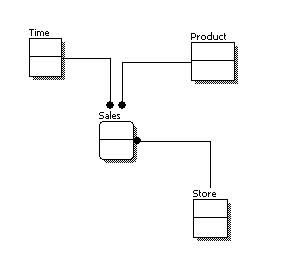
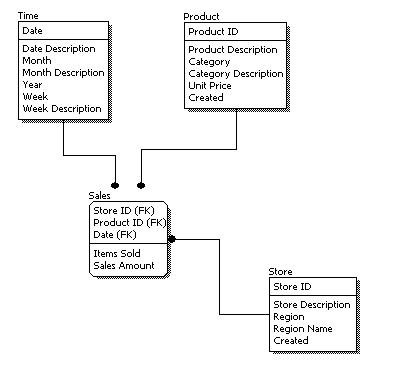
Here we compare these three types of data models. The table below compares the different features:

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Conceptual | Logical | Physical |
| Entity Names | ✓ | ✓ |  |
| Entity Relationships | ✓ | ✓ |  |
| Attributes |  | ✓ |  |
| Primary Keys |  | ✓ | ✓ |
| Foreign Keys |  | ✓ | ✓ |
| Table Names |  |  | ✓ |
| Column Names |  |  | ✓ |
| Column Data Types |  |  | ✓ |

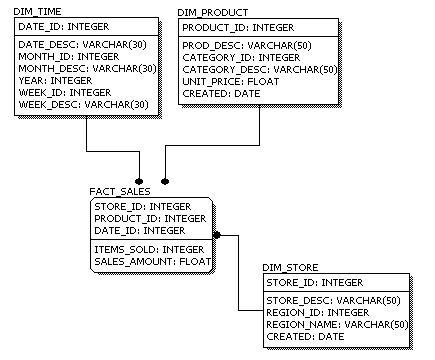
**Conceptual Model Design**



**Logical Model Design**



**Physical Model Design**



Normalization with Example

**Why do we need to do normalization?**  
  
To eliminate redundancy of data i.e. having same information stored at multiple places, which eventually be difficult to maintain and will also increase the size of our database.  
With normalization we will have tables with fewer columns which will make data retrieval and insert, update and delete operations more efficient.  
  
  
**What do we mean when we say a table is not in normalized form?**  
  
Let’s take an example to understand this,  
Say I want to create a database which stores my friends name and their top three favorite artists.  
This database would be quite a simple so initially I’ll be having only one table in it say friends table. Here FID is the primary key.

|  |  |  |
| --- | --- | --- |
| **FID** | **FNAME** | **FavoriteArtist** |
| 1 | Srihari | Akon, The Corrs, Robbie Williams. |
| 2 | Arvind | Enigma, Chicane, Shania Twain |

This table is not in normal form why?  
  
FavoriteArtist column is not atomic or doesn’t have scalar value i.e. it has having more that one value.  
Let’s modify this table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FID** | **FNAME** | **FavoriteArtist1** | **FavoriteArtist2** | **FavoriteArtist3** |
| 1 | Srihari | Akon. | The Corrs | Robbie Williams. |
| 2 | Arvind | Enigma | Chicane | Shania Twain |

This table is also not in normal form why?  
  
We have now changed our table and now each column has only one value!! (So what’s left?)  
Because here we are having multiple columns with same kind of value.  
  
I.e. **repeating group of data or repeating columns.**  
  
So what we need to do to make it normal or at least bring it in First Normal Form?

1. We’ll first break our single table into two.
2. Each table should have information about only one entity so it would be nice if we store our friend’s information in one table and his favorite artists’ information in another

(For simplicity we are working with few columns but in real world scenario there could be column like friend’s phone no, email , address and favorites artists albums, awards received by them, country etc. So in that case having two different tables would make complete sense)

|  |  |
| --- | --- |
| **FID** | **FNAME** |
| 1 | Srihari |
| 2 | Arvind |

|  |  |
| --- | --- |
| **FID** | **Favorite Artist** |
| 1 | Akon. |
| 1 | The Corrs |
| 1 | Robbie Williams |
| 2 | Enigma |
| 2 | Chicane |
| 2 | Shania Twain |

FID foreign key in FavoriteArtist table which refers to FID in our Friends Table.  
  
Now we can say that our table is in first normal form.  
  
Remember For **First Normal Form**  
  
**1...Column values should be atomic, scalar or should be holding single value**  
**2...No repetition of information or values in multiple columns.**  
  
**3...So what does Second Normal Form means?**  
  
  
 **Second normal form** our database should already be in first normal form and every non-key column must depend on entire primary key.  
  
Here we can say that our Friend database was already in second normal form l.  
Why?  
  
Because we don’t have composite primary key in our friends and favorite artists table.  
  
Composite primary keys are- primary keys made up of more than one column. But there is no such thing in our database.  
But still let’s try to understand second normal form with another example  
This is our new table

|  |  |  |  |
| --- | --- | --- | --- |
| **Gadgets** | **Supplier** | **Cost** | **Supplier Address** |
| Headphone | Abaci | 123$ | New York |
| Mp3 Player | Sagas | 250$ | California |
| Headphone | Mayas | 100$ | London |

In about table **ITEM+SUPPLIER**together form a composite primary key.  
  
Let’s check for dependency  
  
If I know gadget can I know the cost?  
  
No same gadget is provided my different supplier at different rate.  
  
If I know supplier can I know about the cost?  
  
No because same supplier can provide me with different gadgets.  
  
If I know both gadget and supplier can I know cost?  
  
Yes than we can.  
  
**So cost is fully dependent (functionally dependent) on our composite primary key (Gadgets+Supplier)**  
  
Let’s start with another non-key column Supplier Address.  
  
If I know gadget will I come to know about supplier address?  
  
Obviously no.  
  
If I know who the supplier is can I have it address?  
  
Yes.  
  
**So here supplier is not completely dependent on (partial dependent) on our composite primary key (Gadgets+Supplier).**  
  
This table is surely not in Second Normal Form.  
  
So what do we need to do to bring it in second normal form?  
  
Here again we’ll break the table in two.

|  |  |  |
| --- | --- | --- |
| **Gadgets** | **Supplier** | **Cost** |
| Headphone | Abaci | 123$ |
| Mp3 Player | Sagas | 250$ |
| Headphone | Mayas | 100$ |

|  |  |
| --- | --- |
| **Supplier** | **Supplier Address** |
| Abaci | New York |
| Sagas | California |
| Mayas | London |

We now how to normalize till second normal form.  
  
But let’s take a break over here and learn some definitions and terms.  
  
**Composite Key**: -Composite key is a primary key composed of multiple columns.  
**Functional Dependency** – When value of one column is dependent on another column.  
  
So that if value of one column changes the value of other column changes as well.  
  
e.g. Supplier Address is functionally dependent on supplier name. If supplier’s name is changed in a record we need to change the supplier address as well.  
  
S.Supplier–àS.SupplierAddress  
  
“In our s table supplier address column is functionally dependent on the supplier column”  
  
**Partial Functional Dependency –**A non-key column is dependent on some, but not all the columns in a composite primary key.  
  
In our above example **Supplier Address** was partially dependent on our composite key columns (Gadgets+Supplier).  
  
**Transitive Dependency-**A **transitive dependency** is a type of functional dependency in which the value in a non-key column is determined by the value in another non-key column.  
  
With these definitions in mind let’s move to Third Normal Form.  
**For a table in third normal form**

* It should already be in Second Normal Form.
* There should be no transitive dependency, i.e. we shouldn’t have any non-key column depending on any other non-key column.

Again we need to make sure that the non-key columns depend upon the primary key and not on any other non-key column.

|  |  |  |  |
| --- | --- | --- | --- |
| **Album** | **Artist** | **No. of tracks** | **Country** |
| Come on over | Shania Twain | 11 | Canada |
| History | Michael Jackson | 15 | USA |
| Up | Shania Twain | 11 | Canada |
| MCMXC A.D. | Enigma | 8 | Spain |
| The cross of changes | Enigma | 10 | Spain |

Although the above table looks fine but still there is something in it because of which we will normalize it further.  
  
Album is the primary key of the above table.  
  
Artist and No. of tracks are functionally dependent on the Album(primary key).  
  
But can we say the same of Country as well?  
  
In the above table Country value is getting repeated because of artist.  
  
So in our above table Country column is depended on Artist column which is a non-key column.  
  
So we will move that information in another table and could save table from redundancy i.e. repeating values of Country column.

|  |  |  |
| --- | --- | --- |
| **Album** | **Artist** | **No. of tracks** |
| Come on over | Shania Twain | 11 |
| History | Michael Jackson | 15 |
| Up | Shania Twain | 11 |
| MCMXC A.D. | Enigma | 8 |
| The cross of changes | Enigma | 10 |

|  |  |
| --- | --- |
| **Artist** | **Country** |
| Shania Twain | Canada |
| Michael Jackson | USA |
| Enigma | Spain |

**Normally this is considered enough and we don’t really go on applying the other normal forms.**  
  
**Most of real-world application has databases which are in third normal forms.**

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

## Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

## Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

## Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

## Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

## Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

## Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

## Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

## Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

## Rule 9: Logical Data Independence

The logical data in a database must be independent of its user’s view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

## Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

## Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

## Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.