

# Database Management: Definitions

---

- Some basic definitions are required by way of introducing the concept of database management

## **Database**

- An organised collection of data
- A database is structured in a manner, such that the relationships which exist between different items or sets of data are explicit
- In general, a database will serve the requirements of a variety of users rather than a single individual
- If multiple users can share data, unnecessary redundancy in data storage can be avoided

## **Database Management System (DBMS)**

- A software package for the storage, manipulation and retrieval of data from a database

## **Query**

- The retrieval of all or a subset of the data stored within the database, by the database management system software, in response to a user-defined selection request

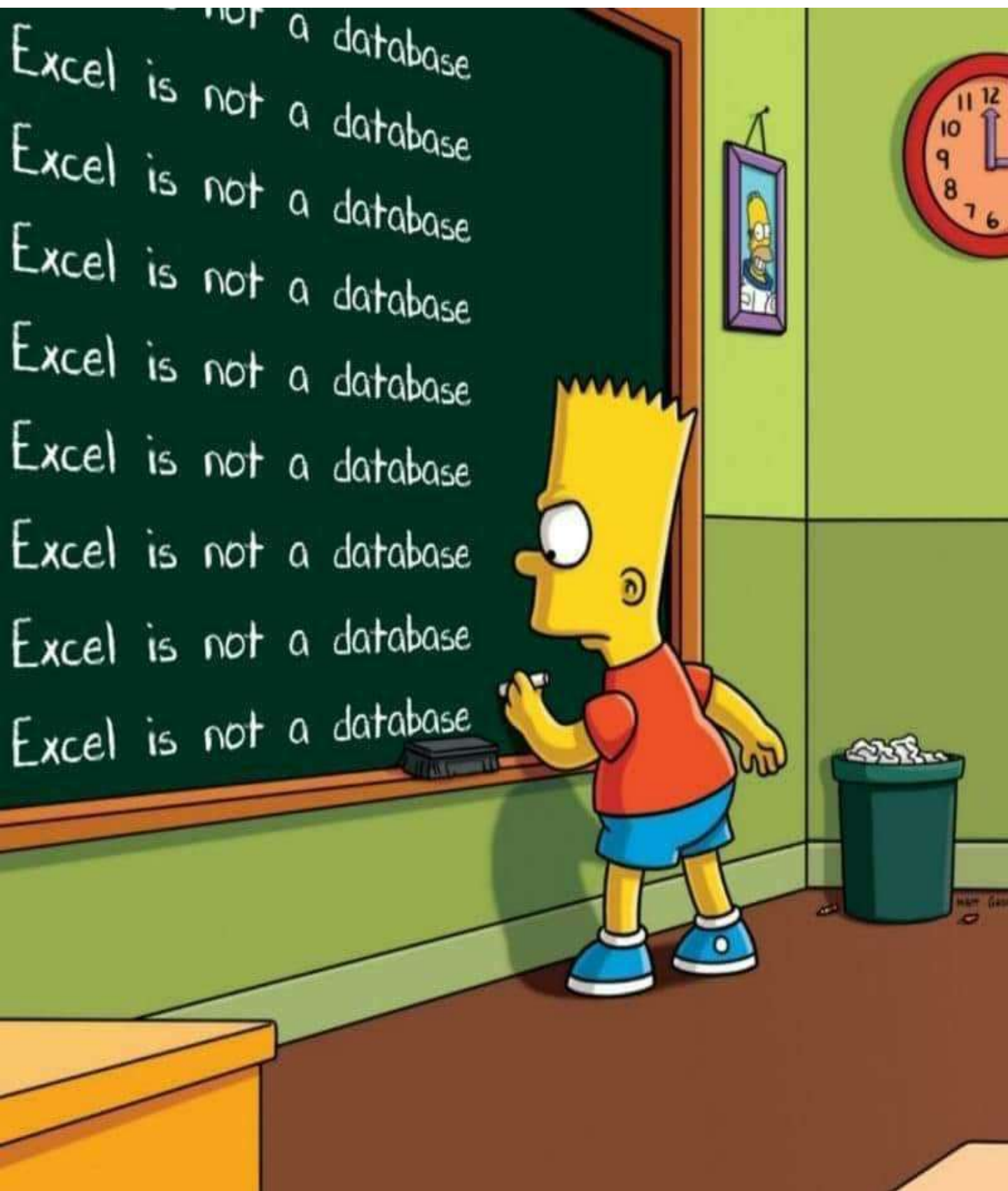
# What's different about a DBMS?

---

- A DBMS gives a level of sophistication in handling data beyond paper-based records or a simple 'computer file'
- Confusion sometimes exists between a DBMS and a spreadsheet
- Spreadsheets are primarily about doing calculations not storing data
- Consider the following analogy:
  - A *spreadsheet* is like a notebook. It is possible to enter data, but there is little control over its organisation, other than simple lines, and perhaps columns. Simple manipulations are possible. However, when you come back to the notebook some months later it may not be clear what the data are, how they were organised, or indeed how they could be combined with similar data collected later.
  - A *Database Management System (DBMS)* is like a card index. The data are organised into drawers, within these drawers into alphabetically-ordered sections, and within these sections into carefully ordered cards. Each card has the individual items forming the data record organised into different areas. Finding individual data records, subsetting or even re-ordering these is a relatively straight forward task.
- While it is easy to move data from a database to a spreadsheet, the converse may require much cleaning up of the data


# Excel is not a Database !

---



# History of Database Management

---

- Long history of Data Processing
  - First 6000 years by hand
  - Slow communication rates (by foot!)
- Mechanical Data Processing
  - From 19th C.
  - Communications still slow
- Electronic Data Processing
  - From 1940's
  - Still slow communications by human until the 1970s
- Properties of Human Systems
  - Collect and maintain their own filing systems
  - Not very efficient
  - Understand the meaning of their data, allowing for the detection and correction of mistakes
- None of these should apply to computer systems
- Development:

```
graph LR; A[SPECIFIC TO APPLICATIONS] --> B[GENERIC]; B --> C[ALL ENCOMPASSING]
```
- Today, comprise a core or kernel, with interfaces such as:
  - generic query interface
  - screen forms interface
  - bulk data loader
- Have become the very core of GIS

# Physical and Logical Database Design

---

## *Physical Design*

- This is concerned with the location of different parts of the database within the computer file system
- The database may be spread across multiple physical disk drives to balance i/o load or for security in the event of media failure
- The physical database design is the responsibility of the database administrator

## *Logical Design*

- This represents the user's view of the relationships between datasets stored in the database
- Logical and physical design should be quite separate, users should not have to concern themselves with where and how their datasets are physically stored (Martin, 1976)

# Data Analysis

---

- The first stage in logical database design
- Data analysis allows a clearly defined conceptual model of the relationships between different datasets to be developed, before the necessary resources are committed to actual implementation of the database
- Without a clear understanding of these relationships, it is likely that the database will be inefficient in design and will have a poor match to users' requirements
- A variety of possible data analysis or data modelling techniques have been developed. The approach of Chen (1976), based on the entity-relationship model, has perhaps met with the widest acceptance
- One of the most important aspects of data analysis techniques is the fact that they have associated with them clear and unambiguous diagramming standards
- The ability to represent the structure of a complex set of data interrelationships in a consistent graphical form is a powerful aid to understanding the problem and communicating the results of the data analysis to the intending users of the final database

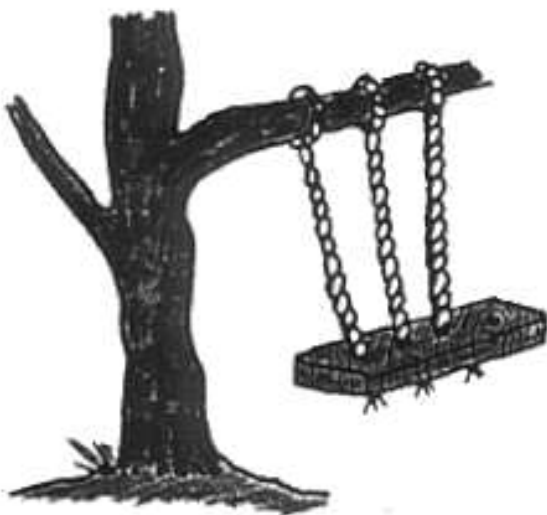
# Data Analysis and System Design



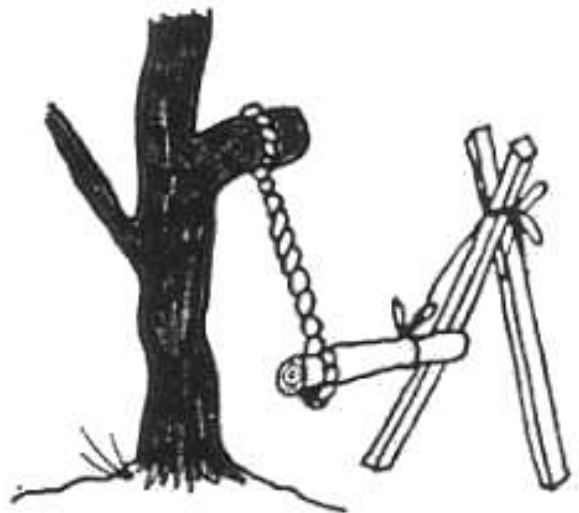
What the user asked for



How the analyst saw it



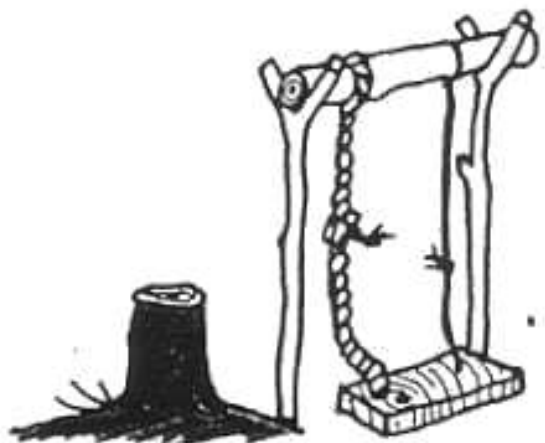
How the system was designed



As the programmer wrote it



What the user really wanted



How it actually works

Some end users would like to see some up experts strong up . . . and the feeling is probably mutual.

# Chen's E-R Model

---

- After Chen (1976)

## *Fundamental Concepts*

- Entity sets
- Attributes
- Domains
- Relationship sets
- Mappings

## *Entity sets*

- These represent the generic structure of 'objects' which are relevant to the specific database being designed. Examples would include towns, census districts, hotels

## *Attributes*

- Each entity belonging to a particular entity set will have a number of characteristics or attributes. These might include town populations, id numbers for census districts and street addresses for hotels

## *Domains*

- Each attribute has a range of possible values which constitute its domain or value set. Id numbers may be restricted to a range from 001 to 999 or hotel quality ratings between 1 and 5



# More E-R Concepts

---

## *Relationship Sets*

- Formally defined as subsets of the cross product of two or more entity sets (Tsichritzis and Lochovsky, 1982)
- The specific relationship between individual members of the respective entity sets provides the subsetting mechanism, eg. that certain members of the hotel set are located in a particular town
- Relationships are also called 'mappings'

## *Types of Mappings*

- One-to-One mappings  
eg. each town has one and only one set of municipal offices
- One-to-Many mappings  
eg. each town has a number of hotels
- Many-to-Many mappings  
eg. in a situation where wholesalers are distributing goods to different shopping centres, each centre will be served by multiple wholesalers and each wholesaler will distribute goods to several centres
- Any mapping may be:
  - **Mandatory**, where a relationship must exist  
eg. restaurants must have food dishes
  - **Optional**, where a relationship does not necessarily exist eg. not all restaurants are in towns
  - There is a *directional element* to these additional properties eg. towns must contain people, but people do not necessarily have to live in towns. Equally, restaurants must offer food dishes, but not all food dishes are found in restaurants (pot-noodles are only found in student flats!!)

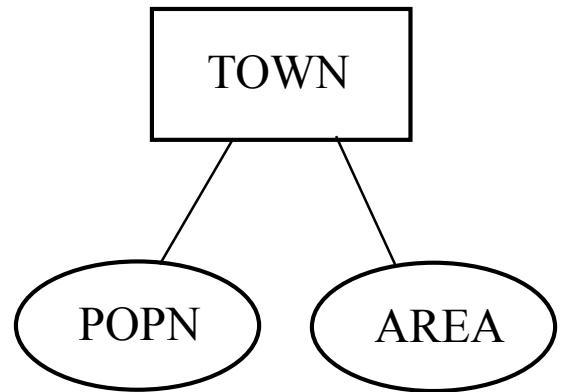
# E-R Diagramming Standards

---

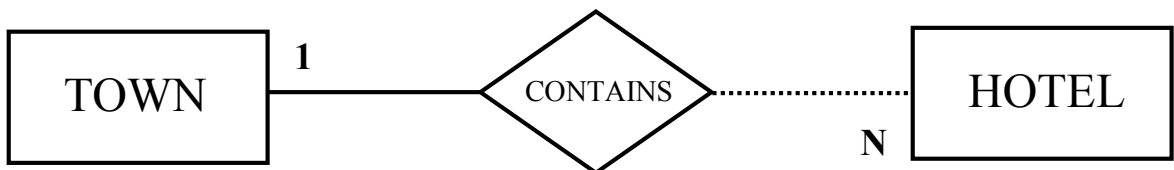
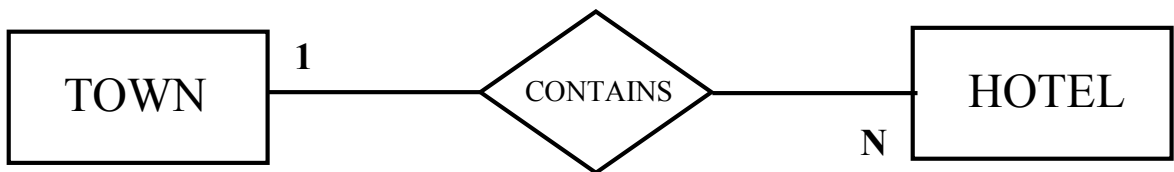
Entity Sets are drawn as a *rectangular box*



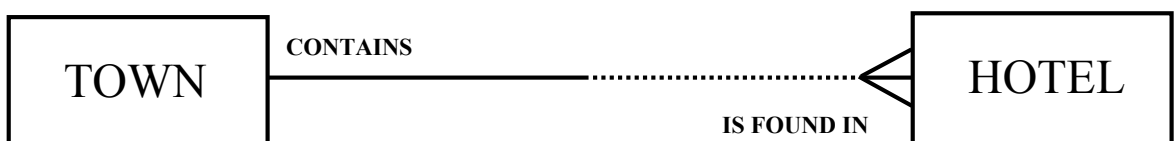
Attributes as *ovals*



Relationship Sets as *diamonds*



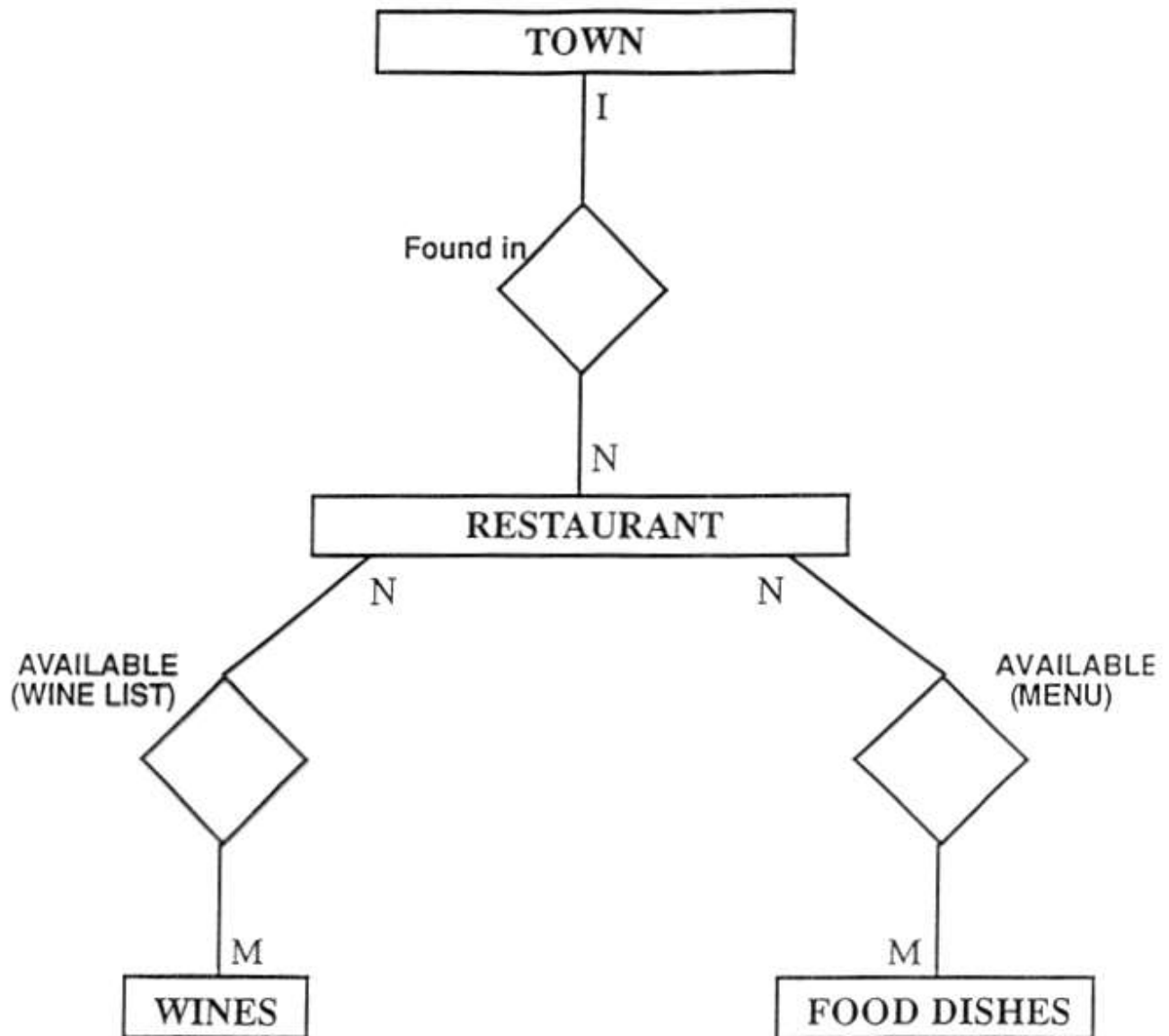
NB. Optional relationship



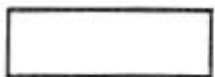
• **Be Consistent !!**

# A Complete E-R Diagram

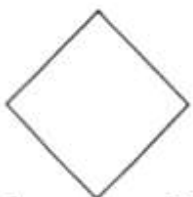
---



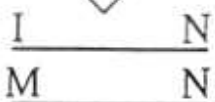
Key



Entity Set



Relationship Set



One-to-many mapping  
Many-to-many mapping

# Stages in Database Design

---

1. Define the problem for which you require database methods
2. Determine likely and appropriate data requirements and their sources
3. Construct the data model to show the structure of the data and the inter-relationships which require to be represented
4. Translate the model into an appropriate database structure
5. Implement the database

*A good and thorough database design will prevent the need for Stage 6.*

6. Re-design and re-implement the database

# Problems encountered in Data Modelling

---

## 1. Understanding the Problem

A very clear understanding of the user requirements is required at the outset.

## 2. Granularity of the Model

Too much detail?

## 3. Getting lost in the Detail

One approach is a 'top down' design, where the major entities are defined and then a separate Entity-Relationship Model is constructed for each, which contains the detailed sub-entities etc.

## 4. Should derived data be included?

eg. balance on customer accounts, or total population for a census district

## 5. Dealing with Time

There are a multitude of issues relating to time eg.

- Are the data dynamic in time?
- Continuous? Discrete intervals?

# Types of DBMS

---

- *Hierarchical Systems*
  - *Network Systems*
  - *Relational Systems*
  - *Object-Oriented Systems*
- 
- It is really only relational systems which are today established in the commercial marketplace
  - Broadly, the first two types represent older systems, and the chronological order also reflects an increasing reliance on theoretical considerations, in terms of design, rather than pragmatic data processing requirements
  - Hierarchical and Network systems have largely had their day
  - Some software packages may have characteristics drawn from more than one of the above types
  - In general, the category into which a specific software product falls, gives an indication of how it structures datasets and their interrelationships at the level of logical database design
  - Choice of system will have a major impact on the way the data model maps on to the database structure

# Hierarchical Systems

---

## *Background*

- Not based on a formal theoretical model of database design
- Follow the approach of the IBM IMS software, first released in 1968
- A specific entity set is defined to be the root of the hierarchical tree. Parent-child pointers from each level down to the one below represent the relationships between linked types of entity sets
- Each link at the given level must follow back through other levels to the root. No distinct sub-trees are allowed in a strict interpretation of the approach
- Each occurrence of an entity set is stored with its attributes and pointers to the level below

# Sample Data

---

- Restaurants*

Rest Id	Name	Address	Town
1	Peking Kitchen	29 Dublin Street	36
2	Giovanni's Trattoria	12 High Street	1
3	Frying Chipper	54 Newington Road	36

- Wines*

Wine Id	Name	Vintage
A	Gewurtztraminer	1998
B	Zinfandel	1997
C	Mosel	2000
D	Merlot	1996

- Wine Lists (similar for “food dishes”)*

Restaurant 1

Wine Id	Price
C	£8.95
D	£7.50

Restaurant 2

Wine Id	Price
B	£6.30
C	£8.15

Restaurant 3

Wine Id	Price
A	£14.50
D	£12.50

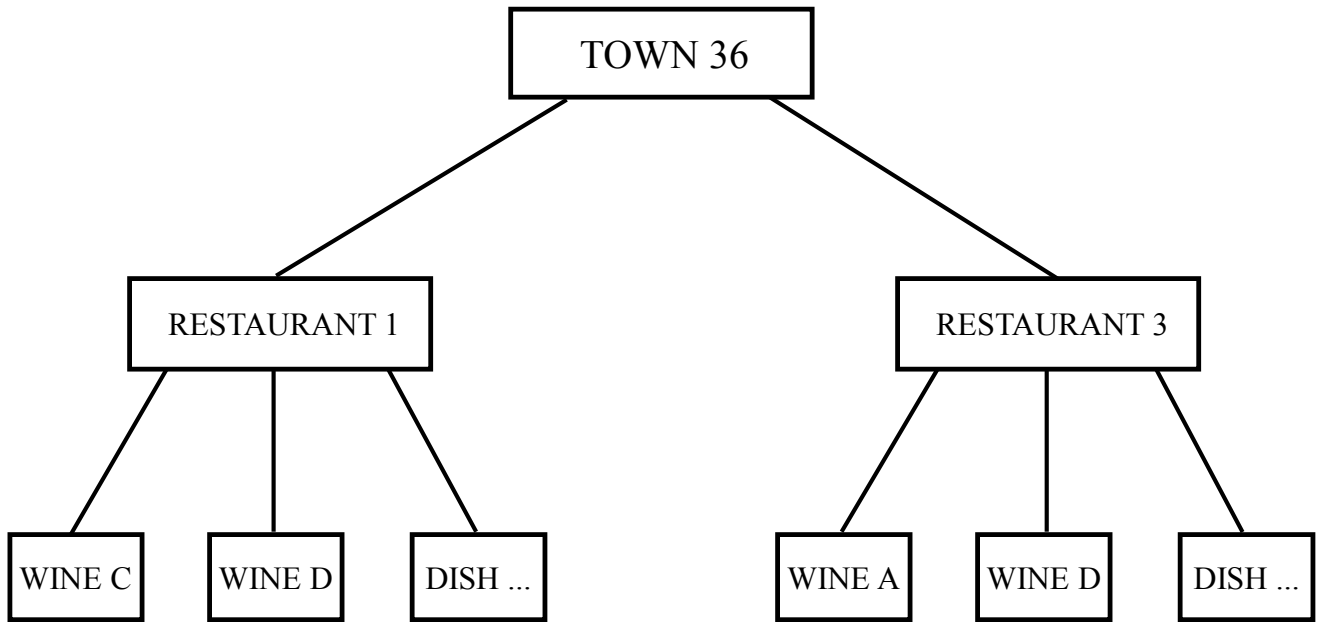
- Towns*

Town Id	Name
1	Dalkeith
36	Edinburgh



# Hierarchical Implementation

---



# Hierarchical Systems: Problems

---

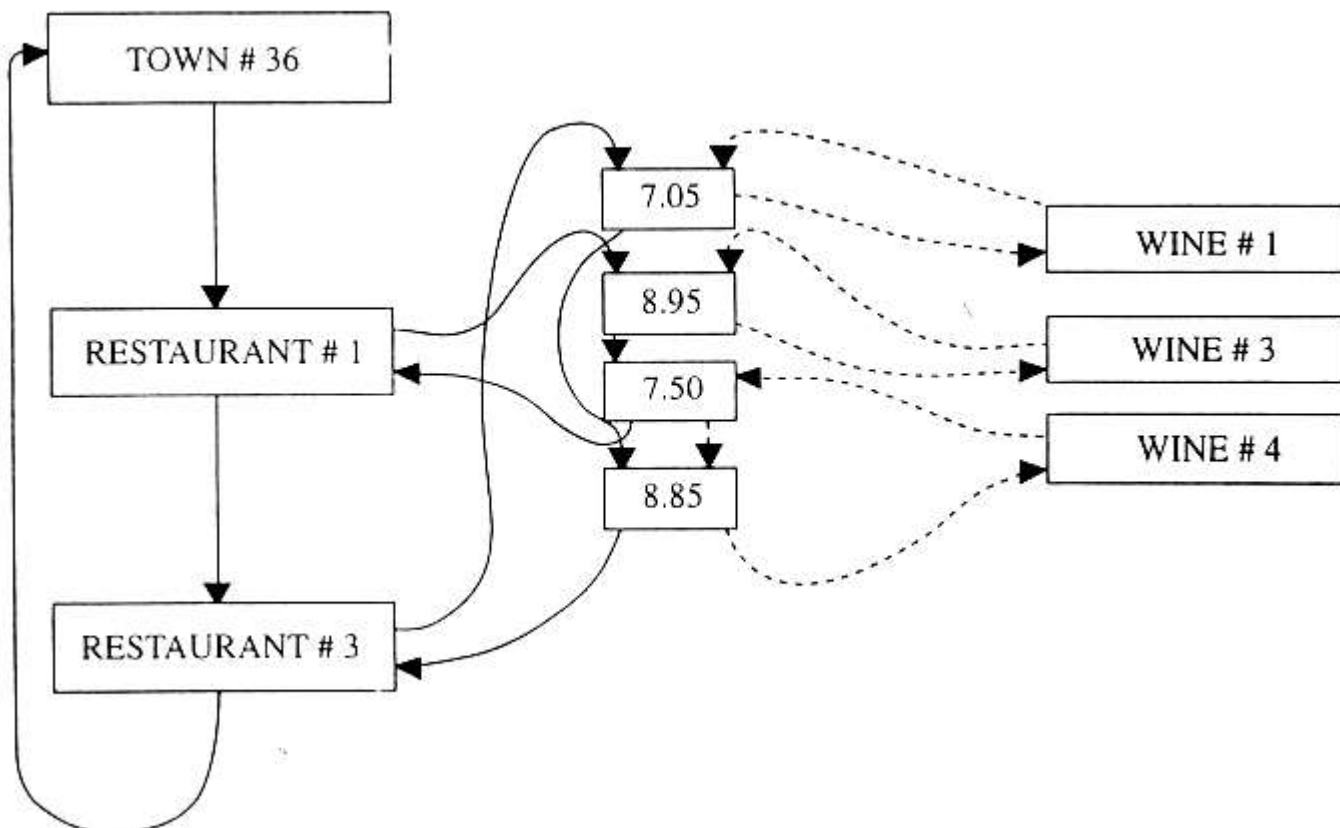
- Although one-to-many relationships can be handled effectively (town-restaurant), many-to-many relationships (restaurant-wine) are more difficult. Strictly speaking, every time a wine is found 'belonging' to a particular restaurant, for instance, its attributes have to be repeated, as there is no way of knowing where else in the database they may be found
- The solution to this problem is usually to violate the single tree structure and create separate sub-trees which classify the characteristics of entities such as the wines. Pointers from the main tree are then established to these sub-trees, as and when they are required
- For many applications, an hierarchical approach is inflexible because the record type that constitutes the tree structure has to be set up at the outset and is comparatively difficult to change afterwards
- Hierarchical structures make extensive use of internal pointer systems. These provide very fast access to data for problems that suit a tree-structured approach, but changing pointer structures if re-organisation of the data is required, is a resource intensive operation
- These factors and the level of computing skills required to manage these databases, have ensured hierarchical systems were quickly superseded, except of pure-hierarchy applications
- Hierarchical systems never had significant use in GIS applications

# Network Systems

---

## *Background*

- These are also called CODASYL systems, because they are based on the proposals of the Database Task Group of the Conference on Data Systems Languages (Olle, 1978)
- Specific implementations of these proposals include products such as IDMS and MDBS III
- Network systems differ from hierarchical systems in that a 'child' entity can have more than one parent and indeed any entity can be linked to any other
- Each entity set with its attributes is considered to be a node in the network. Relationship sets are handled as linkages (pointers) between individual entities in different entity sets



# Network Systems: advantages and disadvantages

---

- All the different forms of mappings can be handled using sets of pointers between entities
- The approach is powerful and relatively flexible
- For many applications it is also very fast and efficient in terms of computer processing resources
- There is no redundancy in data storage because all entity information is held in only one place and all linkages are handled by means of pointers
- From the implementation viewpoint, it may be comparatively difficult to set up the database correctly
- Although the query languages are comprehensive, they may appear complex to less expert or casual users
- Often the system is accessed via COBOL applications
- If major restructuring of the database is required, this may be time consuming because of the extensive structures that have to be rebuilt
- Network systems had some use in GIS:
  - Intergraph's DMRS / IGDS system
  - Were used in problems which require a combination of GIS and locational analysis techniques (Densham and Armstrong, 1987)

# Relational Systems

---

- Relational systems developed in the late 1970s and began to make a commercial impact in the mid 1980s
- These are the first class of database management systems to be based on a sound body of theory
- Relational DBMS are characterised by their simple structure
- Data are stored in *tables* (also known as relations) of *rows* and *columns*
- *Relational Table*

	COLUMN ↓		
ROW →			

- Examples include:
  - Oracle
  - Sybase
  - IBM DB2
  - MS SQL Server
  - MS Access
  - PostgreSQL (Free-and-open-source - FOSS)
  - MySQL (Free-and-open-source - FOSS)
  - Amazon Aurora (cloud-based)

# Relational Systems: Background

---

- The concepts of the relational approach were first set out by Dr Edgar F. (Ted) Codd (1969, 1970, 1979) as a means of describing data using their 'natural structure' only
- A further aim was to ensure independence of user-written application programs from the detailed storage formats of data within the database
- Firmly grounded in the mathematics of *set theory*, *predicate logic* and *relational algebra* (Ullman, 1982)
- Relational systems are characterised by simplicity of structure
- All the data are stored in tables (relations) of rows and columns
- Ignoring indexes, used to enhance performance, no pointers are used at all in relational systems
- From the design viewpoint, E-R modelling fits very closely with relational systems:
  - Each entity set is represented by a table
  - Each column represents one of the attributes of the entity
  - Each row (or tuple) in a table represents the data for an individual entity.
- Many-to-many relationships are explicitly represented by tables of data values, rather than by hidden pointer structures

# Many-to-Many Relationships

---

- Consider the wine lists from the sample data

**Restaurant 1**

Wine Id	Price
C	£8.95
D	£7.50

**Restaurant 2**

Wine Id	Price
B	£6.30
C	£8.15

**Restaurant 3**

Wine Id	Price
A	£14.50
D	£12.50

- These are represented in a *relationship table* as follows:

Rest Id	Wine Id	Price
1	C	£8.95
1	D	£7.50
2	B	£6.30
2	C	£8.15
3	A	£14.50
3	D	£12.50

# Oracle

---

- Market-leading relational database management system
- History:
  - 1974-79 IBM develop 'System R' as a research project based on Ted Codd's relational model theory
  - 1976 Concepts of 'System R' and Structured Query Language (SQL) published
  - 1977 Oracle Corporation formed by ex-IBM staff to undertake commercial development of a DBMS based on 'System R'
  - 1979 Version 1.0 of Oracle released, first commercial implementation of SQL

(IBM developed 'System R' into SQL/DS (1982) and eventually DB2 (1985))
- The SQL query language is the de facto industry standard for accessing relational DBMS
- Oracle is now used widely throughout the world for:
  - Financial and banking systems
  - Human Resources (HR)
  - Sales and Customer ordering
  - Scientific data etc. etc.
- Oracle provide a range of customised applications
- Most GIS systems have an interface to Oracle at least for their attribute data
- Oracle supports object-oriented structures lying over the relational core (object-relational technology)



# SQL

---

- A database query language with a natural structure, close to English
- Two sets of statements concern us:
  - **Data Definition Language (DDL)**  
Used to create the various database objects (tables, views, indexes etc.)
  - **Data Manipulation Language (DML)**  
Allows interaction with the data within the database, to retrieve, subset, create new data and update existing data
- The most often used is the SELECT statement, for example:

```
SELECT FIRSTNAME, SURNAME, DOB FROM MYTABLE  
WHERE DOB > '01-JAN-70'  
ORDER BY SURNAME
```

- Retrieves *sets of rows* from a relational database (although the set may be restricted to one row)
- SQL is not a procedural language, which iterates through data records, on a record-by-record basis, deciding which to process further and which to not to use
- This gives rise to significant economies in programming
- There were competitor query languages eg. Ingres' QUEL, but this died in 1986
- SQL has become an ANSI and ISO standard

# Relational Concepts

---

## 1. The Primary Key

- Each table (relation) represents a set
- Mathematically, sets cannot have duplicate values, so tables cannot have rows whose entire contents are duplicated
- Since every row must be different from every other, there must be a single column value or a combination of multiple column values that can be used to define a primary key for the table, which will allow each row to be uniquely identified
- No column that forms all or part of the primary key can have null values, otherwise the unique identification property will be lost
- The uniqueness property means that the primary key will serve as the sole row-level addressing mechanism in the relational database model (Date, 1986)

# Relational Concepts II

---

## 2. Relational Joins

- Since there are no pointer structures for linking different entity and relationship sets, a different mechanism must be employed that utilises only the data available in columns in different tables
- This mechanism is called a *relational join*.
- Conceptually, this involves matching data values in a column or columns in one table to corresponding values in a column or in a second table
- When matching rows in a second table are found, other values from those rows can be used to match to a third and so on, as indicated in the accompanying example
- Matching is frequently based on a *primary key* in one table linked to a column in the second which is termed a *foreign key*
- Thus a relational join is always based on a primary key - foreign key pairing, forming either end of the join
- For joins to work, the interrelationships between entity sets must have been correctly identified at the data analysis stage, so that the correct foreign keys are included
- Columns used in joins should be defined using the same domain of values
- The foreign key should not be allowed to contain values which are excluded from the list of allowable values for the primary key in the other table, otherwise an impossible join condition will result

# The Relational Join

RESTAURANTS TABLE

Rest Id	Name	Address	Town
1	Peking Kitchen	54 High Street	Dalkeith
2	Giovanni's Trattoria	1 Commercial Street	Leith
3	Stac Polly	29 Dublin Street	Edinburgh

P. Key

WINE LIST TABLE

Rest Id	Wine Id	Price
1	C	£8.95
1	D	£7.50
2	B	£6.30
2	C	£8.15
3	A	£14.50
3	D	£12.50

WINES TABLE

Wine Id	Name	Vintage
A	Gewurtztraminer	1998
B	Zinfandel	1997
C	Mosel	2000
D	Merlot	1996

P. Key

- Note that the 'Wine List' table is a slightly unusual relationship table, because it contains an attribute (of the relationship)

# Defining Keys

---

- In addition to *primary keys* and *foreign keys*, there are three other types of key which need to be briefly considered:
  - Concatenated Keys
  - Candidate Keys
  - Alternate Keys

## 1. Concatenated Keys

- A concatenated key is any key which is composed of more than one column
- All columns together form the key

## 2. Candidate Keys

- A candidate key is a column (or set of columns) which uniquely identifies a row in a table
- If the column (or in the case of a concatenated key, any one column forming part of the key) is removed the ability to be able to uniquely identify a row is liable to be lost
- By definition, a table will have at least one candidate key, but it may have more than one
- If a table has only one candidate key, then this becomes the *primary key*
- If there are several candidate keys, then a decision has to be taken as to which become the primary key
- Some candidate keys may make better primary keys than other

## 3. Alternate Keys

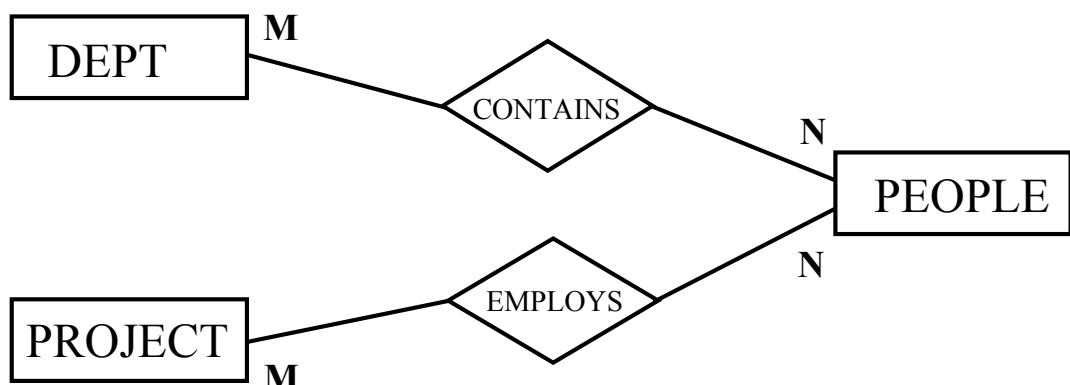
- Those candidate keys which are not defined as the *primary key* are known as *alternate keys*

# Preventing Redundancy

- By definition the join mechanism introduces necessary redundancy into a relational database.
- Unfortunately, without careful table design it is possible to introduce further unnecessary redundancy
- This can be a *major problem* !!
- To prevent this a clear understanding of Codd's Theory of Normal Forms is required. This specifies what values columns may contain and how columns in a table to be dependent on the primary key
- Consider the following:

Dept. No	Dept Name	Employ. No.	Employee Name	Address	Project	Deadline
1	Highways	210	James White	21 High Street	GIS	10th January
1	Highways	212	Joe Brown	5 Main Road	Environmental	16th March
1	Highways	212	Joe Brown	5 Main Road	GIS	10th January
2	Social Work	210	James White	21 High Street	GIS	10th January
2	Social Work	217	Carol Green	2 Long Row	GIS	10th January
2	Social Work	231	Andrew Rose	17 Edinburgh Lane	GIS	10th January
2	Social Work	215	Pauline Black	1 Smart Street	GIS	10th January
2	Social Work	215	Pauline Black	1 Smart Street	Environmental	16th March
3	Administration	215	Pauline Black	1 Smart Street	GIS	10th January
3	Administration	215	Pauline Black	1 Smart Street	Environmental	16th March
3	Administration	225	Jenny Violet	92 Main Road	Environmental	16th March

- Had data analysis been properly undertaken, three entities would have been identified:
  - Departments
  - People
  - Projects



# Preventing Redundancy II

---

- This would give three entity tables, with no redundancy, thus:

Dept. No.	Dept. Name
1	Highways
2	Social Work
3	Administration

Employee No.	Employee Name	Address
210	James White	21 High Street
212	Joe Brown	5 Main Road
217	Carol Green	2 Long Row
231	Andrew Rose	17 Edinburgh Lane
215	Pauline Black	1 Smart Street
225	Jenny Violet	92 Main Road

Project	Deadline
GIS	10th January
Environmental	16th March

- The last table could have a better primary key than “Project”, for example a “Project No.” could be added
- In each case above, a many-to-many relationship exists between the tables, but this has yet to be encoded
- Two further relationship tables will have to be created:

Dept. No.	Employee No.
1	210
1	212
2	210
2	217
2	231
2	215
3	215
3	225

Employee No.	Project
210	GIS
212	Environmental
212	GIS
217	GIS
231	GIS
215	GIS
215	Environmental
225	Environmental

- This introduces a small amount of necessary redundancy. In some cases (one-to-one and certain one-to-many relationships) these extra tables would not be required
- Using the *relational join*, it is perfectly possible to *temporarily* display these tables back in the un-normalised form if really required

# First Normal Form

---

## First Normal Form

- To be in first normal form, tables must contain rows and column values must be atomic
- That is, repeating groups of data, such as multiple values of a census variable for different years, are not permitted

Example:

NAME	POPN1971	POPN1981	POPN1991	POPN2001
Dalkeith	23,899	24,104	24,295	26,321
Penicuik	N/A	12,321	13,988	14,450
West Linton	N/A	2,300	3,449	2,995
Musselburgh	N/A	N/A	19,340	23,411
Aberlady	N/A	N/A	1,650	1,932

Should be rendered as:

NAME	YEAR	POPN
Dalkeith	1971	23,899
Dalkeith	1981	24,104
Dalkeith	1991	24,295
Dalkeith	2001	26,321
Penicuik	1981	12,321
Penicuik	1991	13,988
Penicuik	2001	14,450
West Linton	1981	2,300
West Linton	1991	3,449
West Linton	2001	2,995
Musselburgh	1991	19,340
Musselburgh	2001	23,411
Aberlady	1991	1,650
Aberlady	2001	1,932

- Two reasons:
  - Avoids null values
  - Allows unlimited expansion in terms of additional census years



# Second Normal Form

---

- Second normal form requires that every column which is not part of the primary key must be fully dependent on the primary key
- This requirement can be understood by considering the next example, which shows a table that is not in second normal form:

PRIMARY KEY

Rest Id	Wine Id	Wine Name	Price
1	C	Mosel	£8.95
1	D	Merlot	£7.50
2	B	Zinfandel	£6.30
2	C	Mosel	£8.15
3	A	Gewurtztraminer	£14.50
3	D	Merlot	£12.50

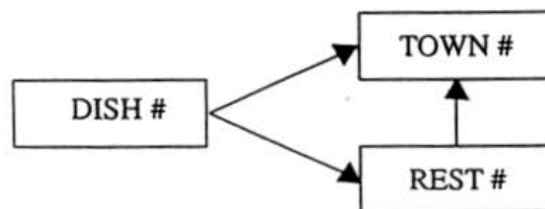
- Wine name is dependent on the wine-id number, because it is an attribute of the wines entity set, but is not dependent on restaurant-id
- The effect of not meeting the requirement can be seen by the introduction of unnecessary redundancy into the table in rows 1 and 4 and in rows 2 and 6
- If the wine name in the first row is updated and the corresponding change in the fourth row is overlooked, the database will be left in an inconsistent state

# Third Normal Form

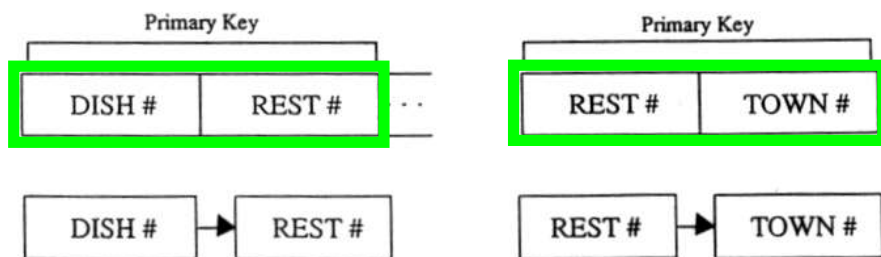
- The third normal form requires that every non primary key column must be non-transitively dependent on the primary key. The following example shows a table structure that is not in third normal form

Primary Key		
DISH #	REST #	TOWN #
1	1	36
2	4	18
3	1	36
4	2	46

- The example may appear to be a convenient way of indicating which menu items (dish id) are available in which towns, but unnecessary redundancy appears in rows 1 and 3



- It is apparent that the town id is related to the dish id via the restaurant id, ie. two distinct relationship sets have been conflated. This is the transitive dependence that we wish to avoid
- Instead, the relationship set tables should be kept distinct



- This eliminates the redundancy while making it very easy to add new dish/restaurant or restaurant/town combinations to the database

# Further Normal Forms

---

## Boyce-Codd Normal Form (BCNF)

- The second and third normal forms assume that all attributes not part of the candidate keys depend on the candidate keys
- They do not deal with dependencies *within* the keys
- BCNF deals with such dependencies
- Most relations that are in 3NF are also in BCNF. Infrequently, a 3NF relation is not in BCNF

## Fourth Normal Form

- Under fourth normal form, a record type should not contain two or more independent multi-valued facts about an entity. In addition, the record must satisfy third normal form.
- Fifth Normal Form is available to deal with more specialised design requirements
- Something called Domain/Key Normal Form (DKNF) has also been proposed (Fagin, 1980)

## Summary

- To summarise the normalisation process:  
"In general, each attribute of a table should represent a fact about the primary key, the whole primary key and nothing but the primary key" (Kent, 1983)

# Rules for Converting an E-R Model into Database Tables

---

- In translating a data model into a set of relational tables, it is often necessary to explicitly define a relationship table

## 1. One-to-One Relationships

- If membership of the relationship is mandatory for all occurrences of each of the two entities involved, put all the attributes into a **single table**.
- If membership is mandatory for only one of the two entities, define **two** tables, one for each entity. Post the *primary key* of the non-mandatory entity into the table of the mandatory entity as a *foreign key* column.
- For example, the case where a town can only have one Lord Provost (Mayor), a Provost must represent a town, but only larger towns have one, the table structure could be as follows:

PROVOST-ID	SURNAME	INITIALS	SALARY	TOWN-ID

TOWN-ID	NAME	POPEN	LOCATION	PARTY

- (the wrong way round gives lots of nulls!)
- If membership is non-mandatory for both entities, define **three** tables, one for each entity and one as the relationship table

# Rules for Converting II

---

## 2. One-to-Many Relationships

- If membership of the 'many' entity is mandatory, define **two** tables, one for each entity. Post the *primary key* of the 'one' entity into the table of the 'many' entity as a *foreign key* column
- For example, the case where population totals for a town are available for several census years. However, since each population total refers only to one specific town, the membership is mandatory. The table structure could be as follows:

TOWN-ID	LATITUDE	LONGITUDE	NAME

POPREC-ID	POPN	YEAR	TOWN-ID

(In this case the POPREC-ID may not be necessary, because the columns POPN, YEAR and TOWN-ID together make a concatenated primary key)

- If membership of the 'many' entity is non-mandatory (optional), define **three** tables, one for each entity and one as the relationship table

## 3. Many-to-Many Relationships

- All many-to-many relationships require the definition of **three** tables.

[illegible]

# Relational Systems: Advantages

---

## Advantages

- A rigorous design methodology based on sound theoretical foundations
- All other types of database structure can be reduced to a set of relational tables, ie. the relational representation is the most general
- This includes object-oriented structures
- Modifiability - new tables and new rows of data can be added without difficulty and without the need to reconstruct large networks of pointers
- The relational join mechanism, which allows flexibility in ad hoc data retrieval because the linkages (joins) between tables are established temporarily at query time
- Easy to use, to understand and to implement compared to other types of system
- Powerful and standardised query language facilities using SQL
- The use of a standard route to access these systems means applications can be easily designed, maintained and ported between different vendor's DBMS
- This has given rise to a significant range of applications, with RDBMS forming the underlying data store for major applications such as GIS
- Rich variety of data types, from simple character, number and date, to BLOBs and user-defined types

# Relational Systems: Disadvantages and Summary

---

## Disadvantages

- The simple structure from the user viewpoint hides a complex internal structure, which requires significant processor power
- Queries involving multiple relational joins can result in response times that are slower than would be desired
- BUT the powerful indexing mechanisms and other optimisation strategies can ensure consistent performance, even under heavy user load
- However, these mechanisms have to be used and some can be complicated to set up
- Further, it is all too easy for inexperienced users to create a badly designed and inefficient database structure, by failing to undertake proper data analysis prior to implementation
- Not all relational systems live up to all the advantages (eg. MS Access), although at least one sophisticated system (PostgreSQL) is free

## Summary

- The important advantages of the relational approach and the availability of good proprietary software systems have contributed greatly to the rapid adoption of this technology in the GIS field since the early 1980's
- Relational systems now dominate the DBMS market in the GIS sector and this is expected to continue for the foreseeable future



# Object Oriented DBMS

---

- Some data does not fall naturally into relational tables
- One approach to modelling the real world is to break it down into a series of interacting objects
- These points have given rise to a further class of database management systems: OODBMS
- In terms of GIS, the need for eg. user-defined datatypes are a tacit recognition that relational systems may be inadequate
- For GIS purposes it must also be possible to access the operations to be performed on these elements. This brings us very close to the object-oriented approach
- Few people agree on exactly what the term ‘object oriented’ actually means (Rowe and Stonebraker, 1987) or what it does or does not include
- Further confused by the fact that one class of OODBMS has grown out of OO programming methods (persistent extensions to C++)
- Generally:
  - OODBMS permit a better representation of the real world
  - Potentially work well with GIS
  - But often over-complex
  - Lack of standards
  - Not reliable in commercial environment
  - Therefore *relational* and *object-relational* systems preferred

# Conclusions

---

- We have seen importance of database management
- There are a range of systems available
- Data modelling is critical to successful and efficient design
- Within GIS:
  - Attribute data has been handled with DBMS since the early 1980s
  - This is key to the flexibility of attribute-based analysis
  - The principles of database management mean there are problems of maintaining entity-integrity when only part of the database lies within a formal DBMS
  - For this reason, and to take advantage of the other benefits of formal database management techniques, spatial data is increasingly being held within the DBMS
- It is primarily relational systems which have had an impact in GIS, although object-relational and O-O DBMS have a place
- Much more in the *Advanced Spatial Databases Option*

# Spatial Modelling Assessed Practical 1

## Data Modelling / Database Design Exercise

A regional archaeologist wants to set up a database to keep track of important archaeological finds that lie in a number of farmer's fields. This database will provide the definitive record of these finds and be used by various individuals to assist in both protecting the sites and in aiding the local farmers in their agricultural activities.

You have been chosen as the consultant responsible for designing and implementing this database. In your user requirements assessment certain specifications and constraints have become evident:

- 1) The database is required to support flexible querying, using both spatial and aspatial search conditions – this must support **both** the work of the archaeologists and of the farmers.
- 2) The database should be capable of supporting management information reports.
- 3) A simple spatial referencing system is to be used, as illustrated on the attached schematic map. This map provides in addition, the locations of both the archaeological finds and individual fields.

You have defined the **entities** that will exist within the data model as follows:

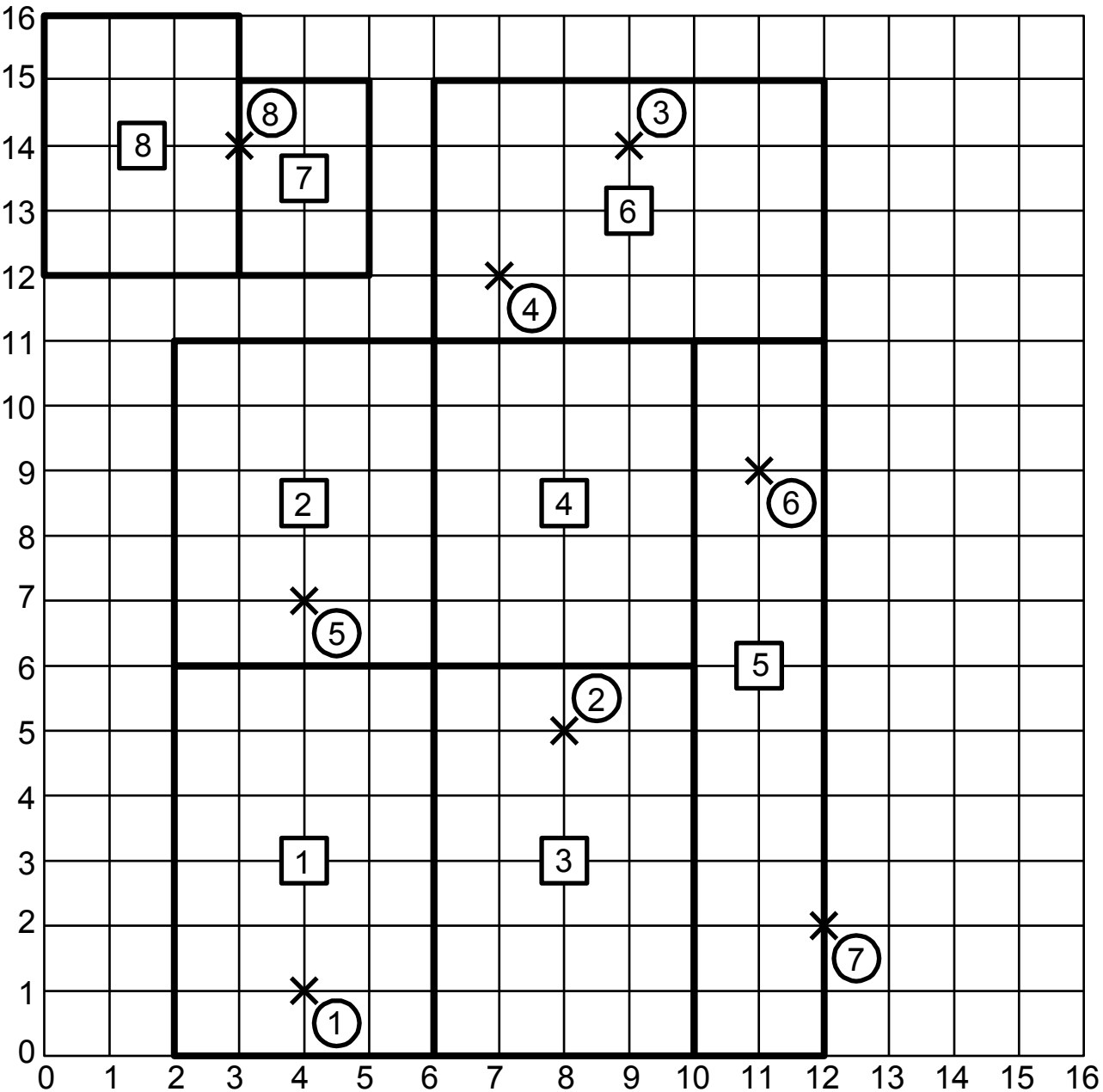
- **FARMER'S FIELDS**  
Data pertaining to the individual fields, including the spatial extent and ownership information.
- **SPECIFIC ARCHAEOLOGICAL FINDS RECORDED**  
Details of the individual finds, including information such as depth and allowing for comment on condition, decoration, damage, orientation etc.
- **CROPS**  
Information on crops which are grown in the various fields. This would include a crop name and the definition of their growing season.
- **ARCHAEOLOGICAL ARTEFACT CLASSES**  
Information on the types of artefacts into which the individual archaeological finds can be classified. This would include the name of the artefact class, the age and the use of the particular object.

You are required to:

- 1) Beginning with **entities** you have outlined above, identify the appropriate **attributes** which are required to meet the specified user requirements.
- 2) Identify appropriate **domains** for each attribute set.
- 3) Generate an **Entity-Relationship** (E-R) Model using a standard diagramming methodology.
- 4) Define a set of **normalised** relational database table structures that can be used to implement the model.
- 5) Identify the **keys** that are necessary in the database implementation.

**NB:** Any assumptions or “enterprise rules” which you make on the basis of the user requirements specified above must be explicitly stated, rather than implicitly assumed in the construction of the data model.

# Field Plan



key:



FIELD ID



FIND ID