The Fundamentals of Database Design Using Entity Relationship Analysis
By Rhys Shobbrook

Introduction
One important aspect database development is that if you don’t get the design right at the start, you can end up with huge problems when using the database. Repairing a database can be very complex and time consuming. Often it is easier to start again.

A common technique of database design is Entity Relationship Analysis (ERA).

Revision
Remember the hierarchy of data. A database is made up of files. In a relational database (such as MS Access) the files are called tables. The tables are made up of records. Each record has fields to store the data in.

Figure 1 - Hierarchy of data
**Entities**

What is an entity? An entity is a thing, object, person, event, etc. that we are interested in storing data on. In a university's database, typical entities would be student, staff, room, etc. In a library database, typical entities would include author, title, publisher, etc.

Later on, we will cover the difference between a strong entity and a weak entity.

We represent an entity by using a rectangle with the name of the entity in the middle of the rectangle.

![Figure 2 – Representing Entities](image)

In a relational database (e.g., MS Access) we create a separate table for each entity we are storing data on.

**Attributes**

What do we want to record about the entities? For a student, you may want to record the student's name, family name, date of birth, address, phone number, etc. For a book, you may want to record the title, author, publisher, date of publication, edition, etc.

Anything that can be recorded about an entity is an attribute or characteristic of that entity. The attributes that we select to record depend on the purpose of the database. A student database would not record details about a student's teeth, but a database for a dentist would definitely include details on teeth. A part of the art of database design is identifying the relevant attributes.

A good rule of thumb is if we want a particular piece of information outputted in a query or report then we need to record that attribute. Always try to keep in mind how the database is to be used.

While the entities determine the tables we create, the attributes determine the fields in each table. In other words, every attribute you decide to record, becomes a field in the table that is created for that entity. In the example of the university database above, student is an entity type, and would have a table created for it. Fields are created to store the attributes in (i.e., name, family_name, date_of_birth, etc.).

One field must fulfill the role of the primary key. The primary key (PK) uniquely identifies each record in the table. An example of a primary key is your student number. No-one else has your student number in Swinburne. While there may be others who have the same first name, or may have the same family name, the student number is unique. Every table needs to have one field that is the primary key.

We can represent the attributes of an entity by drawing bulleted lines from the entity and recording the attribute name next to the bullet as shown in Figure 3 - Representing Attributes. The primary key is underlined.
You will notice with the example of the library database, attributes such as author and publisher have also been listed as entities, whereas edition and date of publication are only listed as attributes. How can we tell whether we an attribute is also an entity? This depends upon rules of normalisation, which we will not be covering in this course, however if you need to record more than one specific piece of information about an object then we treat it as an entity.

For example in the library database when we record information about the author we would most likely want to at least record the author's name and family name, but what if two authors have the same name? How can we distinguish between them? If we also record the author's date of birth or place of birth then we can use this to help distinguish between them.

Whenever we need to record more than 1 attribute about an object, we then treat that object as a separate entity.

Let's analyse a specific case. In Swinburne's computer labs we have desks. The desks have chairs in front of them and computers on top of them. In Swinburne’s case, they do not keep details on specific desks and chairs; they only keep details on specific computers.

While the chairs and the desks are clearly objects, and hence entities, because we are not recording specific information on each we do not create tables for them in the database. In Swinburne’s case the important information is the number of desks and chairs that are in the room. In this case it is more effective to treat the number of desks as an attribute of the Room, similarly with the number of chairs.

However with the computers, if one does not work, we need to report it to ITS. ITS need to identify the computers in the room that need repair, and will want to review the repair history for that computer. So in this case we need to store individual information on each computer and hence we need to create a table for the computer entity.
**Relationships**

We would then create a link between the table for the rooms and the table for the computers. When we create a link between tables it is called a **relationship**.

Most relationships are between two (and only two) entity types. For example, the Computer is in the Room. This means there is a relationship between the Room entity type and the Computer entity type. When there are only two entity types involved in a relationship then it is called a **binary relationship**.

We can represent relationships by using a rectangle to represent an entity and a diamond to represent a relationship. A binary relationship would look like this:

![Figure 4 - A binary relationship](image)

Most of the relationships you will come across will be binary relationships. However you will come some other types.

**Unary Relationships**

Unary relationships occur when there is a relationship between two instances of the same entity type. A typical situation where you will find a unary relationship is a company where both the worker and the supervisor are both of the employee entity type.

![Figure 5 - A unary relationship](image)

**Ternary Relationships**

A ternary relationship arises when a relationship can only exist when three entities are involved. If any of the entities are not present then the relationship cannot exist.

For example, in a biological context, a mother cannot produce a child without the involvement of a father. The relationship between the three can only come into existence with the involvement of the three.
You will not be required to model unary or ternary relationships in this course.

**Cardinality**

Cardinality refers to the type of relationship that exists between the entities.

Can the relationship exist between many instances of the entities, or can it only be one? For example, a mother can have many children, but a child can only have one mother. In Australia, a husband can have one wife; a wife can have one husband.

These relationships can be represented as a one-to-one relationship, a one-to-many relationship or a many-to-many relationship.

A one-to-one relationship is represented by a vertical bar (e.g. ‖ ) at each end of the relationship line.

![Figure 7 - One-to-one relationship](image)

A one-to-many relationship is represented by a vertical bar on the one side of the relationship (e.g. ‖ ) and a three lines coming from the relationship line (e.g. (<)

![Figure 8 - One-to-Many relationship](image)
A many-to-many relationship is represented by three lines coming from both ends of the relationship line (e.g. ~<~)

![Figure 9 - Many-to-Many relationship](image)

MS Access uses different symbols in its entity relationship diagrams. A ‘1’ above the relationship line labels the ‘one’ side of the relationship and a ⊗ for the ‘many’ side.

![Figure 10 - MS Access ERD symbols](image)

Identifying the cardinality of a relationship is very important as it determines how we create relationships (or links) between our tables.

To create a relationship in a relational database we must create a foreign key (FK). The foreign key must be of a compatible data type with the primary key of the table we wish to create a relationship with. For example, if we have a primary key that is comprised of 8 text characters, then the foreign key must have 8 text characters. In MS Access if you use autonumber for your primary key then your foreign key must be the same number type as the autonumber. This is generally the ‘long integer’ number type.

To complete the relationship the value of the primary key of the record you are linking to, is entered into the foreign key.

Let’s look at the example below (Figure 11 - Using foreign keys); if I wanted to find which student topped the exam results, I would find the largest score in the Exam_Results table [1]. In the field next to the largest score I would retrieve the student number (this is the foreign key) [2]. I would use this value to find the record with the matching primary key in the student table [3] and from that record retrieve the name of the student [4]. Thus our query about who got the best exam result retrieves Jack Strong’s name.
But the important question is ‘what side of the relationship do I place the foreign key?’ The answer comes from our cardinality analysis. The foreign key is always placed on the many side of a one-to-many relationship. See [Figure 8 - One-to-Many relationship](#) and compare it with [Figure 10 - MS Access ERD symbols](#). Notice that the many side of the relationship is on the child’s side. Consequently the ‘MotherForeignKey’ is placed in the ‘Child’ table and links to the primary key in the ‘Mother’ table.

Many-to-Many relationships cannot be modelled in this way. This will be covered in more detail in advanced database courses. But in short they must be decomposed into numerous binary relationships where weak entities are created to facilitate the many-to-many relationship. For example if a doctor can see many patients and a patient can see many doctors, which table do we put the foreign key into? As it is a many-to-many relationship we decompose it and create a weak entity called an ‘appointment’. We also create a table for that weak entity. A doctor can have many appointments, but an appointment can only have one doctor. A patient can have many appointments, but an appointment can only have one patient. Thus the foreign keys are created in the ‘Appointment’ table.

### Mandatory Relationships

We need to next ask ourselves ‘Is the relationship optional or mandatory?’ In a mandatory relationship one of the entities cannot exist unless the relationship exists. For example a child must have a female as a mother. The child cannot exist unless there was a mother.

But a female does not have to be a mother. The mother relationship for a female is optional, but it is mandatory for a child.

In Entity Relationship Diagrams we indicate a mandatory relationship by adding a line across the relationship line (e.g. just as we did on the ‘one’ side of a relationship). This means that if the entity is on the one side of a relationship, it will have two lines crossing the relationship line.

We represent an optional relationship by placing an ‘O’ over the relationship line as below in [Figure 12 - Mandatory and optional relationships](#).
We read this symbol in the following way. A mother **MAY** have **MANY** children.

The reverse is read in a similar fashion. We start with the first entity but use the cardinality on the opposite side to express the relationship.

```
Figure 12 - Mandatory and optional relationships
```

Thus Figure 13 - Expressing relationships is read ‘A child **MUST** have **A** Mother.’

Exercise: Draw a relationship diagram that represents the following.

1. A Brother **MAY** have **MANY** Sisters
2. A Driver **MUST** have **A** Licence

Mandatory relationships are implemented in databases by using **Referential Integrity**. Referential Integrity forces the user of the database to maintain the relationship. For example in the Mother/Child example, because the child’s relationship with the mother is mandatory, referential integrity means that when I enter a child’s details, I must also enter in the mother’s primary key into my foreign key (MotherForeignKey). Referential Integrity also means that every child’s mother entered must already have a record in the Mother table.

```
Figure 14 - Referential Integrity in MS Access
```
Summary
From Entity Relationship Analysis we get the following information.

1. **The Entities** – we create a **table** for every entity we want to store more than one piece of information about. Every instance of that entity type is entered into the table as a **record**. (e.g. A table is create for the entity type student. Each student has a record created for their data)

2. **The Attributes** – the attributes or characteristics of the entity are used to create the **fields** in the table. One attribute needs to uniquely identify each record and is called a **primary key**.

3. **The Degree** of a relationship determines how we link the tables together. For a **binary** relationship we create a **foreign key** which has a compatible data type with the primary key of the other table we want to link to.

4. **The Cardinality** of the relationship determines which table we put the foreign key into. In a **One-to-Many** relationship the foreign key is placed in the table on the many side of the relationship.

5. **Mandatory relationships** are implemented by enforcing **referential integrity**.