How to transform a conceptual model to a logical model:

Codd was the created of Relational (Logical) Model in 1970. The primary distinction between the Conceptual Model and the Relational Model is that the Conceptual Model focuses on entities, whereas the Relational Model focuses on tables. The Relational Model's tables have a fixed number of columns but an infinite number of rows, or Tuples. A Tuples set is a record. The attributes are represented by the columns, and each attribute must have a domain.

Let’s compare it with conceptual and then I will explain it rules to keep in mind when you are moving to logical model:

1. Entities: An entity type is converted into a table and normally retains the same name.

Entities have an Attribute or Attributes that make up the Primary Key (PK); if more than one attribute makes up the PK, this is referred to as a Composite Key (CK). Weak entities contain their parent entity's as Foreign Key (FK).

1. Attributes : Attributes are converted into columns in the table. The entity's key characteristic in the Conceptual Model is likely to become the Relational Model's Primary Key (PK).Multi-value Attributes are transformed into tables, and the parent entity is transformed into a Foreign Key (FK) in the table.
2. Relationships: Because there are no relationships in the relational model, relationships with attributes will become tables. In one-to-one relationships, the PK of one entity becomes the FK of the other, with the designer having to decide which one.

In a one-to-many relationship, the ‘many entity' accepts the PK from the 'one entity' as an FK.

1. Participation: In most cases, the PK on the mandatory participation side becomes the foreign key on the optional participation side.
2. Superclass/Subclass: When dealing with Superclass and Subclass, there is no one-size-fits-all answer. The following are three possible solutions to the problem:
3. In the relational diagram, the superclass becomes a table with the attributes shared by all subclasses, and each subclass becomes a table with its unique attributes.
4. Remove the superclass entirely from the relational design; instead, each subclass becomes a table with all common and unique attributes.
5. The superclass is transformed into a table that contains all common and unique properties of each subclass.

Clearing the rules to in more simple way:

Each entity on the conceptual ERD is transformed into a relation.

In the corresponding relation, each attribute becomes an attribute.

Multiple attributes are created when composite attributes are combined.

The displayed candidate key may or may not become the primary key.

The relation should be named the same as the entity, the key(s) should be displayed, and all other properties should be listed.

The foreign key is inserted in either relation in a one-to-one relationship.

The primary key of the one relation is placed in the many relation in a one-to-many relationship.

An associated entity is generated in a many to many connections to connect the two relationships.

Because attributes cannot exist in relationships, a new relationship must be developed.

A new table is built to avoid null values in a partial participation to complete participation connection where the relationship is one to many with the many connection having partial participation.

History of CROW’s Foot notation:

The origins of crow's foot notation may be traced back to an article by Gordon Everest (1976, Fifth Computing Conference, IEEE). The naming convention for notations was changing; in fact, it had been evolving for some years.

When I questioned Mr. Everest about the situation, he stated:

To distinguish it from Bachman's notation, I named it the "inverted arrow." at the time. It was more appealing to me than the arrow since it did not suggest directivity or a physical access channel, and it was aesthetically intuitive, indicating many. Others began referring to it as chicken feet (e.g., Carlis textbook 1) after that.

I now call it a FORK, which is short and to the point and avoids the possessive crow's or the long chicken. The focus of my original work was on “Basic data structures described using a popular example” 2 (the title, which later became chapter 4 in my McGraw Hill text, Database Management, 1986). The usage of the notation was purely coincidental, but well designed.

[X]------< [Y]

demonstrating that a single X can be related to many Ys (and each Y relates to at most one X).

An entity is a representation of a class of object. might be a person, a location, an object, or anything else. Entities are often described through characteristics. An entity is represented by a rectangle with its name on top in crow's foot notation. The name is solitary (entity), not plural (entities).

Attributes:

The identifier is the attribute(s) that uniquely differentiates an instance of the entity. This sort of characteristic is usually denoted by an asterisk.

Relationship:

Relationships depict the connection between two entities.

They are shown as a straight line.

On the relationship line, each connection is usually given a name that is articulated as a verb.

This specifies the type of relationship that exists between the things.

Cardinality:

There are two signs in relationships.

These are shown on both sides of the line.

The first (commonly referred to as multiplicity) refers to the maximum number of times an instance of one entity can relate to instances of another entity.

It might be one or many.

The second specifies the smallest number of times one instance can be linked to another.

It might be zero or one, and it indicates whether the connection is optional or necessary.

These two signs are always combined in a precise order.

The emblem of multiplicity appears initially on the relationship's outskirts.

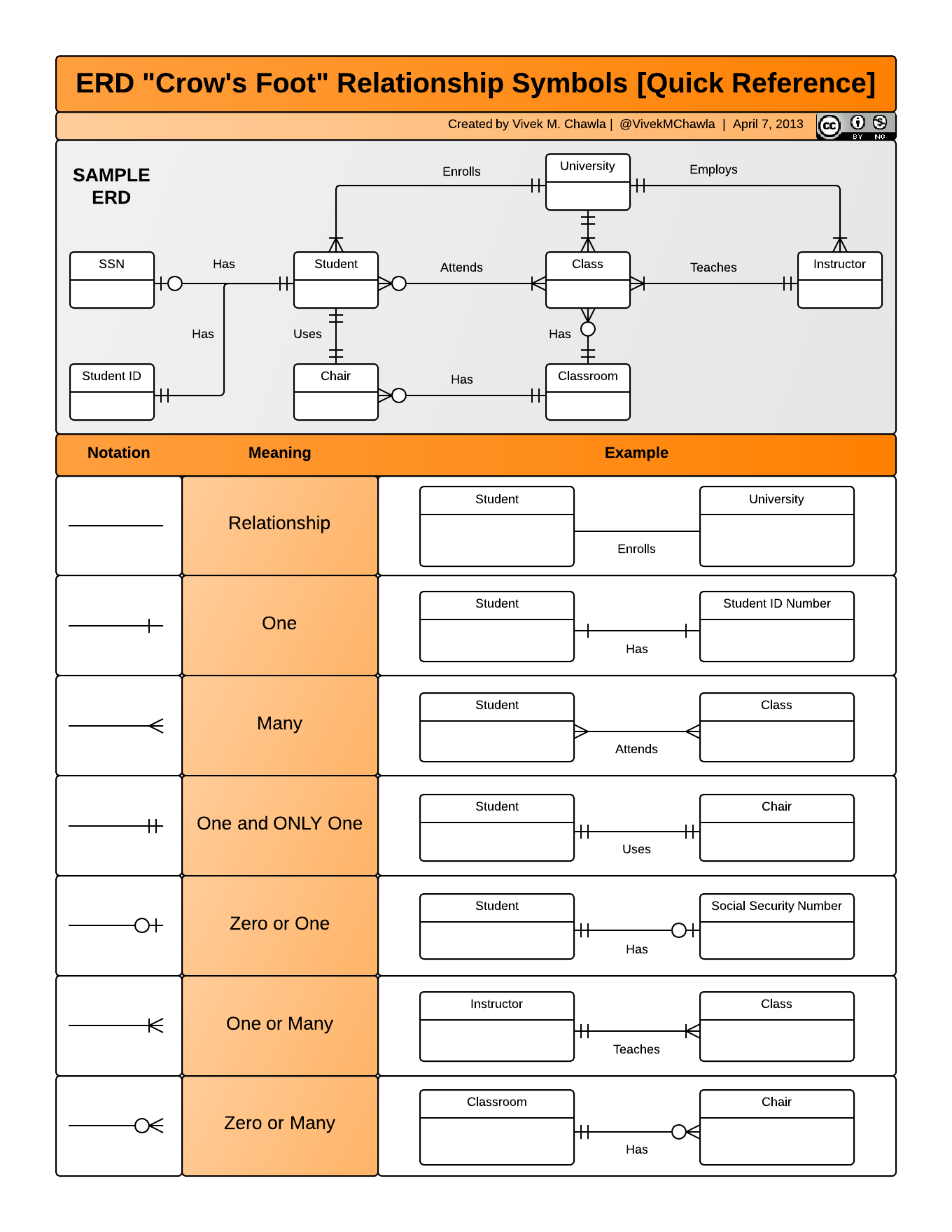
Following the multiplicity sign comes the sign indicating whether the connection is necessary or optional.

The following is written in crow's foot notation:

A straight line perpendicular to the relationship line represents a multiplicity of one and a mandatory relationship.

The three-pronged ‘crow-foot' emblem represents a multiplicity of many.

An empty circle represents an optional connection.



Why we use cross foot notation:

The logical modelling diagram is created to improve both the system and its visual appearance. The relational model describes the events that occur and identifies the data that is required. Another advantage of creating the relational diagram is that it lays the groundwork for the building of the Physical diagram.

Logical diagrams are frequently used to help non-technical business teams grasp the database. This is incredibly valuable to the development team because it allows them to engage the business team in the process. This can be incredibly valuable since it helps the business team to root out assumptions mistakes and discover any flaws in the database architecture.

HOW:

In some aspects, the transition from Conceptual to Relational can be relatively simple.

A significant amount of system visualization has been accomplished, and as a result, it can offer a significantly clearer grasp of what the business outcome is, which is an asset shared by the teams involved.

The goal is to follow the guidelines given above and understanding where judgments about which option to go must be made. This is especially true for more abstract features like multi-valued characteristics, relationships with their own characteristics, and the transition from a relationship structure to Primary Keys and Foreign Keys.

Normalization:

Database normalisation is a method for structuring data in a database. Normalization is a method of dissecting tables in order to minimise data redundancy (repetition) and undesired features such as Insertion, Update, and Deletion Anomalies. It is a multi-step procedure that converts data into tabular format while also deleting duplicate data from relation tables.

Normalization serves primarily two functions.

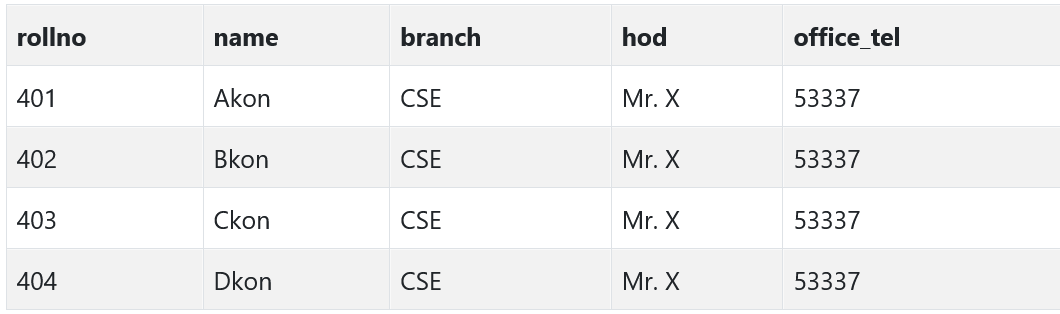
1)Data that is redundant (useless) is removed.

2) Assuring that data dependencies make sense, i.e. that data is stored properly.

Example:

Problems Due to a Lack of Normalization:

If a table is not correctly normalised and has data redundancy, it will not only use extra memory space, but it will also make it difficult to manage and update the database without experiencing data loss.

Let's look at a Student table to see how these anomalies work: 

We have data from four Computer Science students in the table above. As we can see, data for the fields branch, hod (Head of Department), and office tel are duplicated for students in the same branch at the institution; this is known as data redundancy.

**Insertion Anomaly**

I Assume that for a new admission, until and until a student chooses a branch, the student's data cannot be added; otherwise, the branch information must be set to NULL.

Furthermore, if we must input data for 100 students from the same branch, the branch information will be duplicated for all 100 students.

These possibilities are only Insertion anomalies.

Updating anomaly:

What if Mr. X drops out of college? Or is no longer the department head of computer science? In such instance, all student records must be updated, and if any record is missed by accident, data inconsistency would result. This is an update anomaly.

Delete Anomaly:

In our Student table, we keep two types of information together: Student information and Branch information.

As a result, if student records are removed at the conclusion of the academic year, we will also lose branch information.

This is known as the Deletion Anomaly.

**Normalization Rule**

Normalization rules are divided into the following normal forms:

1. First Normal Form: It should only contain attributes/columns with single (atomic) values.

The values put in a column should all belong to the same domain. Each, column in a table should have a distinct name. Furthermore, the sequence in which data is saved is unimportant.

The first typical form expects you to create your database in accordance with a few simple guidelines, which are as follows:

Rule 1: Attributes with a Single Value

Your table's columns should be single-valued, which means they should not have multiple values.

Rule 2: Attribute Domain should not be modified. This is a rule of "Common Sense."

The values entered in each column must be of the same sort or type. It should just save the 'date of birth' for all records/rows.

For example, if you have a column dob to store the dates of birth of a group of people, you cannot or must not preserve the 'names' of some of them in that column with the 'dates of birth' of others.

Rule 3: Each Attribute/Column must have a unique name. This rule requires that each column in a table have a distinct name.

This is selected to reduce misunderstanding while retrieving data or doing other operations on stored data. If one or more columns have the identical name, the DBMS system will be perplexed.

Rule 4: It does not matter what order you do things in.

This rule states that the order in which the data in your table is stored is irrelevant.

1. Second Normal Form:

It should be written in First Normal Form.

Single column Primary Key

It should also not be partially dependent.

What are Transitive Functional dependencies:

1. Third Normal Form

1. It is written in the Second Normal Form.

2. It also lacks Transitive Dependency.

1. BCNF:

is a more advanced variation of Third Normal Form. This form deals with anomalies that are not handled by 3NF. BCNF refers to a 3NF table that does not contain several overlapping candidate keys. The following requirements must be met for a table to be in BCNF:

1. R must be in the third Normal Form.
2. Secondly, X should be a super Key for each functional dependence ( X Y ).
3. Fourth Normal Form:

It's written in Boyce-Codd Normal Form.

It also lacks Multi-Valued Dependency.