depicting a single entity which includes attributes that should be assigned to two separate entities, or mis-modeling the connectivity or degree of a relationship.

In order to portray the errors associated with the various normal forms, we use an example conceptual model that is based on the university modeling problem given in Figure 1. This example is used, in some form or another, in most textbooks on database management systems, and offers the potential for violations of all normal form definitions. While studying the errors depicted below, keep in mind that this university modeling problem is well-known, and, in fact, it is highly unlikely that an experienced database designer would make the mistakes depicted. However, even expert database designers can make the type of mistakes depicted below when they are thrust into situations that require modeling problem domains in which they lack experience, and inexperienced database designers are more prone to make errors..

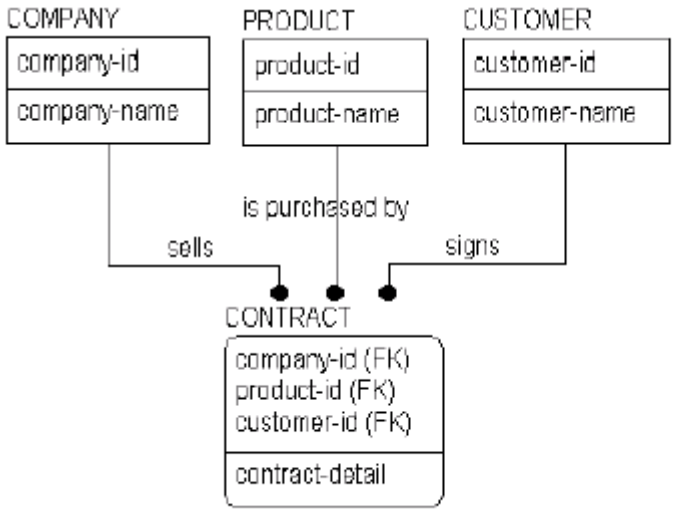
3 MISINTERPRETATION OF REQUIREMENTS

One of the major problems in database design is missed requirements. The Modeler may miss the requirement which may intern very important for data validity and integrity. If you choose your verb phrases correctly, you should be able to “read” a relationship from the parent to the child using an “active” verb phrase.
Example:
A plane flights<transports> many Passengers.
Verb phrases can also be read from the perspective of the child entity. You can often read from the child entity perspective using “passive” verb phrases. For example:
Many Passengers <are transported by> a Plane flight.
Because a data model exposes many of the business rules that describe the area being modeled, reading the relationships helps you validate that the design of the logical model is correct. Verb phrases provide a brief summary of the business rules embodied by relationships. And although they do not precisely describe the rules, verb phrases do provide an initial sense of how the entities are connected.

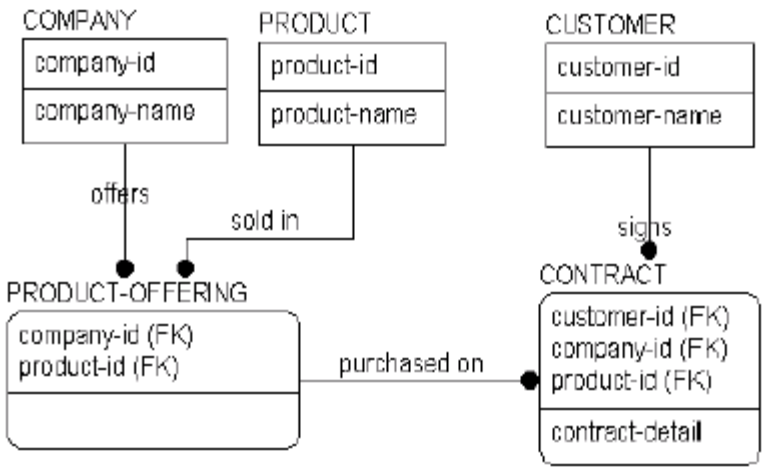
For example:
The requirement may look like ‘Employee may work in a department ‘. The migrated department key has to be null. If the requirement is like ‘Employee must work at least for a department’ this shows that the employee is allocated to a department and can not exist without a department allocated to him. The migrated department key should be not null. It means that there should be tracebility of requirement to your Data model

It is a good practice to make sure that each verb phrase in the model results in valid statements. Reading your model back to the business analysts and subject matter experts is one of the primary methods of verifying that it correctly captures the business rules.

When a single parent-child relationship exists, the relationship is called binary. All of the previous examples of relationships to this point have been binary relationships. However, when creating a data model, it is not uncommon to come across n-ary relationships, the modeling name for relationships between two or more parent entities and a single child table. An example of an n-ary relationship is shown next.



Like many-to-many relationships, three-, four-, or “n-ary” relationships are valid constructs in entity-relationship diagrams. Also like many-to-many relationships, n-ary relationships should be resolved in later models using a set of binary relationships to an associative entity.
If you consider the business rule stated in the previous example, you can see that a CONTRACT represents a three-way relationship among COMPANY, PRODUCT, and CUSTOMER. The structure indicates that many COMPANYs sell many PRODUCTs to many CUSTOMERs. When you see a relationship like this, however, you know that there are business questions begging to be answered. For example, “Must a product be offered by a company before it can be sold?” “Can a customer establish a single contract including products from several different companies?” and, “Do you need to keep track of which customers ‘belong to’ which companies?” Depending on the answers, the structures may change.
If, for example, the answer to the question "Must a product be offered by a company before it can be sold?” is “yes,” then you would have to change the structure as shown.



Because PRODUCTs must be offered by COMPANYs, you can create an associative entity to capture this relationship. As a result, the original “threeway” relationship to CONTRACT is replaced by two, “two-way” relationships. By asking a variety of business questions, it is likely that you will find that most “n-ary” relationships can be broken down into a series of relationships to associative entities.

3.1 Check Constraints

Another problem in OLTP system may be due to incorrect / no check constraints on data columns. If the data can only belong to limited number of pre defined sets, the checks constraints have to be used ensure data integrity. For example Male / Female. The data can belong to only M or F. This is most common missed conversion of requirement to design. The Male Female column should be not null. There is no meaning of null value to a male female column. If the data values are more than five and its likely to get added in future , It better to add a lookup table for that rather than creating check constraints

3.2 Nullable columns

Another issue is with nullable columns. Many modelers create table structure will all nullable columns. This will be a totally bad design since it raise question mark on existence on that entity itself. If you have nullable migrated keys its have major problem with performance. For example If the requirements may like ‘The employee may work in a department ‘ If you keep the dept code in employee nullable , If you need a report to find out how many employees are not allocated to any department ,you have to use SQL in which you specify the dept code is null . This types of SQL will go for a full table scan and can create performance hazard if the table is very big. In order to avoid this problem its better to have not null constraints and while entering the master data for department, you can devote one record for unknown department and use it for unknown values instead of populating null on that columns.

3.3 Recursive Relationship

Its better to avoid recursive relationship in your table design for example: Employee is reporting to one supervisor most of the people create recursive relationship on that table (Figure A). If the designer want to implement requirement is like “Employee always have a supervisor “may not be possible by this design. Suppose the user requires a report which contain all employees who do not have a supervisor may take a longer time to execute since that query wont use the FK index (Go for a full table scan since null is used in the SQL . Select * from Employee where supervisor id is null). In these circum stances its better to use a separate table contains the employee id and supervisor id.

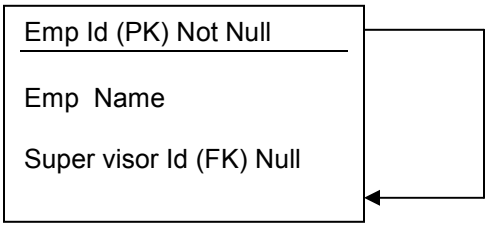


Figure A

4 GENERALAIZATION HIERRARCHY

4.1 What Is generalization

A generalization hierarchy is a structured grouping of entities that share common attributes. It is a powerful and widely used method for representing common characteristics among entities while preserving their differences. It is the relationship between an entity and one or more refined versions. The entity being refined is called the *supertype* and each refined version is called the *subtype.* The general form for a generalization hierarchy is shown below

Generalization hierarchies should be used when
(1) A large number of entities appear to be of the same type
(2) Attributes are repeated for multiple entities, or
(3) The model is continually evolving. Generalization hierarchies improve the stability of the model by allowing changes to be made only to those entities germane to the change and simplify the model by reducing the number of entities in the model.

4.1.1 Creating a Generalization Hierarchy

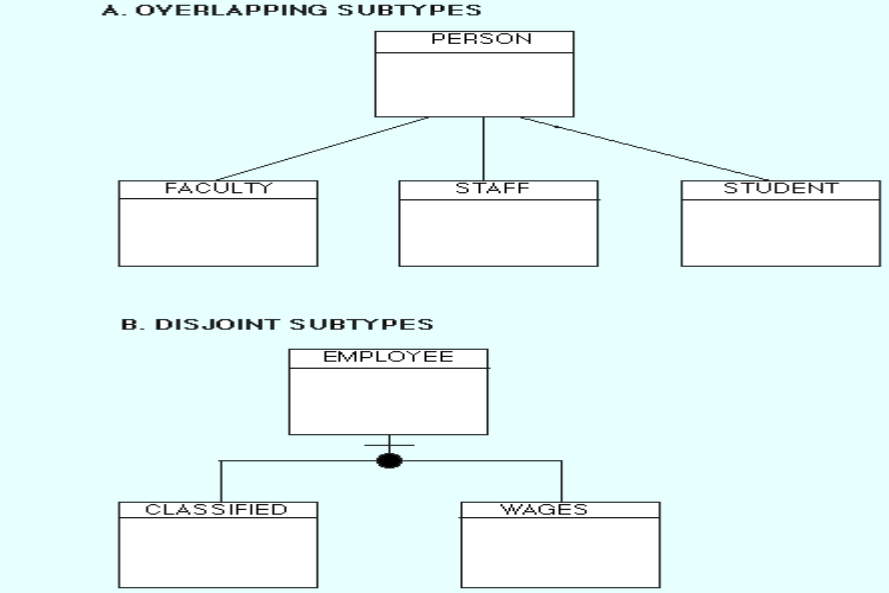
To construct a generalization hierarchy, all common attributes are assigned to the supertype. The supertype is also assigned an attribute, called a discriminator, whose values identify the categories of the subtypes. Attributes unique to a category, are assigned to the appropriate subtype. Each subtype also inherits the primary key of the supertype. Subtypes that have only a primary key should be eliminated. Subtypes are related to the supertypes through a one-to-one relationship.

4.1.2 Types of Hierarchies

A generalization hierarchy can either be overlapping or disjoint. In an overlapping hierarchy an entity instance can be part of multiple subtypes. For example, to represent people at a university you have identified the supertype entity PERSON which has three subtypes, FACULTY, STAFF, and STUDENT. It is quite possible for an individual to be in more than one subtype, a staff member who is also registered as a student, for example.
In a disjoint hierarchy, an entity instance can be in only one subtype. For example, the entity EMPLOYEE, may have two subtypes, CLASSIFIED and WAGES. An employee may be one type or the other but not both.

Figure 1 shows A) overlapping and B) disjoint generalization hierarchy.

Figure 1: Examples of Generalization Hierarchies



4.1.3 Rules

The primary rule of generalization hierarchies is that each instance of the supertype entity must appear in at least one subtype; likewise, an instance of the subtype must appear in the supertype.

Subtypes can be a part of only one generalization hierarchy. That is, a subtype can not be related to more than one supertype. However, generalization hierarchies may be nested by having the subtype of one hierarchy be the supertype for another.

Subtypes may be the parent entity in a relationship but not the child. If this were allowed, the subtype would inherit two primary keys.

4.2 Advantages & Disadvantages of Generalization hierarchy

The following are advantages and disadvantages of using generalization hierarchy

Advantage	*Description*
Column optionally maintained	*I can enforce the optionality of the columns by using NOT NULL column constraints.*
Only applicable columns are stored	*If the Employee is a Full Time Employee, the Temporary Employee table is not populated.*
Table scans faster for super type table	*This is due to the reduced row length for the Super type table.*
Physical data model reflects the true business rule	*This advantage is never given its just due. Many modelers are too lazy to maintain the Logical and Physical views of a data model.*
If there are relationships that only apply to one of the sub types of Employee, they can be accurately and easily enforced	*Adding a foreign key from (or to) the subtype is all that is necessary*
Data integrity can be fully met within the database	*This does require the creation of some simple triggers, but the importance of data integrity cannot be overstated—wrong data is still wrong, no matter how fast it is retrieved.*

And here are the disadvantages:

Disadvantage	Description
Two inserts always required	Assuming a fully enumerated super type, at least one sub type table will always be required for each super type row.
Possible multiple updates	If the set of columns to be updated includes super and sub type tables, then two updates (or more if the sub types are not mutually exclusive) are required.
Read from sub and super type table requires a join or a view	If the query requires columns from both super and sub type tables, then a join is required. (Unless a view is created).
This implementation requires one trigger per table to enforce - one in each of the sub type tables to validate the discriminator, plus one in the super type table to handle updated discriminator values.	This is code that needs to be maintained as part of the database
Full table scan with super and sub type tables is slow due to join	This is the worst performing case scenario; a full table scan requiring a join between the super and sub type tables.

4.3 Rolling Up and Rolling down

The Generalization hierarchy can be roll up /row down depend on the conditions

- Rolling up: Remove the subtypes and copy all of the data elements and relationships from each subtype to the supertype. Also add a type code to distinguish the subtypes.
- Rolling down: Remove the supertype entity and copy all of the data elements and relationships from the supertype to each of the subtypes.

4.3.1 Rolling Up

This is a popular option implemented for obvious reasons. There are less database objects to maintain and the queries are simple. In this option, you take all the columns from each of your sub type tables and "roll up" or As I've previously noted, I am now forced to make all of my sub type columns optional (or nullable).
By Rolling up I am forced to make all of my sub type columns optional (or nullable).The sub type columns are NOT NULL are now NULL since I combined subtype attributes in same table. I have to use another means (e.g. check constraint, application code) to enforce the column optionally. Certainly a check constraint can enforce the business rules, but these cannot be shown in a data model. I've lost the ability to reflect some business rules in the physical model.

4.3.2 Rolling Down

The last option is to "roll down" the super type entity into each of the sub type entities. By "roll down", I mean to take all of the super type attributes and copy them to each of the sub type entities.

Note
When you create a rollup or roll down transform, its logical partition of data by horizontal or vertical partition transform to the resulting table.

4.4 When To Roll Up /Roll Down

There are numbers of factors you need to take care of before deciding whether we need to roll up /roll down. The below is the example which clearly discuss the reason for rolling up /rolling down the generalization hierarchy

Requirements

- User can have one or more address
- Organization can have one or more address
- Subject can have one or more address

Note

There is no many to many relationships. All are one to many from respective master tables – like User , Organization, subject . The address is shared across Organization, user or subject
The entities required for the above requirements are

- User
- Organization
- Subject
- Address Type -> Residence, Official
- Country
- State /Region
- County
-

If you follow some Standards the following attributes need in the Address table . All the below attributes are common to all the addresses except the FKS from respective tables like USER , ORGANIZATION , SUBJECT to denote the type of address

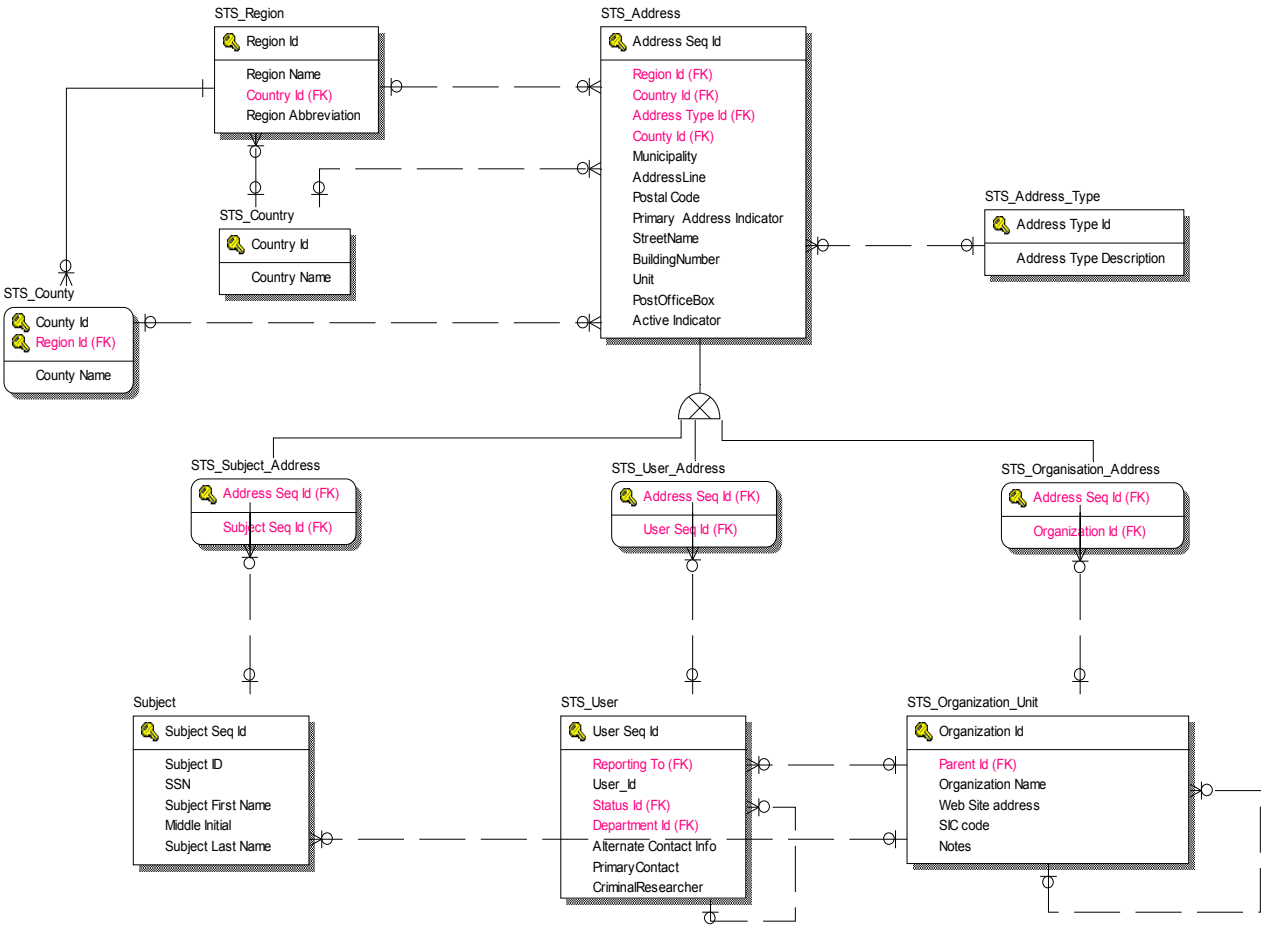
Attribute	Description
Postal Code	This field stores the Postal Code of the address
Active Indicator	This field store Y/N values indicates whether the corresponding address

	is active or not
Primary Address Indicator	This field store Y/N values indicates whether the corresponding address is user's primary address or not
Municipality	This field stores municipality
AddressLine 1	This field stores Address Line 1
StreetName	This field stores Street Name
BuildingNumber	This field stores Building Number
Unit	This field stores Unit
PostOfficeBox	This field stores Post Office Box Number
AddressLine 2	This field stores AddressLine2
Region Id (FK)	This field stores the state Id which uniquely identify one state
Country Id (FK)	This table store the Country Id which uniquely identify one country
County Id(FK)	This field stores the county Id which uniquely identify one county
address Type Id	This field stores the users communication type . One user Can have many email addresses like Official , Residence , personal etc

The below is the data model for address generalization hierarchy from the requirements. In order to have generalization hierarchy, we need to have the all the common attributes at Super type level and differential attributes at sub type level

So we have the following tables
STS_Address -> Stores the address common attributes
STS_user address -> Stores the user specific address's attributes
STS_Subejct address -> Stores the subject specific address's attributes
STS_Organization_address -> Stores the organization specific address's attributes

In order to work the generalization we need common attributes at the top . So we need to move the entire attributes from subtypes to super type except the FKs like subject id , organization id , user seq id .If we move the entire attribute list to super type the model looks like below

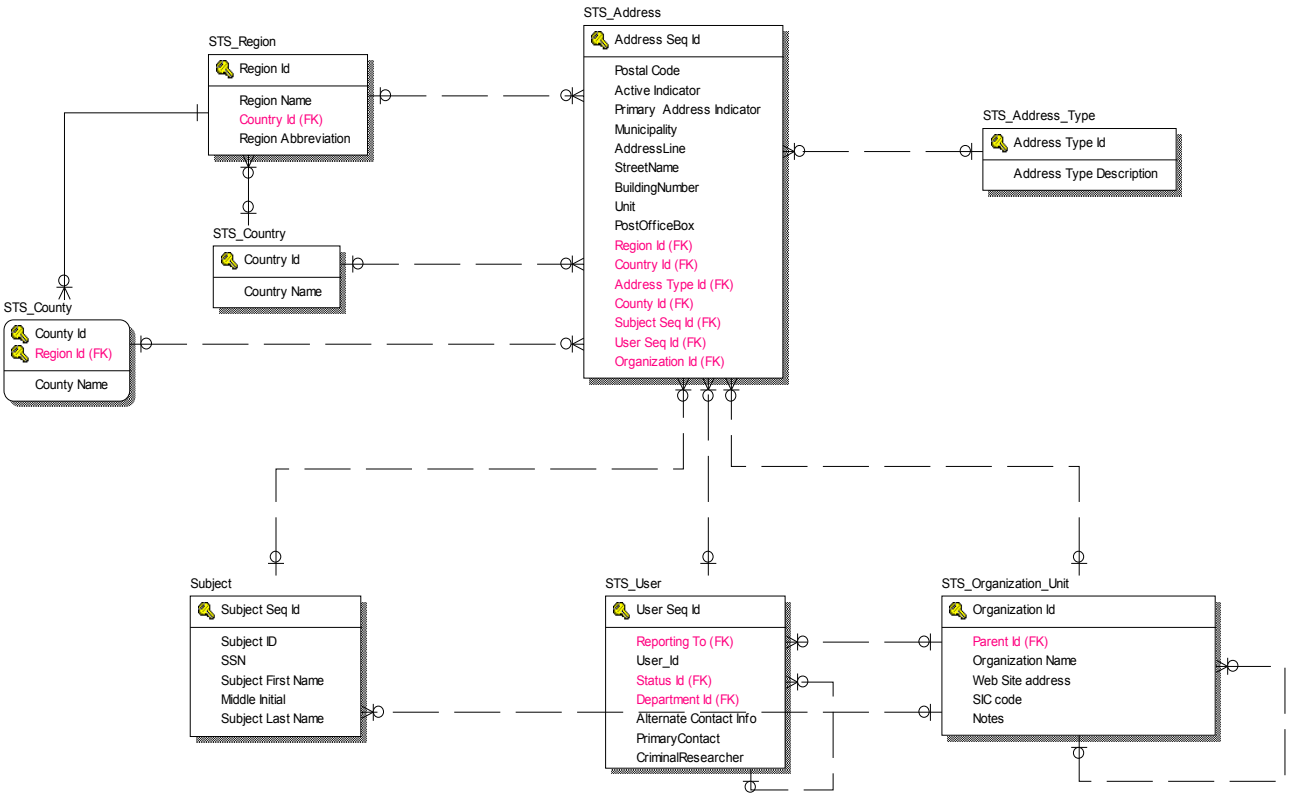


NOTE
The subject Seq id , User seq Id , And organization Id are not nulls

There are two type solutions to resolve super type sub type transformation
Rolling Up and Rolling Down

Since we have only one column at sub type level, we need to either roll up or roll down .
First we try and roll up

4.4.1 Address Hierarchy Rolling Up



Now look at FKS like organization ID , Subject ID , User ID all are nullable . The FKS will be active one at time.

Here are the advantages of this approach:

Advantage	Description
Only a single SQL statement is needed for read, update, and delete actions	In a high transaction environment, this can pay big dividends.
Good performance on full table scans with super and sub type columns	The table row length is the longest of all solutions, but no join is required.
Less database objects to maintain, simpler SQL	Not a database performance issue per se, but worthwhile to mention.

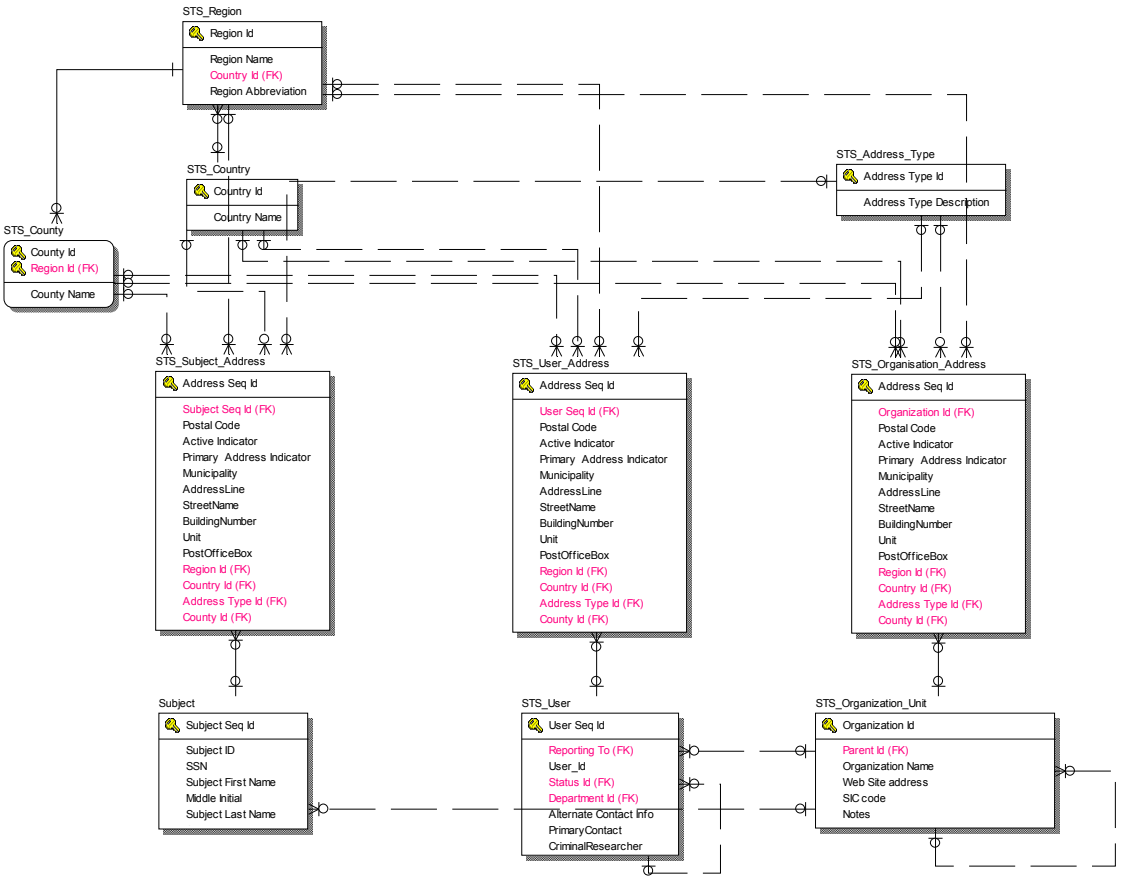
Here are the disadvantages:

Disadvantage	Description
Column optionally lost using NOT NULL constraint	As mentioned, I need to use a database check constraint or program code to enforce column optionality.
Poor performance for full table scans on super type columns	Since I added all of the sub type columns to the super type, I've degraded the performance due to the increase in row length.
Physical data model doesn't reflect business rules.	This can be circumvented if the logical model is maintained
If there is a relationship that only applies to one type of Employee, it is impossible to reflect it in	If the sub type is a parent, a foreign key with a trigger on the child table is required where the

the model and difficult to enforce	trigger validates the discriminator value. If the sub type is a child, an optional non-identifying relationship is required (even if the relationship is mandatory for the sub type) and a check constraint on the employee table is necessary.

4.4.2 Address Hierarchy Roll Down

Now we roll down and see what is the result. Please see below diagram



Now you can see separate tables has been created for storing each type of address

I don't need to use database triggers, check constraints, or application code to enforce the NOT NULL column constraints. The downside of this approach is that I've redundantly stored my super type columns in each of my sub type tables. Instead of storing one fact one place, I now store the same fact for each sub type table. That said, there are performance advantages to this model approach.

Advantages

Advantages	Description
Only a single statement needed for insert, update, delete, and read	Remember, subtypes are normally mutually exclusive, so only one sub type will apply
Fair performance for full table scans on super type columns	Not as bad as all the columns in one table as in Option 2, but not as good as Option 1
Good performance for full table scan with super and sub type columns	Smaller row length outperforms Option 2
Column Optionally maintained	My personal favorite as the Data Modeler
Easy maintenance	For a new sub type simply create a new table

Disadvantages:

Advantages	Description
Redundant attributes, relationships	More opportunity to introduce modeling errors over time
If there are relationships that apply to all employees, they must be created redundantly (one for each subtype)	This can add an order of magnitude of complexity to the design and implementation.

4.5 Factors behind Rolling Up/Down Address Hierarchy Decision

By using rolling down approach we don't need to use database triggers, check constraints, or application code to enforce the NOT NULL column constraints. The downside of this approach is that I've redundantly stored my super type columns in each of my sub type tables. Instead of storing one fact one place, I now store the same fact for each sub type table. That said, there are performance advantages to this model approach.

The following are also factors influenced to roll down the Address hierarchy

1. Data Volume
2. Properly Normalized at 3 NF (Nullable FKS depend on the address stored)
3. Better manageability
4. Better Performance

4.5.1 Data Volume

The Data Volumes of each address tables given below

Table Name	Data Volume
User Address	< 5000
Organization Address	<5000
Subject Address	>Millions

If you look at the above data volume you see that subject address is having maximum number rows compared with other two tables. So its better to have separate table to store the address

4.5.2 Properly Normalized at 3 NF

If you see the data model at rollup design, the one table for everything approach fails third normal form. The table is not properly normalized, since we have attributes like organization id , Subject id , user Id populated depend on the type of sub type.
There is no need for keeping that FKS which is nullable. Further more you need an addition attribute like address_entity_type to determine which entities data being populated (to have join operation with either organization , user or subject).
The attributes inherited into the parent from the children rely on both the key and the determinant. For example, User Address relies on the address Seq ID for uniqueness and the address_entity_type for its existence. So it's second normal form at best as it fails "nothing but the key".The option of Nullable FKS depend on the condition is not at all a good design pattern

If you look at roll down approach, you can see its properly normalized at 3 NF

4.5.3 Better manageability

Its better to store each entities address details in separate table in order to manage better. No triggers. No constraints, no complex code, easy to understand.

4.5.4 Better performance

If you see the data volumes of each address table, you can see that if you keep all the address in one table it can have a performance night mare if you join user address to user or organization to organization address

5 NORMAL FORM ERRORS

5.1 First Normal Form (1NF)

Kent's definition for 1NF, *"all occurrences of a record type must contain the same number of fields*," is meant to exclude variable repeating fields and groups. A common data modeling error is failing to recognize when a specific attribute of an entity is multivalued. In Figure 1, the STUDENT entity includes the *Major* field of study attribute. While many textbooks depict this attribute as single-valued, *Major* is actually a multivalued repeating field since a small number of students may elect to double-major while attending college. This is an example of a mis-modeled problem domain error. That portion of Figure 1 related to the STUDENT entity would yield the erroneous table structure given below. Note that the primary key field(s) is indicated by underlining.

STUDENT (StudentId, SName, Major)

If the systems analyst recognizes that *Major* is multivalued, then a simple transformation rule is applied -- the repeating field is removed to a second table along with the primary key field from the STUDENT table to form a composite key in the new table. The proper schema definition yields two tables.

STUDENT (StudentId, Sname)

STU_MAJ (StudentId, Major)

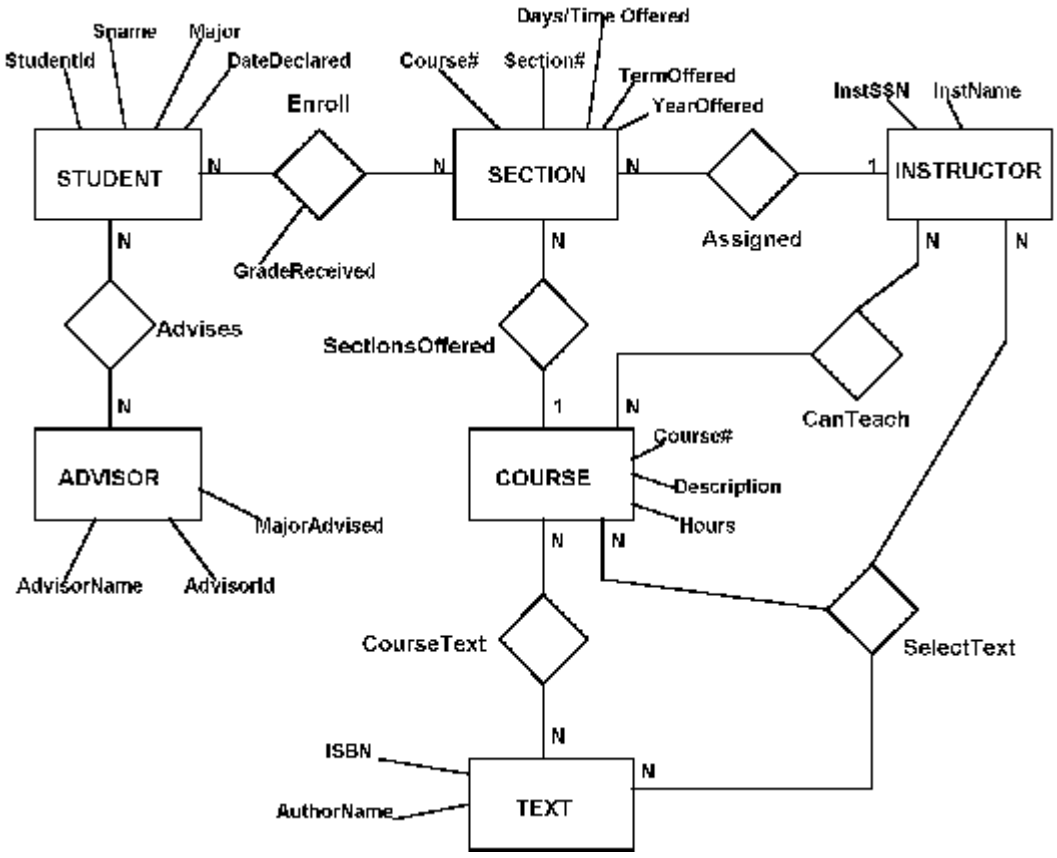


Figure 1. Example Conceptual Model of a University

Extending this data modeling problem, there may be situations where repeating fields are related, as is the case where the university tracks not only each student's major, but also the date the major was declared (*Date Declared* attribute). Again, an erroneous E-R model would yield a single table structure as given below.

STUDENT (StudentId, Sname, Major, DateDeclared)

Normalizing this table requires the systems analyst to understand which fields determine which other fields. Here, the combination of *StudentId* and *Major* uniquely determines values for the *DateDeclared* field. Note that the combination of *StudentId* and *DateDeclared* do not determine *Major* since a student may declare two majors on the same date. Again, the proper schema definition yields two tables. The solution procedure is easily extended to several sets of repeating fields with each set of repeating fields requiring an additional table structure.

STUDENT (StudentId, Sname)
STU_MAJ (StudentId, Major, DateDeclared)

5.2 Second Normal Form (2NF)

Kent combines the definition for 2NF with that for third normal form (3NF), *"a nonkey field must provide a fact about the key, the whole key, and nothing but the key*." Additionally, a table must satisfy 1NF. Violations of 2NF occur when a table's primary key is a composite of two or more

attributes, and the table contains a non-key field that is not fully determined by the composite primary key, thus, a *non-key field is not a fact about the whole key*.

E-R modeling errors related to 2NF arise whenever the E-R diagram represents a restricted or incomplete view of the problem domain, or a grossly mis-modeled E-R diagram. This error is most likely to occur when a binary many-to-many relationship has been mis-modeled, because this type of relationship yields table structures with composite keys. This type of error can also occur for ternary and higher order relationships, and can also occur where the systems analyst has modeled the relationship as a *gerund,* i.e., an abstract object which is at the same time an entity and a relationship. Ternary and higher order relationships are covered later in this article.

Consider the enrollment of students in sections of a course as is depicted by the Enroll relationship in Figure 1. Figure 2 gives a restricted model of enrollment data where the systems analysts has developed a view of the problem domain that consists of a single entity named ENROLL. This model fails to recognize that the STUDENT and SECTION entities exist. The transformation of Figure 2 to a relational schema yields a single table.

ENROLL (StudentId, Course#, Section#, TermOffered, YearOffered, SName, Grade, Days/TimeOffered)

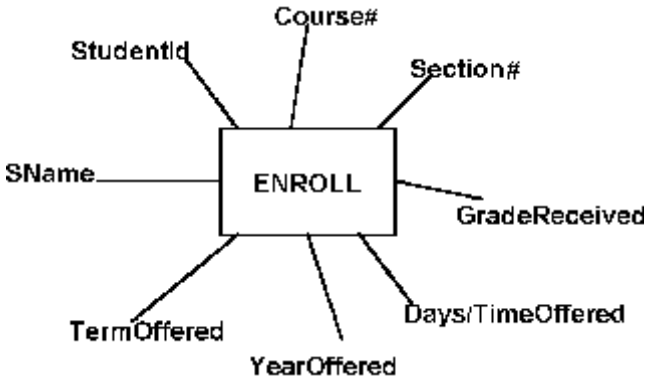


Figure 2. Second Normal Form Modeling Error

The table violates 2NF since the attribute *StudentId* determines *Sname* while the composite of *Course#, Section#, TermOffered***,** and *YearOffered* determine *Days/TimeOffered*. Only the attribute *Grade* is fully, functionally dependent on the primary key. A complete view of the problem domain as shown in Figure 1 would yield the same three table structures that the application of normalization theory will produce by applying the simple transformation rules for transforming a binary many-to-many relationship to a relational schema.
STUDENT (StudentId, Sname)
ENROLL (StudentId, Course#, Section#, TermOffered, YearOffered, Grade)
SECTION (Course#, Section#, TermOffered, YearOffered, Days/TimeOffered)

5.3 Third Normal Form (3NF)

Data maintenance problems associated with 3NF arise only when a table contains two or more nonkey fields, and one of these nonkey fields determines another nonkey field, or as Kent' states, *"a nonkey field is a fact about another nonkey field."* E-R diagrams that fail to capture information properly about binary one-to-many relationships can result in table structures that violate third

normal form. This is similar to the second normal form problem in that the E-R diagram represents a restricted or incomplete view of the problem domain. The typical error here is for the systems analyst to model a one-to-many binary relationship between two entities as a single entity.

Figure 1 gives a one-to-many relationship named Assigned that relates occurrences of the SECTION and INSTRUCTOR entities. Figure 3 gives an incorrect model of the relationship where all attributes are attached to the SECTION entity, and the INSTRUCTOR entity is not modeled. A transformation of Figure 3 yields the table:

SECTION (Course#, Section#, TermOffered, YearOffered, Days/TimeOffered, InstSSN, InstName)

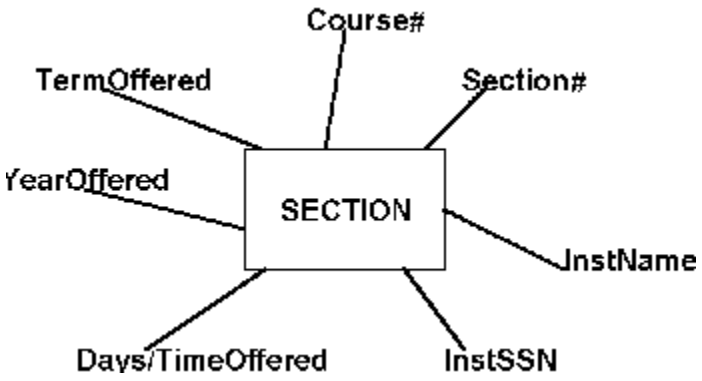


Figure 3. Third Normal Form Modeling Error

The table violates 3NF since the attribute *InstSSN* determines *InstName*, and neither field is key. The correct E-R diagram shown in Figure 1 yields two tables with the *InstSSN* field in the SECTION table serving as a foreign key link.

SECTION (Course#, Section#, TermOffered, YearOffered, Days/TimeOffered, InstSSN)

INSTRUCTOR (InstSSN, InstName)

Boyce-Codd Normal Form (BCNF)

BCNF is often referred to as a more stringent definition of third normal form. In fact, Kent's intuitive definition of second normal form and third normal form also encompasses BCNF. It is also helpful to examine the formal definition of BCNF in order to differentiate this more stringent definition of 3NF from Codd's original definition of 3NF. The formal definition can be found in any textbook on the topic: *a table is in BCNF if every determinant is a candidate key*. This rather non-intuitive definition simply means that data maintenance problems can arise whenever a table has more than one field or combination of fields that can serve as the key identifier for the table.

Figure 4 depicts a model which violates BCNF. This particular modeling problem is found in both McFadden and Hoffer (1994) as well as Kroenke (1995). The model in Figure 4 gives a restricted view of what should be modeled as a binary many-to-many relationship named Advises that links occurrences of the STUDENT and ADVISOR entities (see Figure 1 for the correct E-R diagram

for the Advises relationship). Additionally, the model given in Figure 4 fails to recognize that the *Major* attribute is multivalued. The relational transformation of Figure 4 yields a single table structure with *StudentId* and *Major* as a composite primary key since this combination determines the *AdvisorId* attribute.

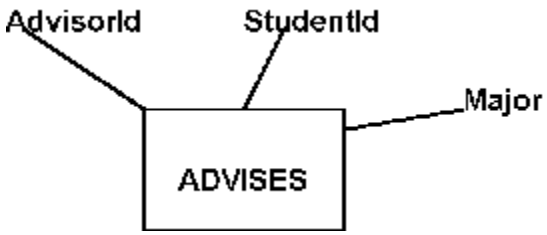


Figure 4. Boyce-Codd Normal Form Modeling Error

ADVISES (StudentId, Major, AdvisorId)
Assuming that advisors provide academic guidance for only a single major, there are two functional dependencies:

StudentId + Major -> AdvisorId
AdvisorId -> Major

Note that no single attribute determines the other two attributes; further, the selection of *StudentID + Major* as the primary key for the table still results in potential data maintenance problems (see McFadden & Hoffer for a more detailed explanation). BCNF is achieved by moving the attribute that is a determinant, but not a candidate key to a separate table; in this case, *AdvisorId.*

ADVISES (StudentId, AdvisorId)
ADVISOR (AdvisorId, Major)

It is interesting to compare the above solution to that obtained by applying standard transformation rules to the Advises binary many-to-many relationship shown in Figure 1. The transformation of Advises and the associated entities in Figure 1 yields four relational tables as given below. Note that this solution subsumes the above table structures, and recognizes that the *Major* and *DateDeclared* attributes for the STUDENT entity are multivalued (see the section on 1NF above). Further, the BCNF problem is avoided when the data model is complete with respect to this relationship.

STUDENT (StudentId, Sname)
STU_MAJ (StudentId, Major, DateDeclared)
ADVISES (StudentId, AdvisorId)
ADVISOR (AdvisorId, AdvisorName, MajorAdvised)

5.4 Fourth Normal Form (4NF)

To this point we have examined data modeling errors that have dealt with, at most, two entities. 4NF errors are more rare than those discussed above because they result from situations where a relational table structure includes attributes drawn from three different entities, and where the facts stored in the table represent multivalued dependencies between these attributes. Kent's

definition of 4NF focuses on the multivalued dependencies: *a record type should not contain two or more independent multivalued facts about an entity*, in addition to satisfying lower normal form definitions. Consider the following example table structure from McFadden and Hoffer (1994, page 224).

SELECT_TEXT (Course#, InstSSN, ISBN)

This table attempts to capture information about the textbooks used in courses by instructors. By itself, the table gives a very restricted view of reality. Figure 1 provides an unrestricted view of the SelectText relationship which is actually a ternary many-to-many-to-many relationship among the INSTRUCTOR, COURSE, and TEXT entities. This relationship properly models the situation where a course may have several instructors, each course uses several textbooks, and the texts used for a given course are selected by the instructor. In this situation the relationship is a ternary one, and the SELECT_TEXT table structure correctly models the relationship even though its restricted view does not depict the associated base tables that would store the detailed information about courses, instructors, and textbooks. Nonetheless, no data maintenance anomalies exist for SELECT_TEXT.

Suppose, however, that the situation is changed. What if there is no relationship between the text used for a course and the instructor who teaches the course, *i.e.* the instructor doesn't select the text; rather, the instructor uses whatever text is specified by the academic department offering the course. In this situation, there are two multivalued dependencies among the data elements, but no determinants, and the *InstSSN* and *ISBN* attributes are independent:

Course# ->> InstSSN
Course# ->> ISBN
InstSSN <--/-->> ISBN

For each Course# , there is a well-defined set of instructors who teach the course and a well-defined set of textbooks used for the course. Modeling this as a ternary relationship, as is done with the SELECT_TEXT table, can lead to difficulties in updating the table as depicted in Table 1a, and represents a mis-modeled problem domain error. Here three instructors teach MNGT 444. A departmental decision to add a textbook with ISBN 0-9999-9999-9 to the Management 444 course requires adding three rows to the table, one for each instructor, because the textbook is used by all instructors who teach Management 444. Adding a row for only one of the Management 444 instructors would imply that there is a relationship between instructor and textbook when no such relationship exists.

The SELECT_TEXT table is converted to 4NF by dividing the table into the two new tables given below and depicted in Table 1b. These two new tables recognize the independence between instructors and textbooks. Note that this is very similar to the situation in 1NF, except here there are two independent multivalued facts about courses.

Table 1a.

Course#	InstSSN	ISBN
IS 402	444-44-4444	0-1111-1111-1
IS 402	555-55-5555	0-1111-1111-1
MNGT 444	111-11-1111	0-2222-2222-2
MNGT 444	222-22-2222	0-2222-2222-2
MNGT 444	333-33-3333	0-2222-2222-2
MNGT 444	111-11-1111	0-9999-9999-9
MNGT 444	222-22-2222	0-9999-9999-9
MNGT 444	333-33-3333	0-9999-9999-9

TABLE_A (Course#, InstSSN)

Course#	InstSSN
IS 402	444-44-4444
IS 402	555-55-5555
MNGT 444	111-11-1111
MNGT 444	222-22-2222
MNGT 444	333-33-3333

TABLE_B (Course#, ISBN)

Course#	ISBN
IS 402	0-1111-1111-1
MNGT 444	0-2222-2222-2
MNGT 444	0-9999-9999-9

Interestingly, a more complete E-R model that captures the data relationships properly avoids the potential 4NF problem. The proper E-R model for the situation where INSTRUCTOR and TEXT are independent is depicted in Figure 1 by two relationships, CanTeach and CourseText. A transformation of the CanTeach and CourseText binary relationships and their associated base tables yields the following relational table schema. Note that the CAN_TEACH and COURSE_TEXT tables are identical to those produced above by normalizing the SELECT_TEXT table.

INSTRUCTOR (InstSSN, InstName)
CAN_TEACH (Course#, InstSSN)
COURSE (Course#, Description, Hours)
COURSE_TEXT (Course#, ISBN)
TEXT (ISBN, AuthorName)

5.5 Fifth Normal Form (5NF)

Like 4NF, errors associated with 5NF deal with multi-valued facts such as those that arise with ternary relationships or situations that look like ternary relationships. Kent notes that *5NF deals with cases where information can be reconstructed from smaller pieces of information which can be maintained with less redundancy*.
Consider the example depicted in Figure 5a and 5b. Instructors can teach for any department, departments offer courses, and instructors can teach many courses, though not necessarily every course. Figure 5a depicts the situation where we want to track which instructors are currently teaching courses for specific departments. The ternary relationship is only appropriate if instructors can be assigned to multiple departments simultaneously, can teach multiple courses, and some courses are offered by more than one department, *e.g.* the course IS 100 is offered by both the Department of Management Information Systems and Department of Computer Science. The binary relationships depicted are inadequate to capture this information. The relational table structure includes tables for the entities as well as the binary and ternary relationships.
Table 1. Example SELECT_TEXT Table Data.

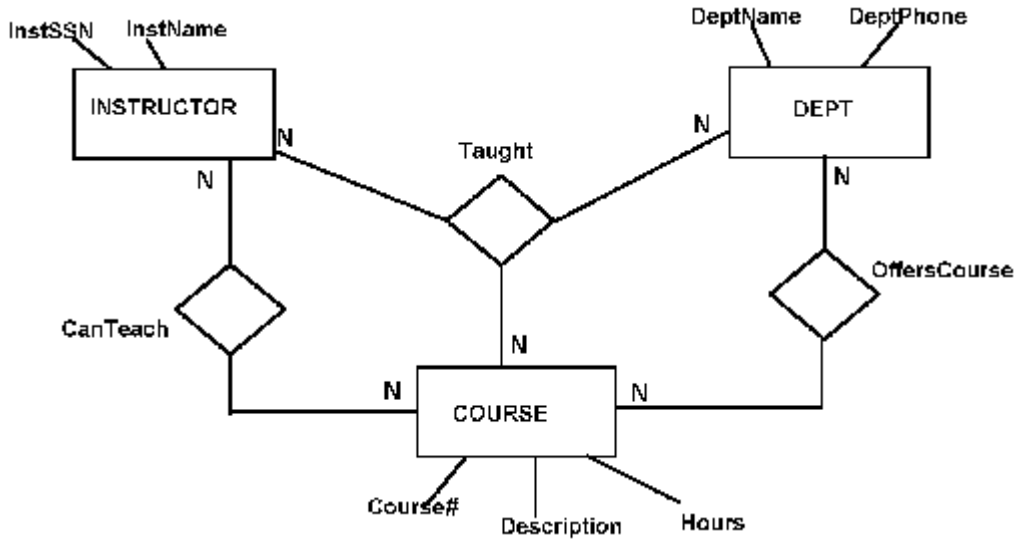


Figure 5a.

Figure 5. Example 5NF E-R Diagram.

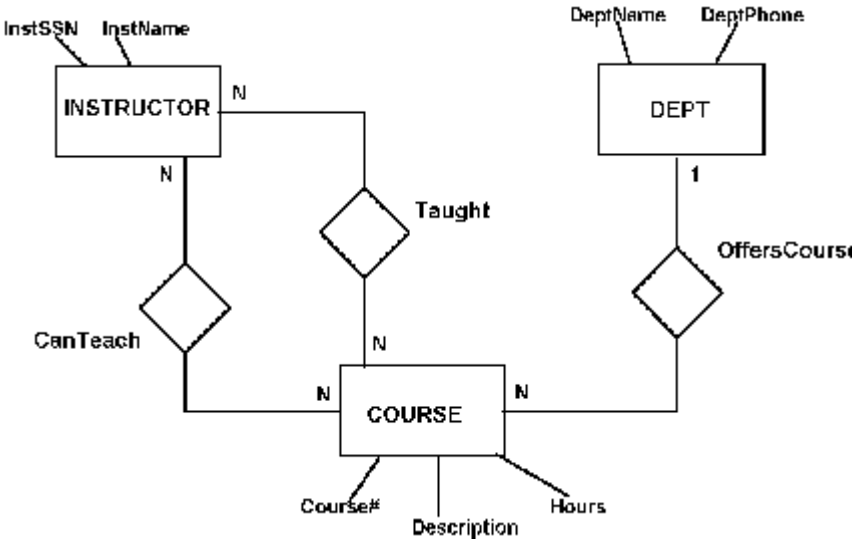


Figure 5b.

INSTRUCTOR (InstSSN, InstName)
COURSE (Course#, Description, Hours)
DEPT (DeptName, DeptPhone)
TAUGHT (InstSSN, Course#, DeptName)
CAN_TEACH (InstSSN, Course#)
OFFERS_COURSE (DeptName, Course#)
Suppose, however, that the situation changes only slightly, and that courses belong to only a single department. Now the appropriate E-R diagram is that given in Figure 5b, and the relational table structure is:

INSTRUCTOR (InstSSN, InstName)
COURSE (Course#, Description, Hours, DeptName)
DEPT (DeptName, DeptPhone)
TAUGHT (InstSSN, Course#)
CAN_TEACH (InstSSN, Course#)

The simplified table structure enables the identification of which instructor taught which course for which department since courses only belong to a single department. This is accomplished through the TAUGHT table and the placement of *DeptName* as a foreign key in the COURSE table.

6 KEYS TO AVOIDING E-R MODELING ERRORS

The first key to avoiding data modeling errors is to understand the type of errors that can occur. Experts in data modeling tend to assess the accuracy of E-R data models by comparing portions of an E-R model diagram to various data modeling patterns they have learned through experience. Recall that there are two general classes of errors: (1) incomplete data model errors, and (2) mis-modeled problem domain errors.

First and foremost, the database designer must develop a good understanding of the problem domain. A key aspect of the modeling effort is the early identification of functional dependencies among attributes of entities, i.e., which attributes are *determinants* of which other attributes. This includes identifying multi-valued attributes, related groups of attributes, and multi-valued dependencies.
Second, the database designer must, to the extent possible, develop a complete model of the problem domain. This is best accomplished by gathering additional *views* of the problem domain through interviews with potential system users. This effort must go beyond those end-users who use the current information system. In cases where the project deadline allows sufficient time, it is advisable to model a slightly larger problem domain than that for which one envisions developing application programs. This approach helps eliminate 2NF and 3NF modeling errors that can result from mis-modeled binary relationships.

Third, develop table definitions by first converting the E-R diagram to a relational schema, then check to see if each *view* is supported. This step should be combined with normalization as a double check on *view* support and data maintainability, and on the enforceability of integrity constraints.
As a final word of caution, avoid the tendency in the information systems industry to be satisfied with normalizing only to 3NF. As was demonstrated above, 4NF and 5NF problems, although rare, can occur where the problem domain is modeled inadequately.

7 GUIDELINES FOR AVOIDING REQUIREMENT CONVERSION ERRORS

Guideline	Critical Steps
1. Understand Business Requirements	1a. Identify actual relationship between entities while doing logical data modeling.
2. Understand the Business rules	2a. Use the proper verbal phrase like “should work “ 2b. Gather additional end-user views of the business rules
3. Understand the application of business rules	3a. Ask additional questions like “Must a product be offered by a company before it can be sold?”
4. No candidate for Primary key or PK can change or PK can be null able	4a. Create Alternate key along with a Sequence as PK

7.1 Validation Procedure for mis interpreted requirement Errors

Requirements	Error Type	Verification Procedure
Employee Must work in a department	Null constraints	Check the migrated attribute for not null constraints
Value List error (e.g. Gender)	Check constraints	Check the attribute for actual check constrains with a valid list of values Make it Not null
n-ary relationships error	Ternary relationships	Check the relationships. Converts into Binary relationship if the requirements demands Ask Additional questions “Must a product be offered by a company before it can be sold?”
Usage of sequences for Primary Key	Assignment of PK	Check for Alternate keys and create alternate key
Nullable Fks	Performance	Ask users “ is they really requires nullable columns” try alternate way like creating a unknown record in parent entity and use that for all nullable records.

8 GUIDELINES FOR AVOIDING NORMALIZATION ERRORS

This table summarizes the various normalization errors, classifies the errors as resulting from incomplete data models or mis-modeled problem domains, and gives guidance for verifying the accuracy of an E-R model diagram. This table identifies the various data modeling patterns to be checked as you verify an E-R diagram.

Guideline	Critical Steps
1. Understand the problem domain.	1a. Identify functional dependencies among attributes (determinants) during logical data modeling. 1b. Identify multivalued attributes, and related groups of attributes.
2. Develop a sufficiently comprehensive conceptual model.	2a. Gather additional end-user views of the data. 2b. Model a slightly larger problem domain than that required for the project at hand.
3. Use a CASE tool to develop a relational schema (physical model) from the E-R diagram.	3a. Determine if each view is supported by the relational schema-use a quick QBE query generator and report generator as a double-check of view support. 3b. Use normalization as a double-check on view support and data maintainability. 3c. Normalize through 5NF.

8.1 Validation Procedure for Normalization Errors.

Normalization Error	Error Type	Verification Procedure
1NF - Attribute is multivalued.	Mis-modeled problem domain - attribute modeling error.	Check all attributes to ensure they are single-valued.
1NF - Set of attributes are multivalued and related to one-another.	Mis-modeled problem domain - multiple attribute modeling errors.	Check attributes identified as multivalued to determine if they are related to one-another.
2NF - Partial Key Dependency Errors.	Mis-modeled problem domain and/or incomplete data model - binary M:N modeling error.	Check entities with composite identifiers to ensure they are not, in fact, more than one entity - identify functional dependencies among the attributes.
3NF - Transitive Key Dependency Errors.	Mis-modeled problem domain and/or incomplete data model - binary 1:M modeling error.	Check entities with multiple non-identifier attributes for functional dependencies among the attributes.
BCNF - Determinants not serving as candidate keys.	Incomplete data model - binary M:N modeling error.	Check entities with composite identifiers, especially where identifier attributes appear to belong to multiple entities.

4NF - Entity has independent multivalued facts.	Mis-modeled problem domain - modeling a relationship as ternary that should be two or more binary relationships.	Check all ternary relationships for multivalued dependencies and an absence of determinants, and multivalued attributes that are independent of one-another.
5NF - Inability to join tables for binary relationships to form a necessary data view.	Mis-modeled problem domain - modeling relationships as two or more binary relationships where the data should be modeled as a ternary relationship.	Check original views of the database that were used in developing the E-R model, especially those involving three or more entities, for the ability to join entity attributes to form the original view.

8.2 Validation Procedure for Roll Up/Roll Down

The decision roll up and roll down depend on the actual requirements and the decision can vary with cases. You need to take care of the factors described earlier to arrive at the decision

9 APPENDIX –A (REFERENCES &COURTESY)

http://en.wikipedia.org
http://www.developer.com
http://ibm.com
http://www.gatherspace.com/
http://it.toolbox.com/
http://www.srnr.arizona.edu/