

Brenda's Basketball Courts

A Systems Analysis Example

Written by: Mark O'Reilly

Brenda's Basketball Courts

A Systems Analysis Example

Thanks to:

- David Abercrombie
- Stuart Garner, Knowledge Base
- Queensland University of Technology
- The Brisbane Institute of TAFE
- The Moreton Institute of TAFE
- Jordan Russell for Inno Setup
- All those who have offered comments and corrections

This does not imply endorsement of this book by any of these organisations or individuals.

Created by Mark O'Reilly (moreilly@powerup.com.au), 1999-2016
www.majorsoftware.com.au

*This text is being distributed as Freeware.
The copyright remains with Mark O'Reilly*

This text has been made available in 'soft copy' and can be altered and amended as required. You are requested to cite any variations to the original material clearly and fully. Mark O'Reilly cannot be held responsible for any adjustments and amendments made to the original text.

If you find any omissions, oversights or errors within the text, or wish to offer suggestions, please email Mark O'Reilly at moreilly@powerup.com.au

Organisation, brand and product names used in this book may be trademarks or registered trademarks of their respective organisations.

Contents

Introduction	2
The script.....	2
Data Flow Diagrams.....	3
<i>Process for creating a set of Data Flow Diagrams.....</i>	<i>3</i>
<i>Context Level Diagram.....</i>	<i>3</i>
<i>Level 0 Diagram</i>	<i>4</i>
<i>Level 1 Diagrams</i>	<i>7</i>
<i>Physical and Logical Diagrams.....</i>	<i>7</i>
<i>Common Diagramming Errors and Guidelines</i>	<i>8</i>
<i>Creating a Context Level Diagram.....</i>	<i>9</i>
<i>Creating a Level 0 Diagram</i>	<i>10</i>
Data Dictionary	14
<i>External Entities.....</i>	<i>14</i>
<i>Data Flows.....</i>	<i>14</i>
<i>Data Stores.....</i>	<i>17</i>
<i>Processes.....</i>	<i>18</i>
E-R and Normalisation Notes.....	21
<i>Some Background Regarding Entity-Relationship Diagrams.....</i>	<i>21</i>
<i>Some background on Normalisation.....</i>	<i>26</i>
E-R and Normalisation for Brenda.....	29
<i>Entities.....</i>	<i>29</i>
<i>Relationships</i>	<i>29</i>
<i>ER Diagram</i>	<i>29</i>
<i>Attributes</i>	<i>31</i>
<i>Normalisation.....</i>	<i>33</i>
Some Thoughts on Tables	35
1. <i>The Entities.....</i>	<i>35</i>
2. <i>Many to Many and One to Many Relationships</i>	<i>36</i>
3. <i>The Attributes</i>	<i>37</i>
4. <i>Lookup Lists</i>	<i>38</i>
5. <i>A One to One Relationship.....</i>	<i>38</i>
1. <i>References and Resources</i>	<i>39</i>

Introduction

These notes have been produced primarily to illustrate *an* approach to the development of Data Flow Diagrams, portions of a System Dictionary, Entity Relationship Diagrams and Normalisation. The case study itself has been scoped down so as not to detract attention from this primary focus.

An extended script and database design has been presented in the Functional Database Design text.

The example is presented in a step-wise tutorial format so the development process behind each component is better illustrated.

The script

Brenda manages a set of basketball courts. At the start of the season she receives a draw from the LBCG (Local Basketball Co-ordinating Group) which she places in her top drawer for review purposes.

Brenda must ring the various team managers (who are responsible for their individual teams) to verify games the week before they are played. She tells each manager the date and court of their next game. The manager indicates the team's intent to play.

The manager faxes a list of players to Brenda so she can organise the scoring sheets in advance. She stores the faxes and the scoring sheets in her filing cabinet.

The manager verifies the player list when the team arrives to play. The fouls and goals are recorded during the game. After the games, Brenda generates a set of summaries of the individual scorecards. Brenda stores the score cards and the summaries in her filing cabinet.

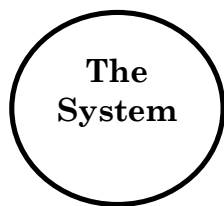
Brenda sends copies of the score card summaries to the LBCG and to the respective team managers.

Data Flow Diagrams

Process for creating a set of Data Flow Diagrams


The process for creating a set of data flow diagrams is provided below:

Context Level Diagram



STEP 1 - Add the System

Identify the main business, the main component, **the system**, the 'thing', that we are focusing on. You need to draw a circle in the middle of your diagram and add the name of the System to the circle.



A square box with the text "External Entity" centered inside it.

STEP 2 - Add the External Entities

Identify all the things (people, businesses, entities) that provide data to or receive data from the system. These are called External Entities. You need to draw a square cornered box for each of these and label each one appropriately. Spread these out around the System you have already drawn.

Please note that External Entities might identify natural groupings. For instance a business might deal with trade as well as general suppliers. These two groups may need to be identified as separate entities, or may need to be grouped as one entity - Suppliers. Where there is uncertainty about such grouping, check back with the client.

Data Flow

STEP 3 - Add the External Data Flow Lines

Identify and add to the diagram any items (physical items, data, or information) that pass between the External Entities and the System. These items are drawn as lines, with arrows, and have the name of the item written along the line. It can be useful to work on one External Entity at a time, and use your knowledge of the system to decide on what lines are required. When working to a script, such as is given in this example, take care not to create lines that are not mentioned, unless they are obvious or inferred. Experience and common sense will help you with this step.

Level 0 Diagram

The Context Level Diagram could be expanded on to a new diagram - a Level 0 diagram. This new diagram would describe the processes and data stores within the system. NOTE: For more complex systems, functional decomposition may be required to simplify the system. This is not detailed in this example.

Nine steps are suggested for the drawing of a Level 0 diagram:

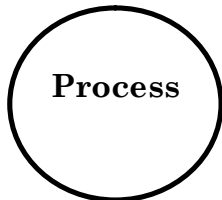
1. Draw the boundary and external components
2. Determine, draw and label internal processes
3. Join internal processes to external components with existing data flow lines
4. Separately list all data stores
5. Check which data stores need to be on the current diagram
6. Draw in appropriate data stores
7. Add in internal data flows
8. Review the data flow through the system
9. Review the Data Flow Diagram against the client requirements or script

STEP 1 - Draw the boundary and external components



This step requires that you sketch a rectangle on the page and place the external entities from the Context Diagram on the outside of the rectangle.

STEP 2 - Determine, draw and label internal processes



This step requires you to focus on the tasks that are completed by people, or are otherwise automated, within the system. In scripts, these processes are occasionally presented to you in separate paragraphs. Otherwise, you need to sit and consider what tasks represent one unit of activity - usually occurring at one time by one person, a group of people, or by an automated process. For instance, if an order is being processed, it may first be recorded, then filled, and then invoiced - Record Order Details, Fill Client Order, Invoice Client Order. Each of these is a Process. Often a Process can be best labeled using a verb followed by a descriptor or qualifier and then a noun. EG: Calculate Sales Tax, or Process Customer Order.

STEP 3 - Join internal processes to external components with existing data flow lines

Data Flow

This step requires you to link the external data flow lines from the Context Diagram to the Processes that you have just created. It is important that every line from the Context Diagram re-appears on this diagram, and that no external data flow line appears on this diagram that was not placed on the Context diagram. If a line has been missed, then you need to go back and enter it on the Context Diagram first.

STEP 4 - Separately list all data stores

This step requires you to list all the Data Stores that are likely to be used within the system. If the system is manual then Data Stores include filing cabinets, in-trays, out-trays, etc. If the system is computerised, then the Data Stores are often suggested by the pieces of paper or the pieces of information on the Data Flow lines - if a piece of information or a piece of paper stops somewhere, then there must be a Data Store for it to go into.

Data Store

STEP 5 - Check which data stores need to be on the current diagram

This step requires you to decide whether the data stores that you have selected should appear on the current diagram. The rules are: If a data store is connected to two or more processes, it is to be placed on the current diagram. If a data store is only connected to one process on the current diagram and this process is likely to be further expanded, then the data store DOES NOT go on the current diagram. If the process IS NOT likely to be further expanded, then the data store IS to be placed on the current diagram.

STEP 6 - Draw in appropriate data stores

Simply draw in the data stores that are required on the current diagram.

Data Flow

STEP 7 - Add in internal data flows

This step requires you to decide on and place data flow lines between two processes, and between data stores and processes within the system boundary. All data stores should have at least one line going in and one line coming out, otherwise they serve no useful purpose.

In some instances, a process can read data from a data store, thus requiring a line with an arrow pointing from the data store to the process. Other times a process is writing data to a data store, thus requiring a line from the process to the data store. Sometimes the process both reads and writes the same data fields. In this last case, a double ended arrow might be used.

You are not allowed to have data flow lines from one data store to another. This would be like one piece of paper jumping out of one filing cabinet into another, and rearranging its data without any computer or human processing. Similarly, you are not allowed to have a data flow line between an external entity and a data store - a process is required to 'manage' the data into or out of the system.

STEP 8 - Review the data flow through the system

Once all the additional elements of the Level 0 diagram have been identified (processes, data stores and internal data flow lines), a thorough check is required of the flow of data through the system.

To do this, select a source of data or flow and follow its pathway through the system, checking that:

1. The flow does not have gaps. That is, check that a sequence of data flow lines does not have a link missing.
2. The flow has appropriate start and end points. For instance, data does not come to rest in a data store with no evidence of further use.

STEP 9 - Review the Data Flow Diagram against the client requirements or script

As a final check, it is necessary for you to return to the script or list of client requirements and check each section or item in turn on the Data Flow Diagram. Ensure that all relevant data flow requirements have been represented on your Data Flow Diagram.

Level 1 Diagrams

Each process on the Level 0 Diagram could be expanded on further to describe its internal function. These new diagrams are all considered to be Level 1 diagrams, and are usually numbered according to the process they describe.

Seven steps are suggested for the drawing of a Level 1 diagram:

1. Draw the boundary and external components
2. Determine, draw and label internal processes
3. Join internal processes to external components with existing data flow lines
4. Draw in appropriate data stores
5. Add in internal data flows
6. Review the data flow
7. Review the DFD against the client requirements or script

NOTE: Steps 4, 5 and 6 from Level 0 reduce to Step 4 in this level.

Physical and Logical Diagrams

It is often suggested that in analysing a system the systems analyst might begin by creating a set of Data Flow Diagrams that depict the physical people and items currently operating within the system. These **Physical Data Flow Diagrams** would identify people and groups by name (identify processes by the name of the person or group responsible for the various tasks), and data stores by the names of the physical items in which the data is stored (eg: green filing cabinet, pink in-tray).

Based on this set of physical data flow diagrams, the systems analyst might create a parallel set of **Logical** diagrams in which people and groups are replaced by role titles, and data stores are named according to the data being stored.

The analyst might then analyse this set of logical diagrams and create a third set of diagrams (also logical) in which roles, responsibilities and data storage are rationalised. Based on this third set of diagrams, the people and the various physical items could be re-introduced in a fourth and final set of diagrams - a set of new physical data flow diagrams.

This document will primarily deal with the initial physical data flow diagrams for this system.

Some Guidelines for Drawing DFD's

- Don't create too many levels - usually work up to 3 levels below context
- Don't create too many processes - usually work up to 5 to 9 processes per level.
- Don't overuse or misuse the Data Flow Diagrams. There are other tools more appropriate to some situations.
- Make use of other tools if and where appropriate.
- Check the content of your Data Flow Diagrams (DFD's) with the Client
- Make sure levels balance with the numbers of flows. For instance, all the data flow lines on the Context Level Diagram should be present as external data flow lines on the Level 1 Diagram.
- External entities should be outside boundaries
- Don't duplicate names for different elements
- Annotate any repetition, and number flows and stores correctly
- Use meaningful names
- Don't mix physical and logical DFD's
- A level with one process is trivial - do you need it?
- A data store should be at a higher level if more than one process at that higher level addresses it
- All data stores must be shown somewhere
- Data stores must have flows in and out
- Watch that data flow directions are correct

Creating a Context Level Diagram

NOTE: The symbols that have been suggested above are from the Yourdon Data Flow Diagram system. The symbols used below are from the Gane and Sarson system. A Yourdon diagram is effectively the same as a Gane and Sarson diagram, only the symbols are different. The following Data Flow Diagrams were prepared using the Ascent CASE Tool. (Visit: www.knowledgebase.com.au)

STEP 1 - Add the System

Script:

Brenda manages a set of basketball courts



STEP 2 - Add the External Entities

Script:

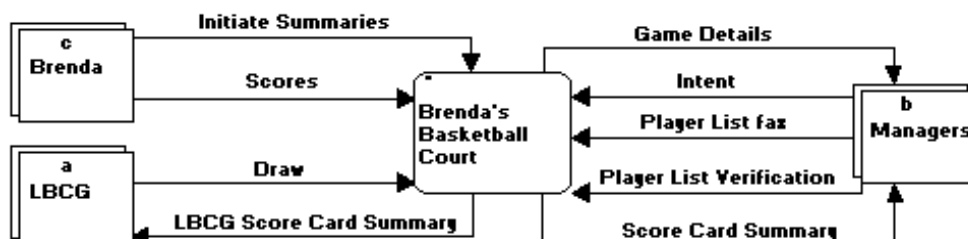
- *At the start of the season she receives a draw from the LBCG (Local Basketball Co-ordinating Group)*
- *Brenda must ring the various team managers*



STEP 3 - Add the External Data Flow Lines

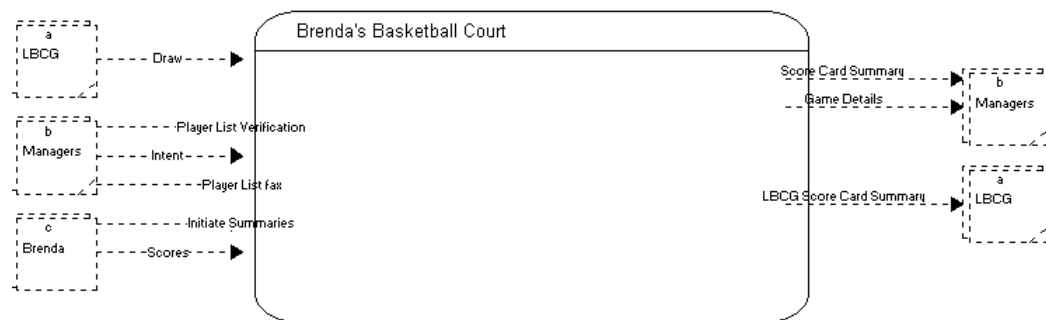
Script:

- *She receives a draw from the LBCG*
- *She tells each manager the date and court of their next game*
- *The manager indicates the team's intent to play*
- *The manager faxes a list of players*
- *The manager verifies the player list when the team arrives to play*
- *The fouls and goals are recorded during the game. After the games, Brenda generates a set of summaries of the individual scorecards*
- *Brenda sends copies of the score card summaries to the LBCG and to the respective team managers*



Creating a Level 0 Diagram

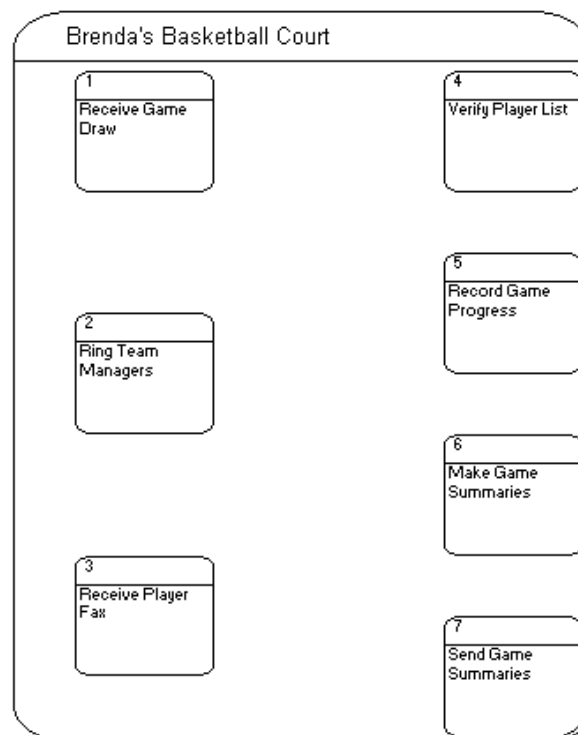
STEP 1 - Draw the boundary and external components



STEP 2 - Determine, draw and label internal processes

Script:

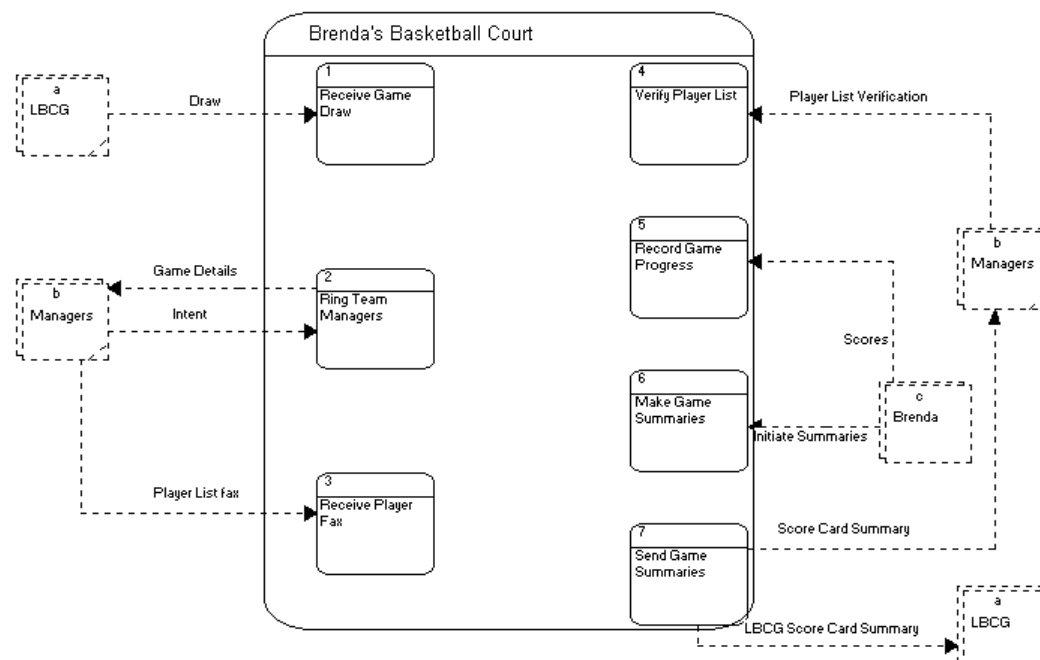
- *She receives a draw from the LBCG*
- *Brenda must ring the various team managers*
- *The manager faxes a list of players to Brenda so she can organise the scoring sheets*
- *The manager verifies the player list when the team arrives to play*
- *The fouls and goals are recorded during the game*
- *After the games, Brenda generates a set of summaries of the individual scorecards*
- *Brenda sends copies of the score card summaries*



STEP 3 - Join internal processes to external components with existing data flow lines

Script:

- *Most of the script is used in this step - a full re-read is appropriate.*



STEP 4 - Separately list all data stores

Script:

- *... which she places in her top drawer for review purposes.*
- *She stores the faxes and the scoring sheets in her filing cabinet*

List:

- Top Drawer
- Filing Cabinet

STEP 5 - Check which data stores need to be on the current diagram

Data Store

Top Drawer

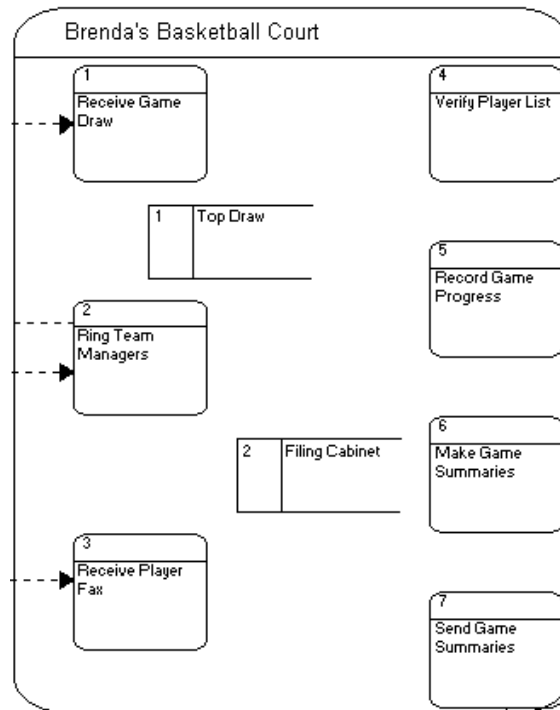
Process linked to this store

1. - Receives the game draw and stores it
2. - Reads the game draw before ringing managers

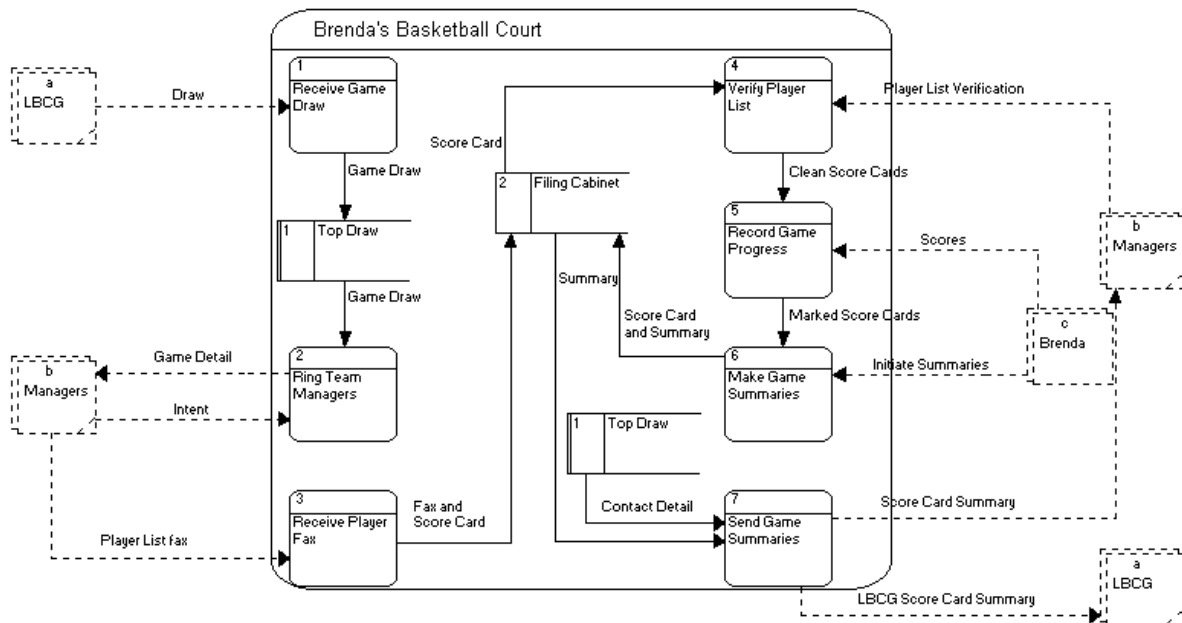
Filing Cabinet

3. - Receives the faxes and stores them with score cards
4. - Retrieves score cards for verification
6. - Re-stores score cards and summaries
7. - Obtains summaries for sending out

STEP 6 - Draw in appropriate data stores



STEP 7 - Add in internal data flows



STEP 8 - Review the data flow through the system

In the case of Brenda's Basketball Courts we would need to check:

1. **The Draw**
The draw comes into the system from the LBCG. It is received and stored by the '*Receive Game Draw*' process, then stored in the top drawer. From the top drawer it is retrieved and used within the '*Ring Team Managers*' process to check each team's intent to play for the given week. We might now ask whether the draw details would be used within any of the other processes. If so, we would look to adjust the Level 0 diagram with additional data flows.
2. **Intent (to Play)**
The 'Intent' to play is generated by the individual team managers and is received by the '*Ring Team Managers*' process. There is no indication that this '*Intent*' to play is used or stored by this process - possibly signifying a gap in the data flow. The use and/or storage of the '*Intent*' to play may need to be ratified by the client.
3. **The Score Cards**
These are generated within the system in response to the Team Managers faxing their respective Team Lists. The score cards can be tracked through the system with a number of triggers impacting on their status and content, and on numerous occasions they are stored and/or retrieved from the filing cabinet.
4. **The Game Summaries**
Brenda initiates the production of the score card summaries, the system generates and stores them and finally sends them to the Team Managers and the LBCG.

STEP 9 - Review the Data Flow Diagram against the client requirements or script

Data Dictionary

External Entities

This part of the Data Dictionary requires you to list and describe the external entities. For each external entity enter the name and a brief description (one or two sentences). This list, as with all those suggested below, should be sorted alphabetically.

Name	Description
Brenda	Brenda is responsible for the entry of scores and fouls etc, and for initiating the production of game summaries.
LBCG	Local Basketball Co-ordinating Group - the body responsible for the management of the local basketball leagues. They prepare draws and manage the weekly game results
Managers	The individual basketball teams are managed by team managers. The managers co-ordinate team members prior and during their weekly games. The managers also co-ordinate with the various playing courts.

Data Flows

The Data Flows require 5 entries:

1. The Name or label on the data flow line.
2. The Alias or common name used for the item associated with the data flow (if any). For instance a form being passed along a given data flow line might be officially known as the R3 form. The Alias might be the 'requisition form'.
3. A Brief Description of the item(s) being passed along the data flow line.
4. The Contents of the data flow. This is a representation of the data elements that make up the data flow. The elements are placed within parentheses which represent repeating units and are separated by commas. Repeating units can be embedded within other repeating units. In practice it is likely that the elements will be drawn from existing documents such as forms, record cards, or reports.
5. The Volume of information travelling along the data flow line. This can be stated in terms of both total volume per day, per month, per year etc, as well as some indication of peak throughput.

EXTERNAL LINES

Name (Alias)	Brief Description	Contents	Volume
Draw	The list of games that are to played by various teams at various sites on different days and times	{ DrawNo, Year, LBCGFax, { TeamName, ManagerName, ManagerPhone, ManagerFax }, { WeekNo, StartingDate, { GameID, GameDate, GameTime, Location, Team1, Team2 } } }	1 per season
Game Detail	The specific game information that is brought to the attention of the individual team managers so that they might verify the team's intent to play.	{ WeekNo, GameDate, GameTime, Location, Team1, Team2 }	20 per week
Initiate Summaries	Request for various summary reports at the end of each game week. Summaries include scores of all games, and player stats per game	{ DrawNo, WeekNo, StartingDate }	1 per week
Intent	An indication from the individual team managers as to whether their team will be playing on the date specified.	{ Response }	20 per week
LBCG Score Card Summary	A summary of all game results for a given week as sent to the LBCG.	{ WeekNo, { GameID, Team1FinalScore, Team2FinalScore } }	1 per week
Player List Verification	An updated list from the individual team managers on arrival at a game indicating the players who will be playing.	{ GameID, TeamName, { PlayerNo, PlayerName } }	20 per week, all arriving within 30 minutes of the game.
Player List Fax	A fax from the individual team managers indicating the players who will be playing for their team in their next game.	{ GameID, TeamName, { PlayerNo, PlayerName } }	20 per week
Score Card Summary	A summary of the team specific game results for the current week as sent to the individual team managers..	{ WeekNo, GameID, GameDate, GameTime, Location, Team1, Team2 Team1FinalScore, Team2FinalScore }	20 per week
Scores	Scores and fouls as per the game currently being played	{ GameID, TeamName, PlayerNo, GoalType, FoulType }	approx 200 - 300 per game

INTERNAL LINES

Name (Alias)	Brief Description	Contents	Volume
Clean Score Cards	A score card with the teams and their players listed but without any associated fouls or goals marked.	{ WeekNo, GameID, GameDate, GameTime, Location, TeamName, { PlayerNo, PlayerName } }	20 per week, each passed just prior to game start
Contact Detail	These are the fax details of the LBCG and the various team managers to whom summaries are to be sent.	{ LBCGFax, { TeamName, ManagerName, ManagerFax } }	1 per week
Fax and Score Card	After a fax is received from the team managers, a draft team score card is generated and stored.	{ WeekNo, GameID, GameDate, GameTime, Location, TeamName, { PlayerNo, PlayerName } }	20 per week
Game Draw	After the season game draw is received from the LBCG, the draw is stored in the top drawer.	{ DrawNo, Year, LBCGFax, { TeamName, ManagerName, ManagerPhone, ManagerFax }, { WeekNo, StartingDate, { GameID, GameDate, GameTime, Location, Team1, Team2 } } }	1 per season
Marked Score Cards	A score card with the teams, their players and associated fouls or goals entered. Team results are indicated.	{ WeekNo, GameID, GameDate, GameTime, Location, Team1, Team2 Team1FinalScore, Team2FinalScore }	20 per week, passed after games are completed
Score Card	Once the teams begin to arrive, the draft score card is amended.	{ WeekNo, GameID, GameDate, GameTime, Location, TeamName, { PlayerNo, PlayerName, Fouls, Goals, Points } }	20 per week, just prior to games.
Summary	The games summaries are drawn from the filing cabinet so summaries relevant to the team managers and the LBCG can be generated.	{ WeekNo, GameID, GameDate, GameTime, Location, Team1, Team2 Team1FinalScore, Team2FinalScore Team1Fouls, Team2Fouls, { PlayerNo, Fouls, Goals, Points } }	20 per week

Data Stores

The Data Stores also require 5 entries:

1. The Name or label on the data store.
2. The Alias or common name used for the data store (if any).
3. A Brief Description of the data store.
4. The Contents of the data store. Like the Contents of a data flow, this is a representation of the data elements that make up the data store.
5. The Volume of information stored in the data store.

Name (Alias)	Brief Description	Contents	Volume
Filing Cabinet	Holds the score cards and weekly game summaries	{ <u>WeekNo</u> , <u>GameID</u> , GameDate, GameTime, Location, Team1, Team2 Team1FinalScore, Team2FinalScore Team1Fouls, Team2Fouls, { <u>PlayerNo</u> , Fouls, Goals, Points } }	1Kbytes x 400 games per season
Top Drawer	The season game draw is stored in the top drawer.	{ <u>DrawNo</u> , Year, LBCGFax, { TeamName, ManagerName, ManagerPhone, ManagerFax }, { <u>WeekNo</u> , StartingDate, { <u>GameID</u> , GameDate, GameTime, Location, Team1, Team2 } } }	4Kbytes, once per season

Processes

For each process identify:

1. The Number of the process
2. The Name or label within the process.
3. A Brief Description of what actions are occurring within the process (for documentation purposes).
4. A Functional Description - a more detailed and exact representation of the actions or activity occurring within the process. This description might be presented as structured English, a decision tree or a decision table.

Number: 1	Name: Receive Game Draw
Brief Description: Receive the game draw from the LBCG and store it in the top drawer.	
Functional Description: For each basketball season Receive game draw from the LBCG Store the draw in the top drawer End For	

Number: 2	Name: Ring Team Managers
Brief Description: Based on the game draw for the next week, ring the individual team managers, informing them of their next week's game and identify their intent to play.	
Functional Description: For each team on next week's draw Ring the team manager Inform the manager of next week's game details Receive the managers intent to play End For	

Number: 3	Name: Receive Player Fax
Brief Description: Receive faxes from each of the team managers with a list of players selected to play in the next game. Based on the fax, a draft score card is created for each game with the team members listed.	
Functional Description: For each team manager fax <ul style="list-style-type: none"> Receive the fax Copy the player names to the appropriate score cards Store the score card and fax in the filing cabinet End For	

Number: 4	Name: Verify Player List
Brief Description: On arrival of the team to the courts for their weekly game, select the appropriate score card from the filing cabinet and make player detail changes as required. Pass the score card to the Record Game Progress process.	
Functional Description: For each team arriving to play <ul style="list-style-type: none"> Receive and/or read player details Retrieve the pre-prepared score card For each player position on the card <ul style="list-style-type: none"> If the detail is not correct <ul style="list-style-type: none"> Update the detail End If End For <ul style="list-style-type: none"> Pass the Clean score card to the Record Game Progress process. End For	

Number: 5	Name: Record Game Progress
Brief Description: As the game proceeds update the record card. Pass the completed card onto the Make Game Summaries process.	
Functional Description: For each game <ul style="list-style-type: none"> For each foul, <ul style="list-style-type: none"> Mark the foul against the appropriate player on the score card End For For each goal, <ul style="list-style-type: none"> Mark the goal against the appropriate player on the score card End For Pass the completed card onto the Make Game Summaries process End For	

Number: 6	Name: Make Game Summaries
Brief Description: Based on the information noted on the various score cards, finalise the score card totals and create summary information for each.	
Functional Description: For each score card <ul style="list-style-type: none"> Finalise the card totals Create a score card summary End For	

Number: 7	Name: Send Game Summaries
Brief Description: Retrieve the game summaries and the necessary contact details. Create a summary document of all games and send to LBCG. Create summary documents for each of the team managers who had a team playing this week and forward them by fax.	
Functional Description: For each week <ul style="list-style-type: none"> Retrieve the contact details For each of the games played this week <ul style="list-style-type: none"> Create a summary document for the respective team managers Send the team summary document by fax End For Create a summary for the LBCG Send the LBCG summary by fax. End For	

E-R and Normalisation Notes

Some Background Regarding Entity-Relationship Diagrams

Entity Relationship (ER) diagrams are a means of graphically representing data structures - the data items and their relationships. In order to create such a diagram, the entities and the data elements must be identified, and some indication of the relationships between the entities is required.

For example, let us say that we wish to create a database to store information about the music that we have at home. We have some CD's, some cassettes, some records and a couple of videos with music clips. We decide we want to store the item title, artist (whether it be an individual, group, or 'various'), cost, type of media (CD/cassette etc) and recording company. And for each item, the names of the tracks, their respective lengths and the songwriter(s) or individual artists.

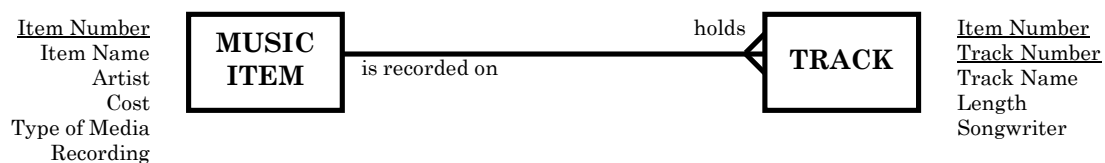
This detail can be broken down as follows:

No	Entities	Attributes (Data Elements)
1	Music Item (eg: CD's, cassettes etc.)	Item Number
		Item Name
		Artist
		Cost
		Type of Media
		Recording Company
2	Track	Item Number
		Track Number
		Track Name
		Length
		Songwriter

The **relationships** between these two entities are:

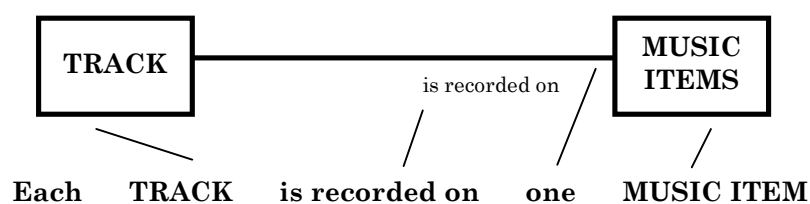
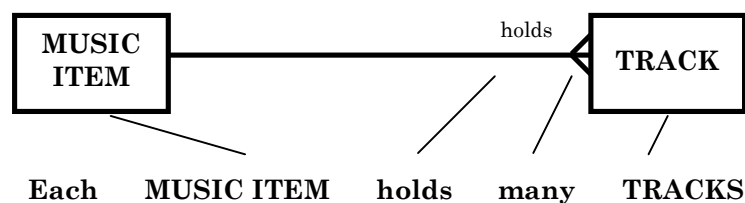
1. A given track is recorded on one music item.
2. One music item holds many tracks

An **Entity-Relationship diagram** for this example might appear as follows:



The following needs to be noted with respect to this ER diagram:

1. The rectangles represent the Entities (the 'things' in the system). By convention, the name of the entity is written in the box.
2. The attributes (data elements) of each entity are listed beside each entity box.
3. There are two types of attributes:
 - (a) **Identifying Attributes** - used to uniquely identify a particular occurrence of an entity. The unique identifier for Music Items is the Item Number. The unique Identifier for Tracks is the Item Number and the Track Number (a given track on a given music item)
 - (b) **Descriptive Attributes** - all other attributes that describe the entity in more detail.
4. The relationship is described by two labels - one above, the other below the line: 'A Music Item **holds** tracks', and: 'A track **is recorded on** a Music Item'.
5. A 'Crows foot' is used on the end of a line to represent a 'many' relationship. The 'crows foot', in this case, appears on the many end - we have many tracks. The full statement of the relationships in this system then are:
 - (a) A Music Item holds many Tracks
 - (b) A Track is recorded on one Music Item.



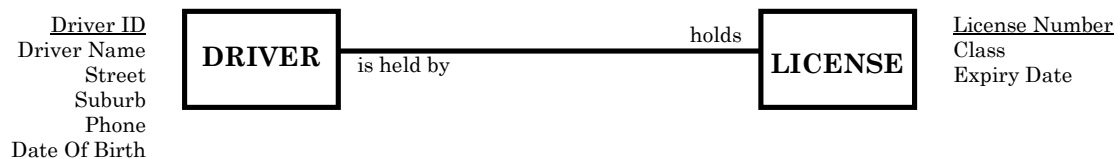
Please note that when the expression *many* has been used above, it might mean none, one or more than one. A music item may have no tracks (unlikely), only one track, or many tracks. Relationships can be described as one-to-one, one-to-many or many-to-many. This aspect of a relation's description is known as **Cardinality**. Cardinality is part of the description of a relationship between two entities. For instance: A Person has one licence, A customer hires many videos. Further discussion relating to relationships and their cardinality is provided on the following pages.

Example of a One-to-One Relationship

The example above represents a One-to-Many relationship. Such relationships are fairly common. Less common however are One-to-One relationships. An examples of a One-to-One relationship might be: a driver holds a license to drive. That is, every driver only has one license. A given license is only held by one driver.

The entities in this example would be Driver and License. The attributes of the Driver might include: Driver ID, Name, Address, Phone and Date Of Birth. The attributes of the License might include: the License Number, Class and Expiry Date.

The Entity Relationship diagram for this example would appear as follows:



Example of a Many-to-Many Relationship

There are a number of instances in the early phases of design where you might identify entities that appear to be related Many-to-Many. Many of these relationships can be simplified if a new entity is introduced to connect them - an **Associate Entity**.

As an example, let us say that we are trying to design a database to track computer system sales in a computer retail outlet. We might initial identify:

- A CUSTOMER entity, with attributes such as CustomerID, Customer Name, Street, Suburb, and Phone.
- An ORDER entity, with attributes such as OrderNumber, CustomerID, SystemNumber and Date.
- A COMPUTER SYSTEM entity, with attributes such as SystemNumber, System Name, Processor, onboard RAM, hard-drive size, etc.

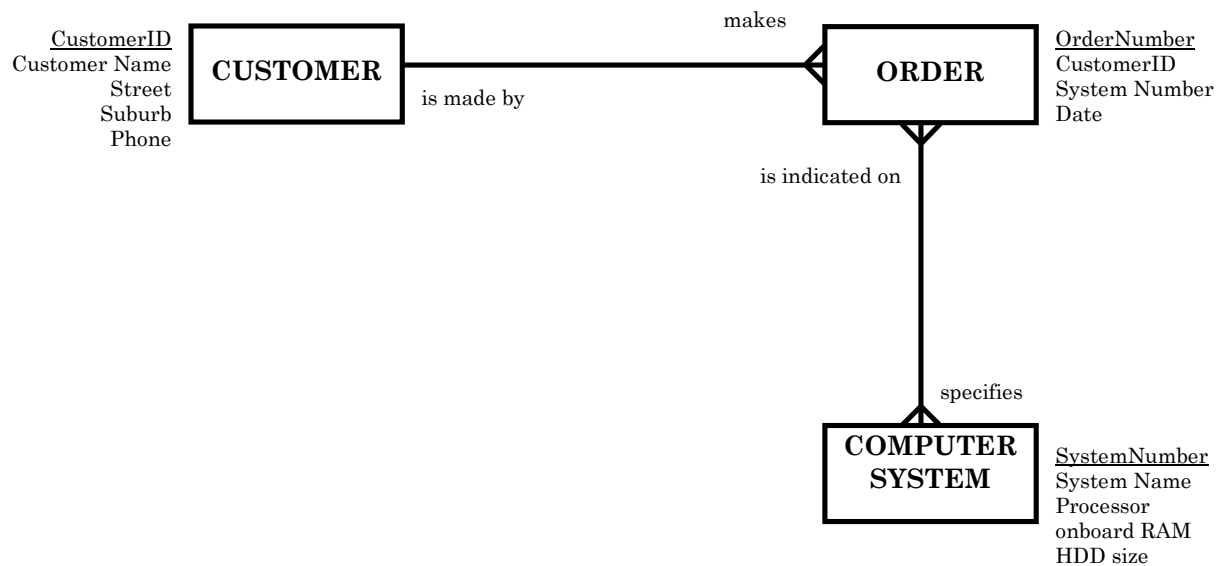
The relationships between these three entities might be:

1. A CUSTOMER makes an ORDER
2. An ORDER is made by a CUSTOMER
3. The ORDER specifies a COMPUTER SYSTEM
4. The COMPUTER SYSTEM is indicated on an ORDER

The cardinality associated with these relationships might be:

1. One CUSTOMER can make many ORDERS
 2. A given ORDER can only relate to one CUSTOMER
- and...
3. One COMPUTER SYSTEM can be placed on many ORDERS
 4. A given ORDER can have many COMPUTER SYSTEMS on it

We would construct the following Entity Relationship diagram based on this information:



You will notice that there is a many-to-many relationship between the Order and Computer System entities.

In general, Many-to-Many relationships are difficult to implement in a Relational DBMS in the form shown above. It is easier to implement a One-to-Many relationship. Many such relationships can be converted from one Many-to-Many relationship to two One-to-Many relationships with the addition of a new entity - an Associative Entity.

Note: *Care must be taken to ensure that the new relationships are in fact one-to-many.*

In this instance we would place an associative entity between the Order and the Computer System entities. We could narrow the focus of the Order entity to only store information about the customer and when they made their order. The purchases - often called Order Lines or Order Details - would become the associative entity.

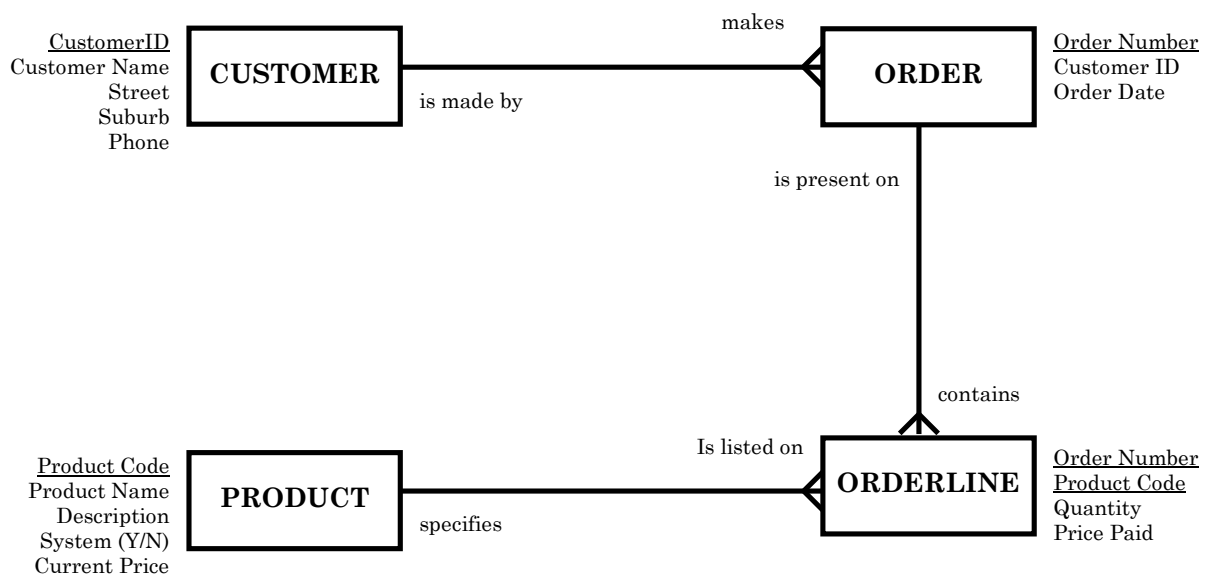
The relationships would then become:

1. A CUSTOMER makes an ORDER
2. An ORDER is made by a CUSTOMER
3. The ORDER details one or more LINES on the ORDER
4. A PRODUCT is listed on an ORDERLINE

The reviewed cardinality associated with these relationships might be:

1. One CUSTOMER can make many ORDERS
2. A given ORDER can only relate to one CUSTOMER
3. One ORDER can have many ORDERLINES
4. A given ORDERLINE can only appear on one ORDER
- and...
3. One PRODUCT can be placed on many ORDERLINES
4. A given ORDERLINE can only have one PRODUCT on it

Thus the Entity Relationship diagram would become:



Some Background on Normalisation

Normalisation is a process that complements and reinforces the process of Entity-Relationship diagramming. It is possible to fully identify a set of entities and their relationships and consequently a set of database tables using Entity-Relationship alone. Nevertheless, Normalisation can allow you to supplement or validate these by starting with data from existing forms or reports and breaking them down into individual relations.

Normalisation can also naturally follow from the data elements identified in the Contents column of the data flow and/or data stores information in a data dictionary.

As an example, let us say that we have a copy of a simple order form from a business that sells PC's. In the Data Flows section of their data dictionary let us assume we identified the content of the order form to be:

{ OrderNumber, OrderDate, ShippingDate, CustomerNumber, CustomerName, Shipping Address, Phone, { ProductCode, ProductName, RetailPrice, Quantity, Subtotal }, Ordertotal }

The process of Normalisation would take the element content of this form and break it down systematically into relations. This process is known to have a number of steps although only the first 3 receive significant attention in general Systems Analysis texts. The output of each step is known as a ***normal*** form. For example:

Original Un-normalised Data Elements

Note: Identifying Attributes have been identified with an underline within the relations below..

{ OrderNumber, OrderDate, ShippingDate, CustomerNumber, CustomerName, Shipping Address, Phone, { ProductCode, ProductName, RetailPrice, Quantity, Subtotal }, OrderTotal }

The Subtotal and Total elements could be removed in the initial stages, realising that these would likely be calculated fields in the resulting database application and would be displayed when data was presented in forms and reports.

{ OrderNumber, OrderDate, ShippingDate, CustomerNumber, CustomerName, Shipping Address, Phone, { ProductCode, ProductName, RetailPrice, Quantity } }

First Normal Form

In this step, the Inner Repeating Groups are removed. The repeating groups are identified by parentheses - { ... }. This step aims to remove or separate the inner repeating group(s) from the outer repeating group. In the process we need to add the Identifying Attribute from the outer group to the inner group so as to retain the link or relationship between the two resulting relations. The Identifying Attribute for the new relation may become a combination of the identifier from the outer group and the identifier for the inner group.

{ OrderNumber, OrderDate, ShippingDate, CustomerNumber, CustomerName, ShippingAddress, Phone }

{ OrderNumber, ProductCode, ProductName, RetailPrice, Quantity },

Second Normal Form

In this step, the Partial Dependencies are removed. It would be a fair statement to say that in this example an element such as ProductName would be better represented by the ProductCode identifier on its own, rather than by the combination of OrderNumber and ProductCode. IE: the ProductName should be in a relation separate from the OrderNumber-ProductCode relation. Thus we need to draw out a new relation from the OrderNumber-ProductCode relation above. In doing so we need to copy the ProductCode to the new relation so as to maintain the link or relationship between the two relations.

{ OrderNumber, OrderDate, ShippingDate, CustomerNumber, CustomerName, ShippingAddress, Phone }

{ OrderNumber, ProductCode, RetailPrice, Quantity },

{ ProductCode, ProductName }

NOTE: we could debate whether the RetailPrice goes to the later relation or stays in the original.

Third Normal Form

In this step, the Transitive Dependencies are removed. Again going back to our example, it can be seen that the Customer Details in the first relation are not best represented by the Identifying Attribute - OrderNumber. The Customer details are more appropriately represented by the CustomerNumber, which is not an Identifying Attribute in the first relation. This type of dependency (on a non-key attribute) is often referred to as a transitive dependency. To remove this dependency, the customer information can be removed to its own relation. To maintain the link or relationship, a copy of the CustomerNumber needs to be left in the OrderNumber relation.

These relations might be named so follows:

CUSTOMERS: { CustomerNumber, CustomerName, ShippingAddress, Phone }

ORDERS: { OrderNumber, OrderDate, ShippingDate, CustomerNumber }

ORDERLINES: { OrderNumber, ProductCode, RetailPrice, Quantity },

PRODUCTS: { ProductCode, ProductName }

NOTE: Something that is imbedded in this set of relations is that a given product can only be added as a line item once for any given order. This is not a problem as such. If a customer wants two of a single product, then two must be entered as the Quantity. Where relations such as OrderLines have two attributes that combine as the identifying attribute, an alternate Identifying Attribute could be added such as OrderLinesID.

E-R and Normalisation for Brenda (Partial)

Entities

From the Data Flow Diagrams and the Data Dictionary, the following entities might be suggested:

- Draw
- Team
- Player
- Game
- Manager

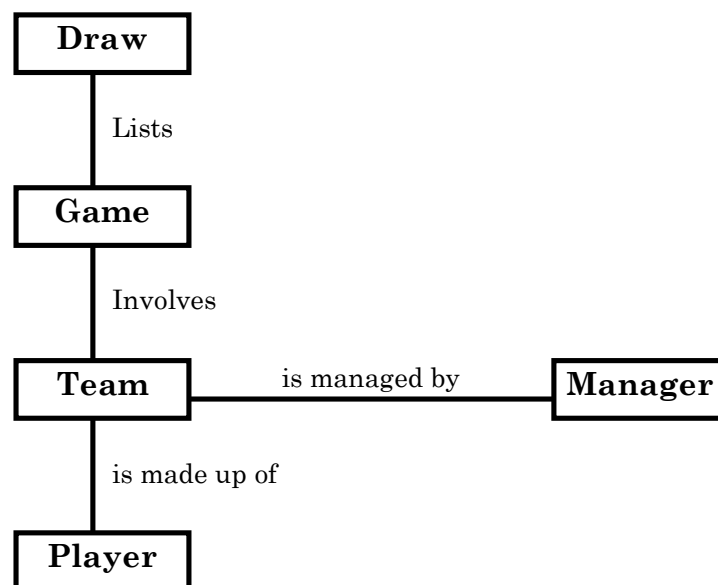
Relationships

As a first *pass* we could identify the relationships between these Entities as:

- A Draw lists Games
- A Game involves two Teams
- A Team is managed by a Manager.
- A Team is made up of Players

ER Diagram

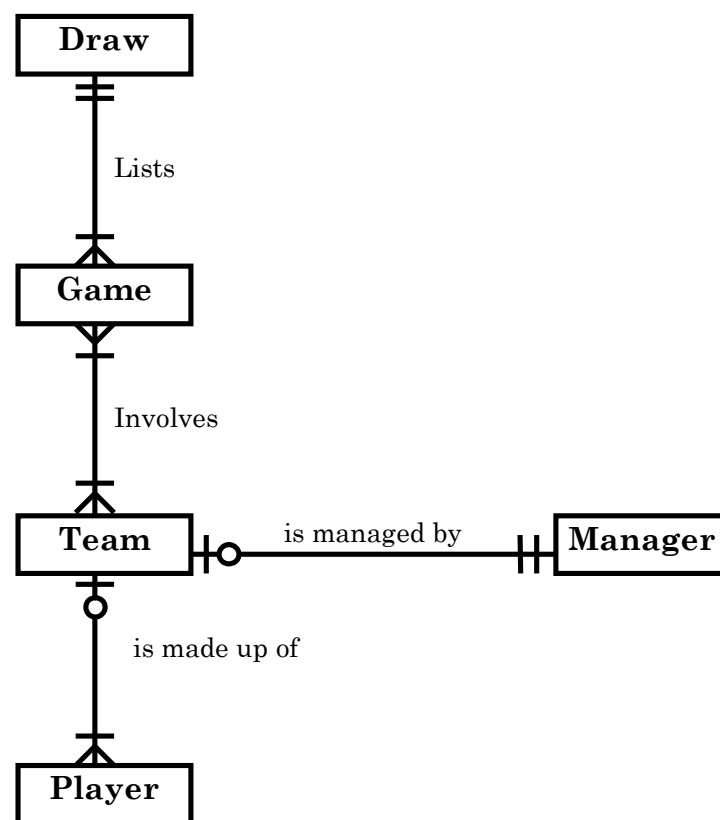
So a rough first draft of the ER Diagram might be:



In making a second pass, we might tie down or qualify the relationships as:

- One Draw lists many Games.
- One Draw must list at least one Game
- Each Game is listed on only one Draw
- Each Game must be listed on at least one Draw.
- A Game involves two and only two Teams
- A Team will play many Games
- A Team should play at least one Game
- A Team is managed by one Manager.
- A Team must have one Manager
- A Manager manages one Team
- A Manager may not be managing a Team currently
- A Team is made up of Players
- A Team must have at least one Player
- A Player is only on one Team (*this may not be a fair assumption!!?? This makes a significant difference if this is not the case*)
- A player may not be on a team currently.

Thus the revised ER Diagram might be represented as:



It is often difficult to represent a many-to-many relationship in a relational database structure. Often an Associative Entity can be placed between the two entities.

For Brenda's data structure, we could place an Associative Entity between Game and Team. It might be that the relationship between game and team is broken down to be:

A Game brings together two Competitors.
A Games must have two Competitors
A Team is a Competitor.
A Team must be a Competitor at least once.

Attributes

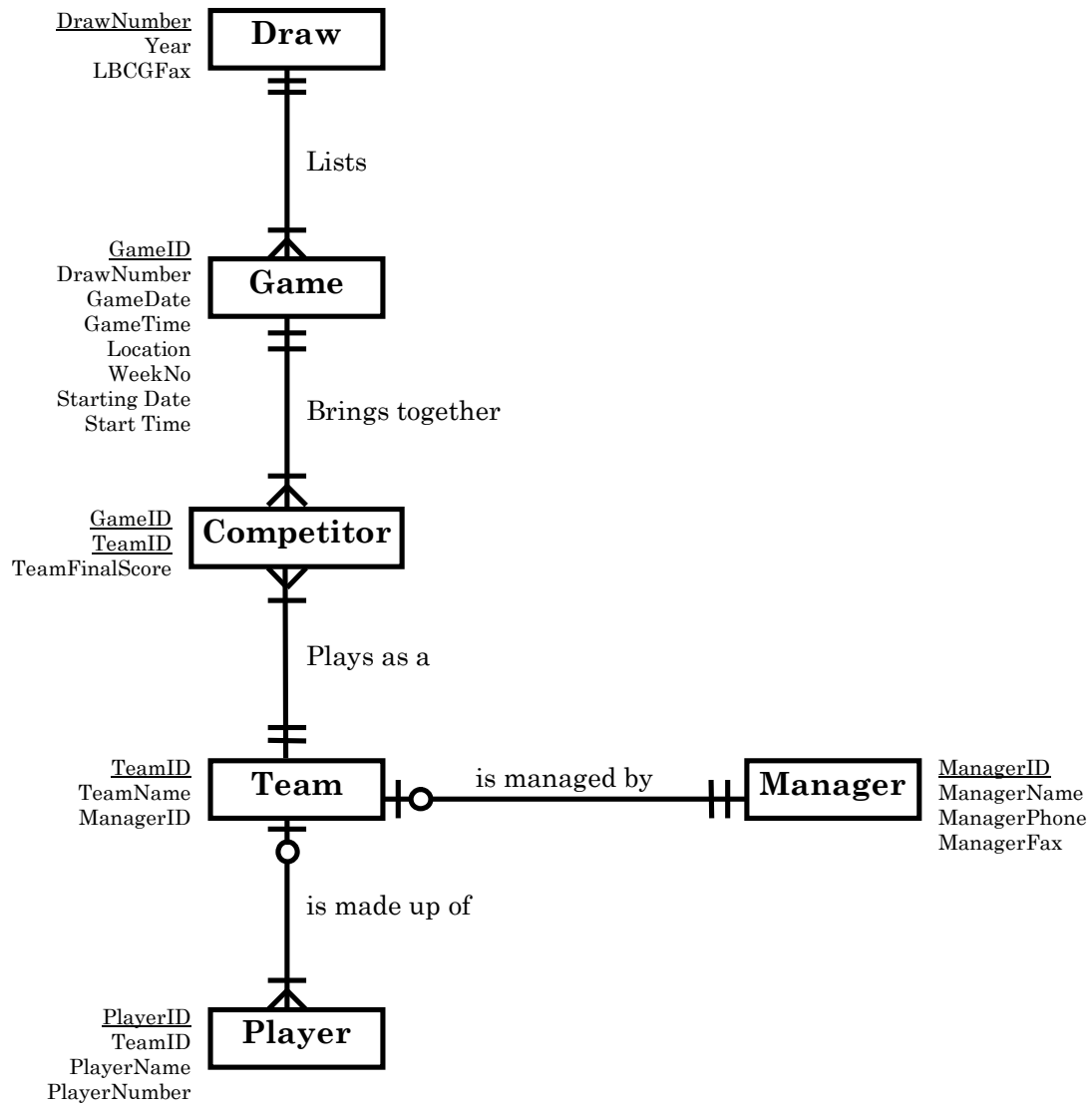
The Identifying Attributes for each of the above entities would be:

Draw:	DrawNumber
Game:	GameID
Competitor:	GameID, TeamID
Team:	TeamID
Manager:	ManagerID
Player:	PlayerID

The Descriptive Attributes for each of the above entities would be:

Draw:	Year, LBCGFax
Game:	GameDate, GameTime, Location, WeekNo, Starting Date, Start Time
Competitor:	TeamFinalScore
Team:	TeamName, ManagerID
Manager:	ManagerName, ManagerPhone, ManagerFax
Player:	PlayerName, PlayerNumber

A final diagram:



Normalisation

Original Un-normalised Data Elements

The DrawNo element is identified as an Identifying Attribute.

```
{ DrawNo, Year, LBCGFax, { WeekNo, StartingDate, { GameID, GameDate, GameTime,
Location, { TeamName, ManagerName, ManagerPhone, ManagerFax, TeamFinalScore {
PlayerName, PlayerNumber } } } } }
```

First Normal Form

In this step, the Inner Repeating Groups are removed. As a first pass, separate the draw from the Week information. The DrawNo must be added to the Week/StartingDate information. A unique identifier for the week (WeekID) has been added as the Identifying Attribute for the Week information.

```
{ DrawNo, Year, LBCGFax }
{ WeekID, WeekNo, DrawNo, StartingDate, { GameID, GameDate, GameTime, Location, {
TeamName, ManagerName, ManagerPhone, ManagerFax, TeamFinalScore, { PlayerName,
PlayerNumber } } } }
```

As a second pass remove the Game information from the Week information. The WeekID must be added to the Game information to maintain the relationship. The GameID would be sufficient as an Identifying Attribute for the Game information.

```
{ DrawNo, Year, LBCGFax }
{ WeekID, WeekNo, DrawNo, StartingDate }
{ GameID, WeekID, GameDate, GameTime, Location, { TeamName, ManagerName,
ManagerPhone, ManagerFax, TeamFinalScore, { PlayerName, PlayerNumber } } }
```

As a third pass remove the Team information from the Game information. The GameID must be added to the Team information. A TeamID has been added to identify the team.

```
{ DrawNo, Year, LBCGFax }
{ WeekID, WeekNo, DrawNo, StartingDate }
{ GameID, WeekID, GameDate, GameTime, Location }
{ GameID, TeamID, TeamName, ManagerName, ManagerPhone, ManagerFax,
TeamFinalScore, { PlayerName, PlayerNumber } }
```

As a forth pass remove the Player information from the Team information. The TeamID must be added to the Player information. A unique identifier for the player (PlayerID) has been added as the Identifying Attribute.

```
{ DrawNo, Year, LBCGFax }
{ WeekID, WeekNo, DrawNo, StartingDate }
{ GameID, WeekID, GameDate, GameTime, Location }
{ GameID, TeamID, TeamName, ManagerName, ManagerPhone, ManagerFax, TeamFinalScore }
{ PlayerID, GameID, TeamID, PlayerName, PlayerNumber }
```

Second Normal Form

In this step, the Partial Dependencies are to be removed. The team information for instance is currently in a relation that has as its Identifying Attribute: GameID and TeamID. The Team would be better identified by the TeamID alone. A copy of the TeamID would need to remain in the existing relation so as to retain the link or relationship. The link between the Team and the Player information would be reduced to TeamID.

```
{ DrawNo, Year, LBCGFax }
{ WeekID, WeekNo, DrawNo, StartingDate }
{ GameID, WeekID, GameDate, GameTime, Location }
{ GameID, TeamID, TeamFinalScore }
{ TeamID, TeamName, ManagerName, ManagerPhone, ManagerFax, }
{ PlayerID, TeamID, PlayerName, PlayerNumber }
```

Third Normal Form

In this step, the Transitive Dependencies are removed. The Manager information is currently in a group with team information and identified by the TeamID Identifying Attribute. This detail would be better placed in a separate group and have a ManagerID as an Identifying Attribute. Thus, the Manager information would be removed from the Team, and a ManagerID would be added to both to maintain the link between the groups.

```
{ DrawNo, Year, LBCGFax }
{ WeekID, WeekNo, DrawNo, StartingDate }
{ GameID, WeekID, GameDate, GameTime, Location }
{ GameID, TeamID, TeamFinalScore }
{ TeamID, TeamName, ManagerID }
{ ManagerID, ManagerName, ManagerPhone, ManagerFax, }
{ PlayerID, TeamID, PlayerName, PlayerNumber }
```

Adding titles to the groups:

```
DRAW: { DrawNo, Year, LBCGFax }
WEEK: { WeekID, WeekNo, DrawNo, StartingDate }
GAME: { GameID, WeekID, GameDate, GameTime, Location }
COMPETITOR: { GameID, TeamID, TeamFinalScore }
TEAM: { TeamID, TeamName, ManagerID }
MANAGER: { ManagerID, ManagerName, ManagerPhone, ManagerFax, }
PLAYER: { PlayerID, TeamID, PlayerName, PlayerNumber }
```

Some Thoughts on Tables

As a rough and ready alternate approach to determining tables for a database, the following might be a guide:

1. *The Entities*

When creating tables for a given system you often acquire one or more of the following: forms, reports, brochures, marketing documentation, and/or other printed data. The first thing you need to train yourself to do is to look for the obvious entities. For example:

- You look at a music CD cover and you see a CD with a title, artist, recording company etc. You also see Tracks - songs etc with title, artist, author and running time. The entities are CD and Track.
- You look at an order/invoice from a computer shop and you see an Order on a piece of paper, a Customer listed against the order, and Products listed on lines on the order. The obvious entities are Customer, Order, and Product.
- You go into a video shop and borrow 5 videos. Again you have the entities: Customer, Loan, and Video.
- You look at an enrolment form for your studies. You see the entities: Student, Enrolment and Subject or Module.
- You look at the contents of this paper. You see Main headings, Sub Headings and Sub-Sub Headings. Such are the entities of the data in this paper.
- You have a table of figures - labels across the top and down the left hand side, and quantities connecting the two in the middle, for instance:

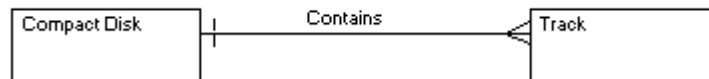
	Heading 1	Heading 2	Heading 3
Item 1	100	200	100
Item 2	50	150	100
Item 3	150	100	50

2. Many to Many and One to Many Relationships

The next task is to determine the relationships and their cardinality.

CD's and Tracks

By inspection, you can see that one CD (usually) has many tracks. Thus the relationship is one CD has many Tracks. (One track is usually only on one CD - *we will assume this for the moment*). You must put the CD_ID into the Tracks table - always copy the Primary Key from the ONE side into the table on the MANY side.



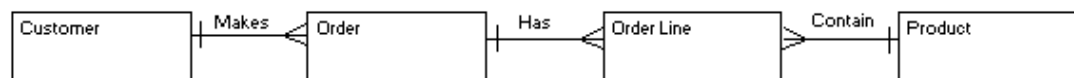
An Order Form / Invoice

By inspecting a copy of an Order, you may see that there can be many Products listed on one Order - one Order can have many products listed. However, one Product can appear on many Orders, giving us a many-to-many relationship. It is easier for a relational database to handle one-to-one or one-to-many relationships than many-to-many.

To simplify a many-to-many relationship, try adding a link table between the two and see if the link table can be considered as an entity in its own right.

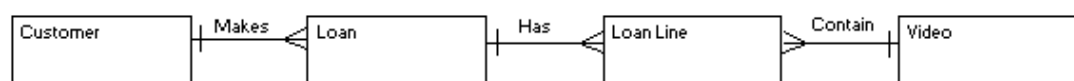
An Order has OrderLines (or OrderDetails). An OrderLine could be considered an entity in its own right. Testing the subsequent relationships: One Order has many OrderLines and an OrderLine appears on only one Order - a one-to-many relationship. An OrderLine can hold only one Product while a Product can appear on many OrderLines - also a one-to-many relationship. The original many-to-many relationship has been converted to two one-to-many's.

Orders also often have a Customer listed. One customer can have many Orders. One Order is taken up by one Customer - a one-to-many relationship. As an ER Diagram this example might appear:



Video Shop

Video shop loans closely mirror the Order Form example above. Customer can have many Loans, a Loan can have many Loan Lines, and a Loan Line contains a Video:



Student Enrolment

Although the enrolment, progress and certification of students through subjects or modules within a course is a relatively complex structure, an enrolment form depicts a relatively straightforward structure, mirroring the two examples above.

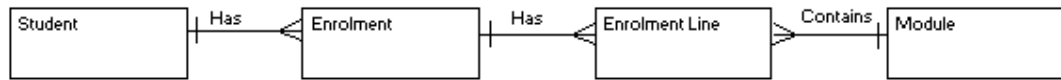


Table of Contents

Looking at the table of contents of this paper or of any book, the one-to-many relationships may jump out - a Main Section has many sub sections which in turn have many sub-sub sections:

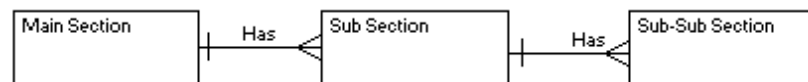
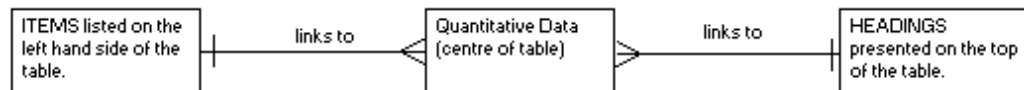


Table of Figures

A table of figures has items listed down the left side and headings listed across the top of the table. These items and headings, are in a many-to-many relationship. We might add a link table representing the quantitative data from the centre of the table of figures:



3. The Attributes

Based on the information at hand, you should be able to identify the primary key(s) for each entity. Keep in mind that the primary key has two fundamental requirements - it must contain NO DUPLICATES and NO NULLS.

You should also be able to identify the other attributes for each entity. List these along with their respective data types.

And finally, with respect to relating tables:

- One-to-one tables can have the same primary key.
- One-to-many tables require you to copy the Primary Key from the one side to the table on the many side.

4. Lookup Lists

Often repetitive data entry and/or the need to use consistent spelling for categories etc, can bring to mind a whole range of small 'lookup-list' tables:

- The CD and Tracks example might be assisted by look-up lists for Music Type and/or Music Distributors.

The 'trick' with lookup lists is that the entry of data into such lists requires far less attention than would be required to enter the data directly into the original table.

- For instance, take the customers for the Order Form or Video Loans examples. We could create a lookup list for suburbs, postcodes and states. This would work well for a video shop whose clientele are from no more than about 20 to 30 local suburbs. After typing in the 20 or 30 local suburbs, little or no ongoing suburb and postcode data would need to be added to the database. However in contrast, let us assume that the 'Orders' business was a national or international mail order business. A new suburb, postcode and state would need to be typed in for each new order, and a lookup list would offer little or no time saving.

5. A One to One Relationship

As a final point, it is worth mentioning that a common use for one-to-one relationships occurs when you wish to enter data against an entity only intermittently.

For instance, when filling in a survey or questionnaire you often get a comment section at the end of the paper. Some people fill this section in, others do not. If all the questionnaire data were to be entered into a database, then the memo field allowing for the entry of the comment could be placed in a separate table linked one-to-one with the question table. That way if a comment were to be entered, only then would a record be created in the comments table. Otherwise, if the memo field were placed in the survey table directly, then space would be allocated as soon as each new survey record was initiated.

1. *References and Resources*

- | | |
|-----------------------------|---|
| Dennis & Wixom | Systems Analysis and Design - An Applied Approach
John Wiley & Sons, Inc, New York, 2000 |
| D'Orazio & Happel | Practical Data Modelling for Database Design
Jacaranda Wiley Ltd, Brisbane, 1996 |
| Garner, Stuart | Systems Analysis and Design using Ascent, Version 2.1
Knowledge Base, Duncraig, 2000 |
| Hawryszkiewicz, Igor | Introduction to Systems Analysis and Design,
Fourth Edition
Prentice Hall, Sydney, 1998 |
| Kendall & Kendall | Systems Analysis and Design, Fourth Edition
Prentice Hall International, Inc, New Jersey, 1999 |
| Shelly, Cashman, Rosenblatt | Systems Analysis and Design, Fourth Edition
Course Technology, Thomson Learning, Boston, 2001 |