Entity Relationship Diagrams

A company employs a number of programmers. Programmers are assigned to work on projects. A programmer can work on only one project but writes many programs for each project. A program may be written by many programmers. Each project is controlled by one manager.

Question 1

Decide what *entities* are required for this system.

Question 2

1. Draw an ER diagram that shows the *relationships* between entities.
2. This involves considering the *degree* of the relationships.
3. You might also like to consider whether a relationsip is *mandatory* or *optional*.
4. What questions can your system answer?

Question 3

Create a *table* for each entity. This requires you to decide some basic *attributes* for each entity.

Question 4

Now consider the relationships and determine the placement of *foreign keys*.

Remember that foreign keys form the links between the tables. A foreign key value must generally be a primary key value in the linked table (i.e. a foreign key value exists in the database as a primary key value: so the *referential integrity* of the data base is maintained). However, in some circumstances, the foreign key might be *null*: it's value is not yet defined. This can happen when relationships are *optional*.

Question 5

Your ER diagram should show a *many-to-many* relationship between *Programmer* and *Program*.

This can be resolved to two *one-to-many* relationships by inserting a *link* entity. (something like *Contract*)

It is important to do this here. Why?

Question 6

By drawing FD diagrams, say if your tables are in 3NF or BCNF or both.

Model answer to ERD workshop

Question 1
Initially, the following entities can be declared:

- Manager
- Program
- Programmer
- Project

There is no need to make an entity 'Company' since all 4 entities above are contained within the company.

Note: It is important that we do not use the *plural* for naming entities. i.e. we should have Program **not** Programs.

Question 2

1. A suggested ER diagram is shown in Figure 0.1.
2. The figure shows the following relationships:
   (a) Program-Programmer relationship: *many to many*
   (b) Programmer-Project relationship: *many to one*
   (c) Manager-Project relationship: *one to one*

   An additional *link* entity can be inserted to resolve the *many to many* relationship. It is shown in Figure 0.2.
3. Some suggested *optional* and *mandatory* relationships are shown in Figure 0.3

   But we will continue to develop the tables from Figure 0.2.
4. The relationships are *navigational routes* through the entities. So we can ask questions that can be answered by following a route.
   - What programs is Mr Y writing?
   - What programs are being written for Project X?
   - Who is Mr Z's manager?

   ... for example

The tables in Figure 0.5 are derived from the **entities** in Figure 0.2 **Note:** the attributes listed in each table do **not** help to model the **relationships** shown in Figure 0.2.
(There are only a few attributes shown. You can of course add much more information to the tables, but remember that we must *normalise* them to 3NF or BCNF eventually.)

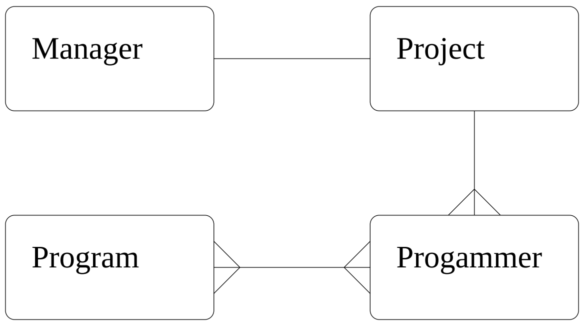

Figure 0.1: ERD for the four entities in answer to Question 1.

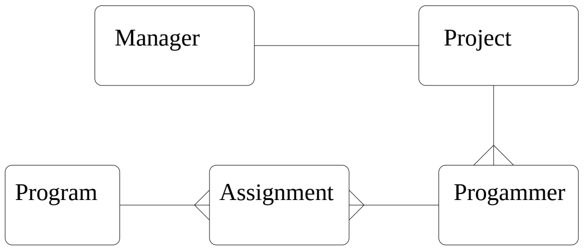

Figure 0.2: ERD with link entity ‘Assignment’ to resolve ‘many-to-many’.

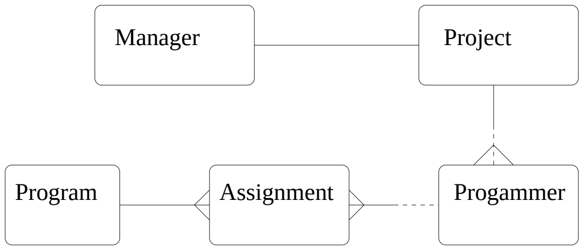

Figure 0.3: ERD with some possible optional relationships

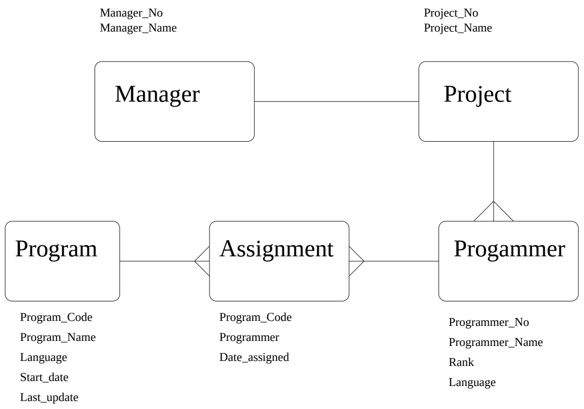

Figure 0.4: ERD with some possible attributes

Manager

Manager_No	*Manager_Name*

Project

Project_Code	*Project_Name*

Program

Program_No	*Program_Name*	*Language*

Programmer

Programmer_No	*Programmer_Name*	*Language*	*Rank*

Assignment

Program_No	*Programmer_No*

Figure 0.5: Tables to model entities Manager, Program, Project, Programmer and Assignment. They are not yet complete, since some are missing their foreign keys.

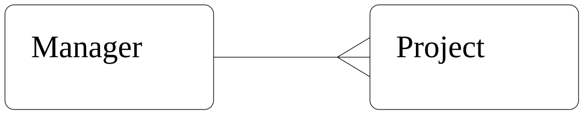

Figure 0.6: A manager can manage many projects

Manager

Manager_No	*Manager_Name*

Project

Project_Code	*Project_Name*	*Manager_No*

Manager_No is foreign key that references Manager

Program

Program_No	*Program_Name*	*Language*

Programmer

Programmer_No	*Programmer_Name*	*Language*	*Rank*	*Project_Code*

Project_Code is a foreign key that references Project

Assignment

Program_No	*Programmer_No*

Program_No and Programmer_No are foreign keys that reference Program and Programmer respectively

Figure 0.7: Primary keys are underlined. Relationships between entities are shown as foreign keys

Question 3
Consider the **relationships** shown in the ER diagram Figure 0.2. Some tables should contain attributes that are in fact *foreign keys.* These attributes will form the necessary links between tables, so that *join* operations can be done.
Note: See Figure 0.7. Primary keys are underlined.

Note: Foreign keys are the means by which tables are *linked* (the relationships are now expressed in the tables).
Note: I have added the attribute *Manager_No* to the **Project** table. *Manager_No* is now a foreign key in the Project table and it *references* table Manager. I can easily ask the question "Who is the manager of Project_No x?" And I can use the 'foreign-key to primary key' join to ask "What project is Manager X in charge of?"
This particular foreign key means that any manager that is shown in any Project table must exist in real life - and be listed in the Manager table !!
Why do this?: Imagine that it becomes possible for a manager to manage many projects. (See Figure 0.6.) Then the tables in Figure 0.7 for Project and Manager can be left unchanged for this situation.
But, if I had chosen to put the attribute *Project_No* into the Manager table instead of putting the attribute *Manager_No* into the Project table, then the following situation might arise, bringing with it the usual anomalies (insert, delete, update):

Manager

Manager_No	*Manager_Name*	*Project_No*
21	Jones	1
21	Jones	2
...	....	.....

The table exhibits *redundancy.* I have to repeat information for every project that a manager works on. Furthermore, I now need **two** attributes in my primary key - {*Manager_No, Project_No*}. It is better to avoid this.
But if I use the solution in Figure 0.7, then their is no such redundancy because each project still has only one manager.

Question 4
Figure 0.2 shows the link entity, Assignment. It is important to make a link entity in *many-to-many* relationships. This helps to remove redundancy and to refine the Primary Key (hopefully) to a singleton set - that contains only one attribute.
The following tables show some pitfalls of many-to-many's:

Program (*Program_No*, *Program_Name*, *Language*, *Programmer_No*)

Programmer (*Programmer_No*, *Programmer_Name*, *Language*, *Rank*, *Project_Code*, *Program_No*)

Can you see some problems here - Primary keys? Redundancy? Foreign keys?

Question 5

Recall that a table is in BCNF if and only if:
every non-trivial and left-irreducible functional dependency has a candidate key as its determinant.
Alternatively, we can simply say 'all determinants are candidate keys.'

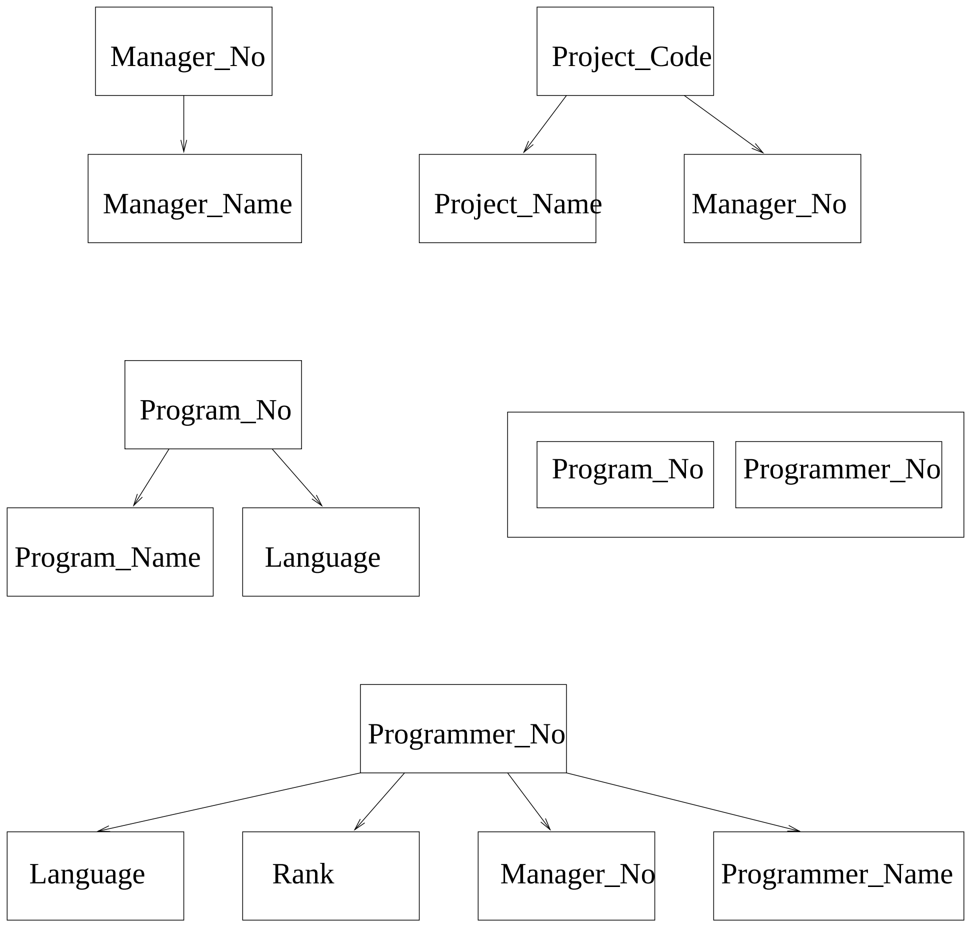

Figure 0.8: FD Diagrams for tables

In terms of FD Diagrams, this means that 'all arrows come out of candidate keys'.
This is true for all the tables shown in Figure 0.7 and their associated FD diagrams shown in Figure 0.8.
If a table is in BCNF, it must be in 3NF, since $3NF \subseteq BCNF$.
Therefore all tables are in both 3NF and BCNF.
Note: If you believe that an attribute like *Project_Name* is a candidate key for table Project, then the Project table is still in BCNF, since:

Project_Code $\leftrightarrow$ Project_Name
Project_Code $\rightarrow$ Manager_No
Project_Name $\rightarrow$ Manager_No

.... all determinants would be still be candidate keys and if we draw the extra arrow on the FD diagram, then all arrows come out of candidate keys.
Final Remark: It is now possible to derive all sorts of tables that summarise particular information in answer to particular questions. But we have been modelling the entities: Program, Programmer, Project and Manager. We would like to have information about each such entity in a base relation. We discovered that a link entity got rid of some undesireable properties in a many-to-many relationship.