Basic Concepts

The only valid reason for storing data using a database system is so that it can, if necessary, be retrieved later in some useful form. In order to do this people must be able to communicate with the database system in order to specify the information they want. In other fields within computer science it is necessary to learn a programming language but when dealing with database systems there are two sorts of vocabulary; one used to tell the database system to store or retrieve information and a second vocabulary to talk about specific items of data. There are many different database systems (although the difference between many of them tend to be superficial) and each of these database systems has its own language but any particular database system will also have a unique language associated with the data it is trying to manage. It is this aspect of database systems we will start with.

In a relational database the data can be thought of as a collection of related tables where each table is grid with rows and columns and where the first row gives the title of each column. The rows are not numbered. This structure has very little to do with what is actually stored; it is just a framework to use when talking about data. All the tables in a database represent data in the real world and the tables are related to each other in some way, if only because they are concerned with the activities of a particular organization. Each **table** is a collection of similar data about similar things. Each element of this collection is a **row** of the table and represents one of the things. Each row contains information arranged according to a fixed pattern. The rows in a table contain different information but it is all arranged according to the same pattern within the row although different tables will have different patterns in their rows. The pattern is expressed as a fixed number of columns arranged in a specific order each of which contain a fixed type of data. This fixed order is indicated by the titles of the columns (the first row of our imaginary table). One single cell in the table grid, a particular column in a particular row, contains a single, atomic piece of information of a fixed type and about a fixed subject. Saying information is **atomic** means that it cannot be further subdivided. The consequence of this is that cells in the table cannot contain lists.

The fixed type of data each column can contain is its **domain**. Several different columns, possibly in different tables, can have the same domain but a particular column in a particular table has exactly one domain. A domain can be thought of as the set of all possible values a column can contain. The domain can be very precisely specified (e.g. M or F for a column called Sex) or very loosely specified (e.g. a string of up to 400 characters including line breaks as the domain of a column called Address).

The terminology of tables, rows and columns is relatively new and is not universally used in database systems but the image of a table as something to store data in is fundamental to all relational systems. Tables are beginning to supersede an older terminology of files for tables, records for row and fields for columns which was derived directly from punched cards and is still widely used, and then there are relations, tuples and attributes which we will come to later.

Each table in a database has a name and each column of a table also has a name as well as a domain. The combination of a table and a column name is sufficient to identify a complete column. This is a group of values of the same domain, one from each row of the table. The actual number of values identified by a table and a column name varies as it depends on the number of rows in the table at the time.

The rows within a table may or may not appear to the users to be ordered. In any case they are identified by values stored within the rows and not by their position within the table. Similarly when they appear in order the order is decided by the data within the rows. For example, in a table containing data about

people where each row contains data about one individual a row may be identified by the name of the person it describes and the rows may be ordered alphabetically in name order. Thus rows within a table are identified by **key**. A key is a column (or group of columns) of a table which contains a different value in each row and thus can be used to identify rows individually. Keys need not be a single column, they can consist of any number of columns of the table. The key column (or columns) must always contain a different value (or set of values) in every row of the table for every possible row that could ever be added to the table.

Sometimes there is more than one possible key for a table; in that case one of these alternatives is selected to be the **primary key** and all the other possibilities are called **candidate keys**. Given the choice, a single column key is better than a multiple column key. In general a column containing a person's name is the worst possible choice. Often there is an obvious alternative. For example student registration number or national insurance number. Some apparently obvious keys are unsuitable, for example driving licence number only works in applications where one can rely on everyone having a driving licence. It is tempting to invent new keys specifically for a particular application but it is not always a good idea. In small systems, where the number rows is never likely to approach 2^{32}, a simple integer is sufficient but even then **generated keys** create their own housekeeping problems and need to be carefully controlled. In most application environments there is a naturally occurring key that will do perfectly well and if that occurs artificial keys should be avoided.

The value in a column which is, or forms part of, the primary key of a table cannot be **null**. A null value is undefined; a column/row combination that has not had any value put into it. This restriction is known as the **Entity Integrity Constraint**.

In order to make connections between tables the key of one table can be used in another table where it is not the key , or not the complete key. In that case the pair of columns must have the same domain in both tables. In one table the column is the key and in the other table, where it is not the complete key, it is known as a **foreign key**. Foreign key columns, unlike primary keys, can contain the same value several times and can also contain null values but if they contain a value which is not null that value must also occurs as a primary key value in the other table. This is known as the **Referential Integrity Constraint**. The point of the referential integrity constraint is that it allows sensible links to be made between tables.

For example, suppose the University administration has a table of data about students and another table of data about degree courses then they could make a connection between the two to represent which student was on which degree course by having a column in the students table whose domain was the key of the degree table. Now to find the names of all the students on a particular degree the University just has to list all the rows with the correct value in the foreign key column representing each student's degree. Alternatively they can find details of the degree a student is registered for by looking up the student's details and then using the degree code to look up the student's degree course details in the degree table. This lookup will only work if there are sensible foreign key values. We can say that we do not know what degree a student is registered for using a null value but we cannot allow a student to be registered for a degree that does not exist hence the need for the referential integrity constraint.