Database Design

The meaning of data in a database is reflected in its structure; the way it is divided into tables and columns. The choice of which tables to create and what their columns are can severely limit the function of a database. One approach to database design, and one particularly suited to an object oriented environment, is known as **Entity/Attribute analysis**. Entity/Attribute analysis divides the data up into different classifications :- entities, attributes, and relationships.

Entities are representations of objects that exist in the real world and that we wish to describe in the database. An **entity class** is a group of similar objects and an entity is an **instance** of an entity class. Each entity class has specific **attributes** that describe its characteristics or properties. Each instance of an entity has a matching set of **attribute values**, one for each attribute of the class. Attributes are not necessarily specific to entity classes; the same attribute may occur in more than one entity class.

In an object oriented programming environment this structure works well but in a relational database the only data structure available is a table. In a relational database entity classes are always represented by tables where each entity instance is a row. The attributes of each entity class then become columns of the table so the domain of a column is the set of all possible attribute values it could contain which means the set of all possible values the attribute it represents could have. The key of the table, the thing that distinguishes one entity from another, is always itself an attribute of the entity class. So the names of tables are really the names of the entity classes they represent and the names of columns are the names of attributes.

Suppose we needed a database for a bank. One obvious entity class here is the customers. They will be represented by a table with columns for their attributes. The choice of appropriate attributes depends on what the bank needs to record about its customers. One possible interpretation is that we need the table

CUSTOMERS (NAME, ADDRESS, STATUS)

where status is used to distinguish between the sort of customers the bank keeps trying to lend money to and the sort of customers the bank keeps trying to reclaim money from. Similarly we probably have to have an entity class called accounts with a few attributes. So we might have

ACCOUNTS (ACCOUNTNO, BALANCE, OVERDRAFTLIMIT)

This simple structure is a start but it is not sufficient to describe everything people want to record in databases because it will not allow us to describe relationships between entities.

A **relationship** is a connection between two entities; it is an instance of a **relationship type**. Relationship types are like attributes in that they describe a general property of entity classes and have individual instances (relationships / attribute values) for individual entities. But the important difference between attributes and relationships is that there are two entity classes in a relationship type and two entities in a relationship. Relationships fall into three categories:

a) 1 : 1 (One to One) The relationship links a pair of entities. For example the relationship 'is married to'.

b) 1 : m (One to Many) The relationship links one entity with a group of other entities. Each of this group can only be linked to one entity. E.g. 'is the father of'.

c) n : m (Many to Many) The relationship links one entity with many other entities each of which may be linked to other entities as part of the same relationship. E.g. 'is the cousin of'.

We give names to relationship types as well as to entity classes and attributes; typically these are verbs whereas entity class names are nouns. One to one relationships can be represented in the database by having a column in the table (attribute in the entity class) that represents one side of the relationship (it does not matter which) that identifies the entity on the other side of the relationship; this column is always a foreign key. One to many relationships can be represented by the same technique except that the foreign key must always be an attribute of the entity class on the many side.

Many to many relationships cannot be represented this way. Instead they must be split up into two one to many relationships with an extra, artificial entity class represented by table called a **linker**. Each side in the many to many relationship is the one side in a one to many relationship with the linker table. The linker table has one row for each relationship instance of the relationship type and need contain nothing but the two foreign key attribute values that represent the two entities that it links. Relationship types can themselves have attributes. These are stored with the foreign key or keys that represent the relationship as a column of the table just like attributes of entities. The technique of representing relationships by foreign keys is the reason why referential integrity is so important. Foreign keys always represent a relationship and the relationship cannot exist if one side of it has disappeared.

In our bank example above there is obviously a relationship between customers and accounts. If the bank allows joint accounts then it has to be a many to many relationship. This means it must be represented by a linker so we must have a table with the keys of the two tables it is trying to link. The key of the account table is obviously ACCOUNTNO but identifying a key in CUSTOMERS is less obvious. The customer's name is not sufficient to identify them and it would be rather a waste of space to repeat the whole of the name and address for each owner of each account. This is an instance where we need to add an artificial key, one which will identify an individual customer without using the longer combination of NAME and ADDRESS. So the customers table must be extended. It has to become

CUSTOMERS (NAME, ADDRESS, STATUS, CUSTOMERNO)

And now we have our linker

ACHOLDERS (CUSTOMERNO, ACCOUNTNO)

The information used to generate tables and columns for a database in the process of entity/attribute analysis can be illustrated by **ER diagrams**. We will not cover them in this course but most standard textbooks on databases will cover them in detail.

In practice the process of splitting data into tables is not simple or clear cut. there is rarely one ideal design although there are plenty of wrong designs.

In designing the bank data structure we have made many assumptions. The structure we have designed will still work if customers only ever have one account or if accounts only have one owner although in either case it will be wasteful in that it will take more storage space than it needs. We will have problems if it turns out that customers, rather than accounts, have overdraft limits, if joint accounts sometimes but not always need two signatures, if different types of account have different interest rates or if we ever want to print out a bank statement with the transactions in the last month or quarter.

The appropriate design depends on the meaning of the data to be stored and extracted both at present and in the future. It is rarely easy to discover this, but it is vital to the success of the database.