Database Design

Database Design

Adrienne Watt

Port Moody

Contents

Preface

The primary purpose of this text is to provide an open source textbook that covers most Introductory Database courses. The material in the textbook was obtained from a variety of sources. All the sources are found in the reference section at the end of the book. I expect, with time, the book will grow with more information and more examples.

I welcome any feedback that would improve the book. If you would like to add a section to the book, please let me know.

Alternate Formats

This book is also available for free download in the following formats:

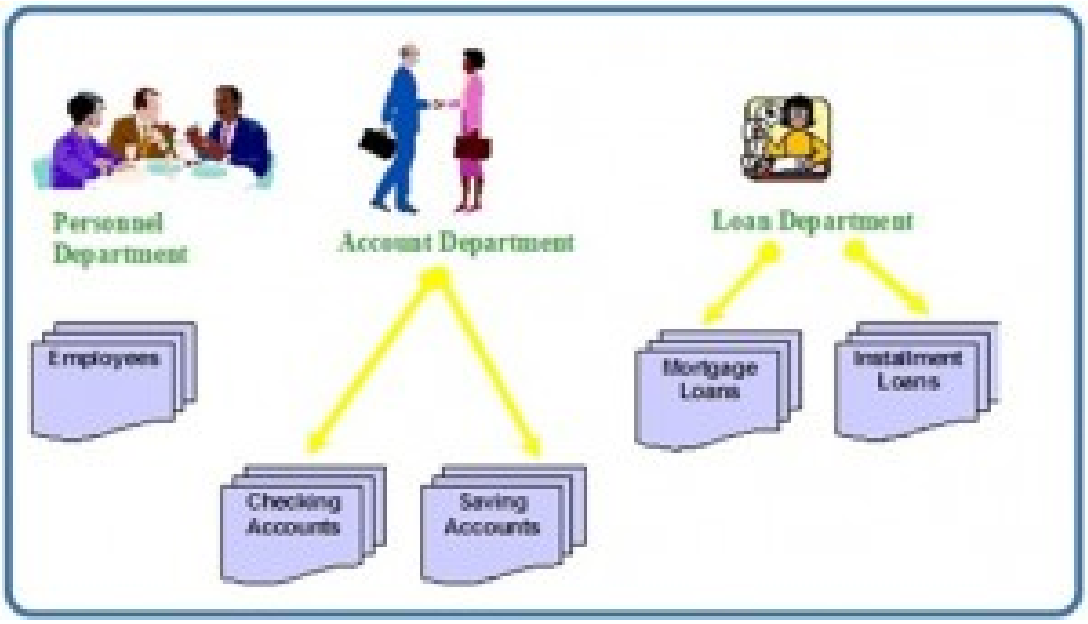
- ePub (for Nook, iBooks, Kobo, etc.)
- MOBI (for Kindle)
- PDF (for printing)

1

Chapter 1 Before the Advent of Database Systems

One way to keep information on a computer is to store it in permanent files. A company system has a number of application programs; each of them is defined to manipulate data files. These application programs have been written on request of the users in the organization. New application will be added to the system as the need arises. The system just described is called the file-based system.

Consider a traditional banking system which uses the file-based system to manage the organization's data in the picture below. As we can see, there are different departments in the Bank, each of them has their own applications that manage and manipulate different data files. For banking systems, the programs can be the ones to debit or credit an account , find the balance of an account, add a new mortgage loan or generate monthly statements, etc.



http://cnx.org/content/m28150/latest/

File-based approach for banking system

Keeping organizational information in this approach has a number of disadvantages, including:

Data Redundancy

Since files and applications are created by different programmers of various departments over long periods of time, it might lead to several problems such as

- inconsistency in data format,
- the same information may be kept in several different place (files)
- data inconsistency, which means various copies of the same data are conflicting ; this wastes storage space and duplicates effort

Data Isolation

- It is difficult for new applications to retrieve the appropriate data, which might be stored in various files.

Integrity problems

- Data values must satisfy certain consistency constraints that are specified in the application programs.
- It is difficult to make changes to the programs to enforce new constraints.

Security problems

- There are constraints regarding accessing privileges.
- Application requirements are added to the system in an ad-hoc manner so it is difficult to enforce constraints.

Concurrency – access anomalies

- Data may be accessed by many applications that have not been coordinated previously so it is not easy to provide a strategy to support multiple users to update data simultaneously.

These difficulties have prompted the development of a new approach in managing large amount of organizational information – the database approach.

Database Approach

Databases and database technology play an important role in most of areas where computers are used, including business, education, medicine, etc. To understand the fundamentals of database systems, we start by introducing the basic concepts in this area.

Role of databases in Business

Everybody uses a database in some way, maybe just to store information about their friends and family. That data might be written down or stored in a computer using a word processing program, or it could even be saved in a spreadsheet. However, the best place for it would be the Database Management Software (DBMS). The DBMS is a power software tool that allows you to store, manipulate and retrieve data in a variety of different ways.

Most companies keep track of customer information by storing it in a database. That data may be customers, employees, products, orders, or anything that may assist the business in their operations.

Meaning of Data

Data is factual information such as measurements or statistics about objects and concepts. We use it for discussion, or calculation. It could be a person, a place, an event, an action, any one of a number of things. A single fact is an element of data, or a data element.

If data is information and information is what we are all in the business of working with, you can start to see where you may be storing it:

- Filing cabinets
- Spreadsheets
- Folders
- Ledgers
- Lists
- Piles of papers on your desk

They all store information, and so too does a database. Because of the mechanical nature of databases, they have terrific power to manage and process the information they hold. This can make the information they house much more useful for your work.

With this understanding of data, we can start to see how a tool with the capacity to store a collection of data and organize it especially for rapid search, retrieval and processing might make a difference to how we can use data. This is all about managing information.

Source: http: //cnx.org/content/m28150/latest/

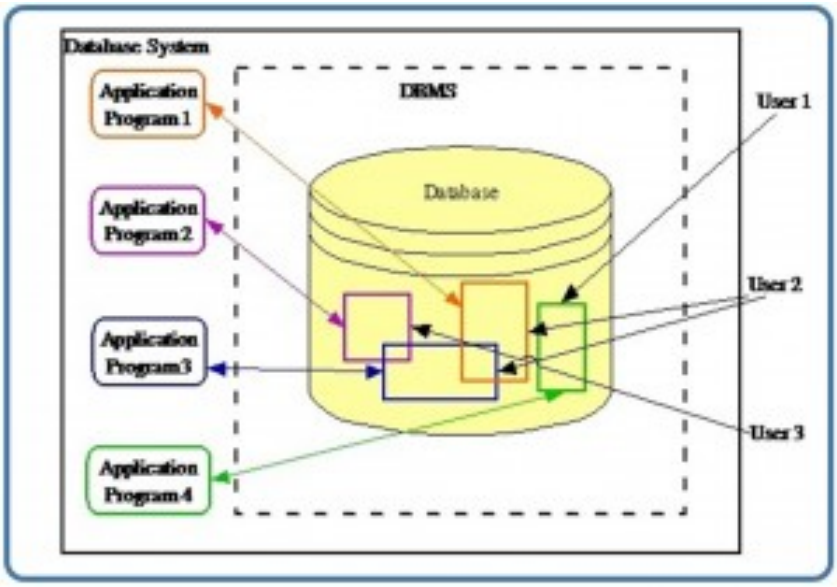
2

Chapter 2 Fundamental Concepts

A Database is a shared collection of related data which is used to support the activities of a particular organization. A database can be viewed as a repository of data that is defined once and then is accessed by various users. A database has the following properties:

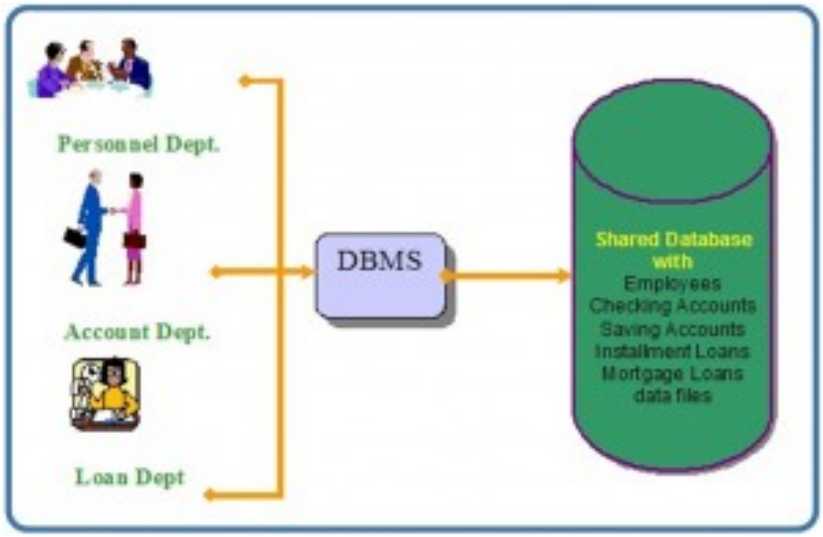
- It is a representation of some aspect of the real world; or perhaps, a collection of data elements (facts) representing real-world information.
- A database is logical, coherent and internally consistent.
- A database is designed, built, and populated with data for a specific purpose.
- Each data item is stored in a field,
- A combination of fields makes up a table. For example: an employee table contains fields that contain data about an individual employee

A Database Management System (DBMS) is a collection of programs that enable users to create, maintain databases and control all the access to the databases. The primary goal of the DBMS is to provide an environment that is both convenient and efficient for users to retrieve and store information.



https://cnx.org/content/m28150/latest/

With the database approach, we can have the traditional banking system as shown in the following image.



https://cnx.org/content/m28150/latest/

3

Chapter 3 Characteristics and Benefits of a Database

There are a number of characteristics that distinguish the database approach with the file-based approach.

Self-Describing Nature of a Database System

A Database System contains not only the database itself but also the descriptions of data structure and constraints (meta-data). This information is used by the DBMS software or database users if needed. This separation makes a database system totally different from the traditional file-based system in which the data definition is a part of application programs.

Insulation between Program and Data

In the file based system, the structure of the data files is defined in the application programs so if a user wants to change the structure of a file, all the programs that access that file might need to be changed as well. On the other hand, in the database approach, the data structure is stored in the system catalog not in the programs. Therefore, one change is all that's needed.

Support multiple views of data

A view is a subset of the database which is defined and dedicated for particular users of the system. Multiple users in the system might have different views of the system. Each view might contain only the data of interest to a user or a group of users.

Sharing of data and Multiuser system

A multiuser database system must allow multiple users access to the database at the same time. As a result, the multiuser DBMS must have concurrency control strategies to ensure several users access to the same data item at the same time, and to do so in a manner that the data will always be correct – data integrity.

Control Data Redundancy

In the Database approach, ideally each data item is stored in only one place in the database. In some cases redundancy still exists so as to improve system performance, but such redundancy is controlled and kept to minimum.

Data Sharing

The integration of the whole data in an organization leads to the ability to produce more information from a given amount of data.

Enforcing Integrity Constraints

DBMSs should provide capabilities to define and enforce certain constraints such as data type, data uniqueness, etc.

Restricting Unauthorised Access

Not all users of the system have the same accessing privileges. DBMSs should provide a security sub-system to create and control the user accounts.

Data Independence

System data (Meta Data) descriptions are separated from the application programs. Changes to the data structure is handled by the DBMS and not embedded in the program.

Transaction Processing

The DBMS must include concurrency control subsystems to ensure that several users trying to update the same data do so in a controlled manner. The results of any updates to the database must maintain consistency and validity.

Providing multiple views of data

A view may be a subset of the database. Various users may have different views of the database itself. Users may not need to be aware of how and where the data they refer to is stored.

Providing backup and recovery facilities

If the computer system fails in the middle of a complex update process, the recovery subsystem is responsible for making sure that the database is restored to the stage it was in before the process started executing.

Managing information

Managing information means taking care of it so that it works for us, and is useful for the work we are doing. The information we collect is no longer subject to "accidental disorganization" and becomes more easily accessible and integrated with the rest of our work. Managing information using a database allows us to become strategic users of the data we have.

We often need to access and re-sort data for various uses. These may include:

- Creating mailing lists
- Writing management reports
- Generating lists of selected news stories
- Identifying various client needs

The processing power of a database allows it to manipulate the data it houses, so it can

- Sort
- Match
- Link
- Aggregate
- Skip fields
- Calculate
- Arrange

So a database tracks data, finding the bits you need and processing them in the way you arrange for them to be processed.

Because of the versatility of databases, we find them powering all sorts of projects. A database can be linked to:

- A web site that is capturing registered users
- A client tracking application for social service organisations
- A medical record system for a health care facility
- Your personal address book in your e-mail client
- A collection of word processed documents
- A system that issues airline reservations

Source: http://cnx.org/content/m28150/latest/

4

Chapter 4 Types of Database Models

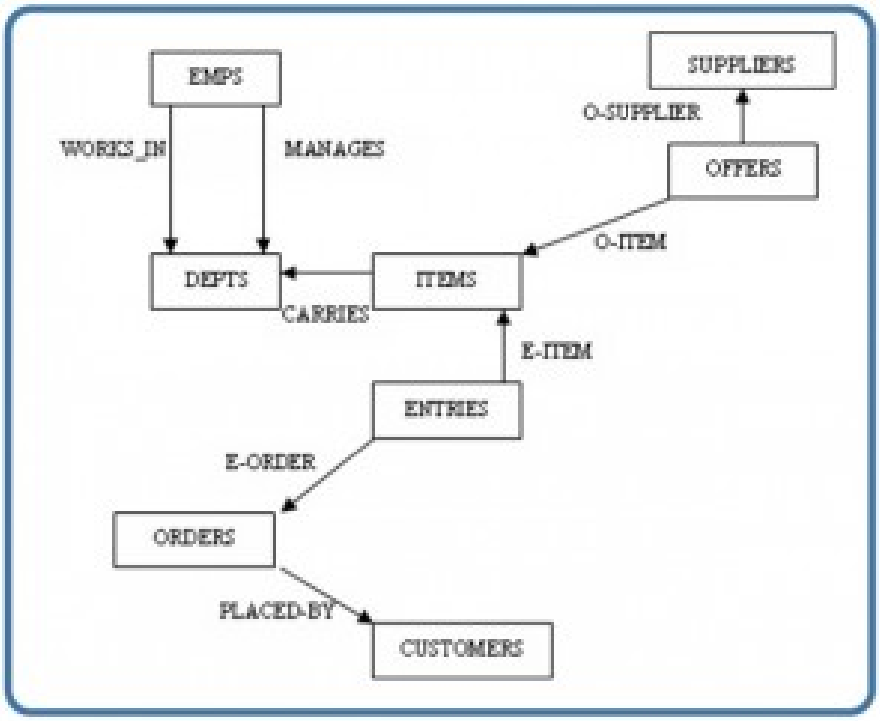
High-level Conceptual Data models

Provide concepts that are close to the way people perceive data to present the data. A typical example of this type is the entity relationship model which uses main concepts like entities, attributes, relationships. An entity represents a real-world object such as an employee, a project. An entity has some attributes which represents properties of entity such as employee's name, address, birthdate. A relationship represents an association among entities, for example an employee works on many projects. The relationship exists between employee and project.

Record-based Logical Data models

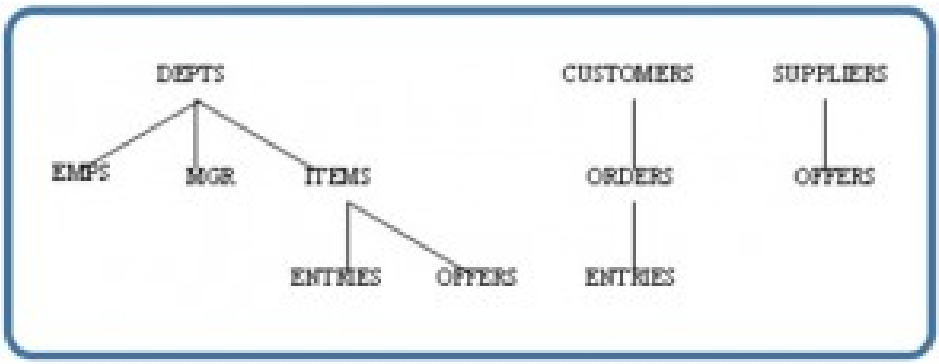
Provide concepts that can be understood by the user but not too far from the way data is stored in the computer. Three well-known data models of this type are relational data model, network data model and hierarchical data model.

- The Relational model represents data as relations or tables.
- The Network model represents data as record types and also represents a limited type of one to many relationship, called set type.



http://cnx.org/content/m28150/latest/

The Hierarchical model represents data as a hierarchical tree structures. Each hierarchy represents a number of related records. Here is the schema in hierarchical model notation.



http://cnx.org/content/m28150/latest/

5

Chapter 5 Data Modelling

Data Modelling is the first step in the process of database design. This step is sometimes considered as a high-level and abstract design phase (conceptual design). The aims of this phase is to:

- Describe what data is contained in the database (e.g. entities: students, lecturers, courses, subjects etc.)
- Describe the relationships between data items (e.g. Students are supervised by Lecturers; Lecturers teach Courses)
- Describe the constraints on data (e.g. Student Number has exactly 8 digits; a subject has 4 or 6 unit of credits only)

The data items, the relationships and constraints are all expressed using the concepts provided by the high-level data model. Because these concepts do not include the implementation details, the results of the data modelling process is a (semi) formal representation of the database structure. This result is quite easy to understand so it is used as reference to make sure that all the user's requirements are met.

The third step is Database Design. During this step, we might have two sub-steps called Database Logical Design which define a database in a data model of a specific DBMS and Database Physical Design which define the internal database storage structure, file organization or indexing techniques. The last two steps shown are Database Implementation and Operations/Interfaces Building. These focus on creating an instance of the schema and implementing operations and user interfaces.

In the database design phases, data is represented using a certain data model. The Data Model is a collection of conceptual concepts or notations for describing data , data relationships, data semantics and data constraints. Most data models also include a set of basic operations for manipulating data in the database.

Degree of Data Abstraction

In this section we will look at the database design process in terms of specificity. Just as any design starts at a high level and proceeds to an ever-increasing level of detail so does database design. For example, when building a home, you start with how many bedrooms the home will have, how many bathrooms, whether the home will be on one level or multiple levels, etc. The next step is to get an architect to design the home from a more structured perspective. This level gets more detailed with respect to actual room sizes, how the home will be wired, where the plumbing fixtures will be placed, etc. The last step is to hire a contractor to build the home. That's looking at the design from a high level of abstraction to an increasingly detailed level of abstraction.

The database design is very much like that. It starts with users identifying the business rules, database designers and analysts creating the database design, and then the database design is physically created using a DBMS.

External models

- Is the user's view of the database
- contains multiple different external views
- is closely related to the real world as perceived by each user

Conceptual model

- These models provide a flexible data structuring capabilities.
- Community view : the logical structure of the entire database
- Contains 'What' data is stored
- Shows relationships among data
  - constraints
  - semantic information (business rules – discussed later)
  - security & integrity information
  - A database is considered as a collection of entities (objects) of various kinds.
- Basis for identification and high-level description of main data objects, avoids details
- They are database independent. It does not matter what database you will be using.

Internal model

- Database is considered as a collection of fixed – size record.
- This model is closer to the physical level or file structure.
- Is a representation of database as seen by the DBMS.
- Requires the designer to match the conceptual model's characteristics and constraints to those of the selected implementation model.
- The entities in the conceptual model are mapped to the tables in the relational model.
- The three most well-known models of this kind are the relational data model, the network data model and the hierarchical data model.

Physical model

- Physical representation of the database
- Lowest level of abstractions
- It is how the data is stored, dealing with
  - Run-time performance
  - Storage utilization & compression
  - File organization & access methods
  - Data encryption
- Physical level – managed by OS
- Provide concepts that describe the details of how data is stored in the computer's memory

Data Abstraction Layer

In a pictorial view, you can see how the different models work together. Let's look at this from the highest level; the External Model.

The External Model is the end user view of the data. Typically a database is an enterprise system that serves the needs of multiple departments. However, one department is not interested in seeing the other departments' data. E.g. Human Resource does care to view the Sales data. Therefore, one user view will differ from another.

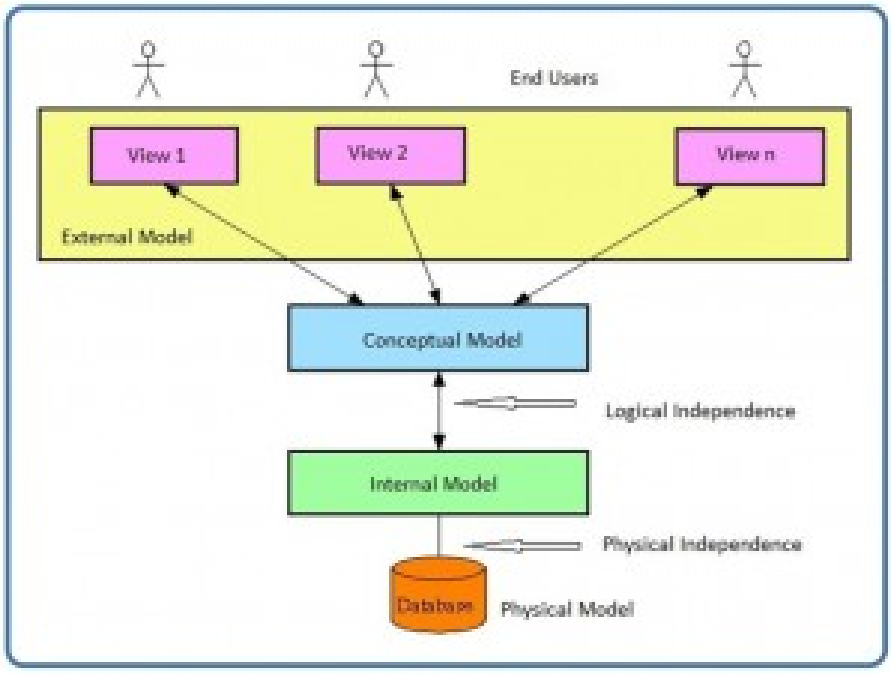
The External model requires that the designer subdivide a set of requirements and constraints into functional modules that can be examined within the framework of their external models (i.e. HR vs. Sales)

As a data designer you need to understand all the data so that you can build an enterprise wide database. Based on the needs of various departments the conceptual model is the first model created.

At this stage the conceptual model is independent of both software and hardware. It does not depend on the DBMS software used to implement the model. It does not depend on the hardware used in the implementation of the model. Changes in either hardware or DBMS software have no effect on the database design at the conceptual level.

Once a DBMS is selected, you can then implement it. This is the Internal model. Here you create all the tables, constraints, keys, rules, etc. This is often referred to as the Logical design.

The physical model is simply the way the data is stored on disk. Each database Vendor will have their own way of storing the data.



Schemas

A schema is an overall description of a database and it is usually represented by the Entity Relationship diagram. There are multiple subschemas which are representations of the external models. To put it into perspective of one database:

- External Schemas : multiple

- multiple subschemas = multiple external views of the data
- Conceptual Schema : one only
  - Data items, relationships & constraints – ER diagram
- Physical Schema : one only
- Record definitions, representation methods
- Access methods, indexes, hashing

Logical and Physical Data Independence

Data independence refers to the immunity of user applications to changes made in the definition and organization of data. Data abstractions expose only those items that are important or pertinent to the user. Complexity is hidden from the database user.

Physical data independence deals with hiding the details of the storage structure from user applications. The application should not be involved with these issues, since there is no difference in the operation carried out against the data.

The data independence and operation independence together gives the feature of data abstraction. There are two types of data independence: Logical and Physical.

Source: http://cnx.org/content/m28150/latest/

Logical data independence

The ability to change the logical (conceptual) schema without changing the External schema (User View) is called logical data independence. For example, the addition or removal of new entities, attributes, or relationships to the conceptual schema should be possible without having to change existing external schemas or having to rewrite existing application programs.

In other words, if the logical schema changed, i.e. changes to the structure of the database like adding a columns or other tables, should not affect the functioning of the application (external views).

Source: http://cnx.org/content/m28150/latest/

Physical data independence

This refers to the immunity of the internal model to changes in the physical model. The logical schema stays unchanged even though changes are made to file organization or storage structures, storage devices, or indexing strategy.

Source: http://cnx.org/content/m28150/latest/

6

Chapter 6 Classification of Database Systems

The database management systems can be classified based on several criteria.

Based on data model

The most popular data model in use today is the relational data model. Well known DBMSs like Oracle, MS SQL Server, DB2, MySQL support this model. Other traditional models can be named hierarchical data model, or network data model. In the recent years, we are getting familiar with object-oriented data models but these models have not had widespread use. Some examples of Object-oriented DBMSs are O2, ObjectStore or Jasmine.

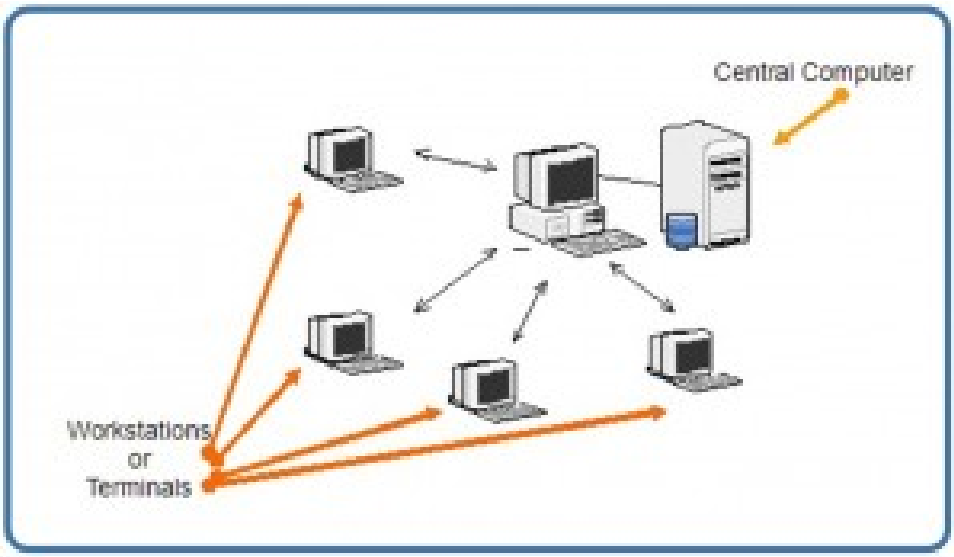
Based on the number of users

We can have a single user database system which supports one user at a time or multiuser systems which support multiple users concurrently.

Based on the ways database is distributed

Centralized Systems

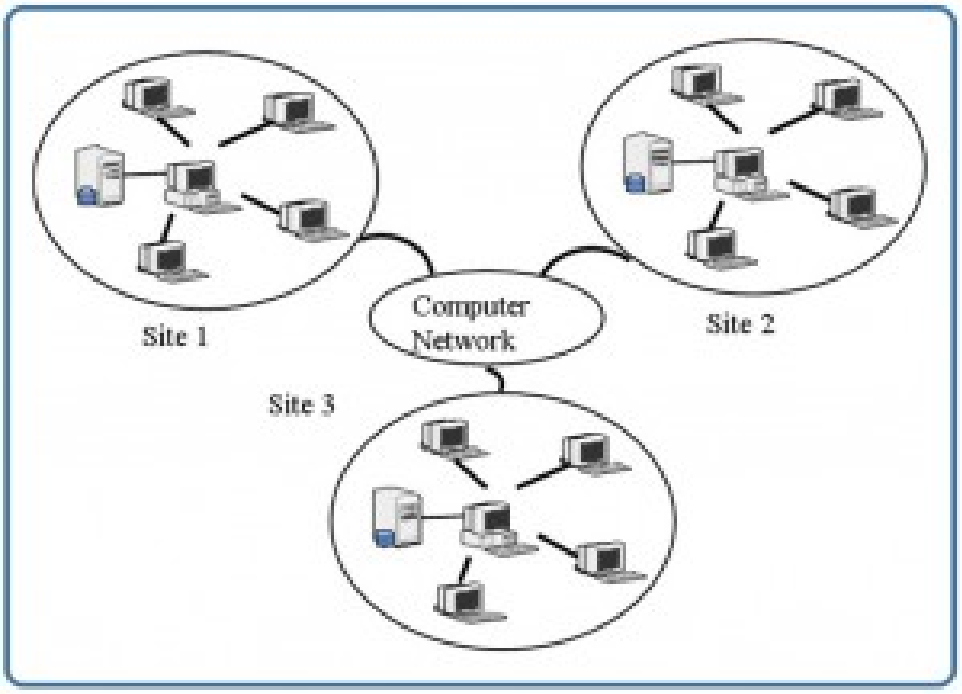
With centralized database systems, the system is stored at a single site.



Source: http://cnx.org/content/m28150/latest/

Distributed database system

Actual database and DBMS software are distributed in various sites connected by a computer network.



Source: http://cnx.org/content/m28150/latest/

Homogeneous distributed Database Systems

Use the same DBMS software at multiple sites. Data is exchanged between various sites and can be handled easily.

Heterogeneous distributed Database Systems

Different sites might use different DBMS software. There is additional software to support data exchange between sites.

Source: http://cnx.org/content/m28150/latest/

7

Chapter 7 The Relational Data Model

This data model is introduced by C.F.Codd in 1970. Currently, it is considered as the most widely used data model.

The relational model has provided the basis for:

- Research on theory of data/relationship/constraint
- Numerous database design methodologies
- The standard database access language SQL
- Almost all modern commercial database management systems

The relational data model describes the world as "a collection of inter-related relations (or tables) "

Fundamental concepts in Relational Data Model

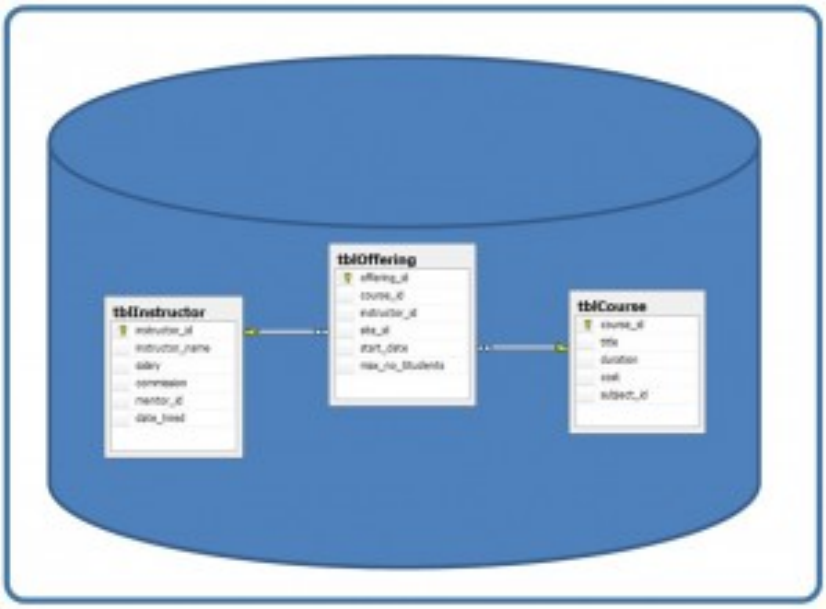
Relation

A relation is a subset of the Cartesian product of a list of domains characterized by a name. Given n domains denoted by D1, D2, …, Dn , r is a relation defined on these domains if r?D1×D2×…×Dn. A relation can be viewed as a "table". In that table, each row represents a tuple of data values and each column represents an attribute.

Source: http://cnx.org/content/m28250/latest/

Table

To put it simply, a table holds the data. A database is composed of multiple tables. The one below shows a database with three tables.



Column

A database stores pieces of information or facts in an organised way. Understanding how to use and get the most out of databases requires us to understand that method of organisation.

The principal storage units are called **columns**, or **fields**, or **attributes**. These will house the basic components of data that your content can be broken down into. When deciding which fields to create you need to think generically about your information. For example, drawing out the *common* components of the information that you will store in the database, and avoiding the specifics that distinguish one item from another.

Look at the following example of an ID card to see the relationship between fields and their data:

Field Name	Data
First Name	Isabelle
Family Name	Whelan
Nationality	British
Salary	109,900
Date of Birth	15 September 1983
Marital Status	Single
Shift	Mon, Wed
Place of issue	Addis Ababa
Valid until	17 December 2003

Domain

A domain is the original sets of atomic values used to model data. By atomic, we mean that each value in the domain is indivisible as far as the relational model is concerned. For example:

- The domain of Marital status has a set of possibilities: Married, Single, Divorced
- The domain of day Shift has the set of all possible days : {Mon, Tue, Wed…}.

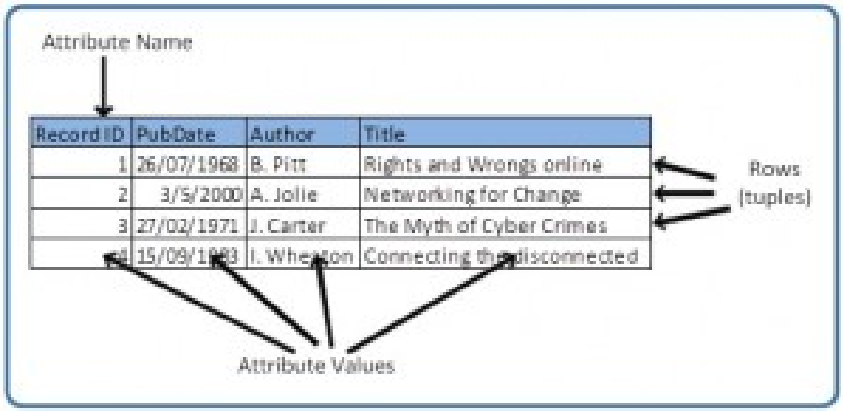
- The domain of Salary is the set of all floating-point numbers greater than 0 and less than 200,000.
- The domain of First Name is the set of character strings that represents names of people.

In summary, a Domain is a set of acceptable values that a column is allowed to contain. This is based on various properties and the data type for the column. We will discuss data types in another chapter.

Records

Just as the content of any one document or item needs to be broken down into its constituent bits of data for storage in the fields, the link between them also needs to be available so that they can be re-constituted into their whole form. Records allow us to do this. A tuple is another term used for records.

A simple table gives us the clearest picture of how records and fields work together in the database storage project. Records and fields form the basis of all databases.



The example above shows us how fields can hold a range of different sorts of data – this one has:

- An ID field: ordinal number. integer
- A Pubdate field: day/month/year. date
- An author field: Initial. Surname. text
- A title field: text.
- A body text field: text.

By creating fields especially for the sorts of data they will house, the database can sort them intelligently. You can request a selection of records which are limited by their date – all before a given date, or all after a given date or all between 2 given dates. Similarly you can choose to have records sorted by date. What is happening here is that the database is reading the information in the date field, not just as numbers separated by slashes, but rather as dates which must be ordered according to a calendar system – because the field was set up as a date field. You are commanding the database to sift through its data and organise it in a particular way.

Degree

The degree is the number of attributes. In our example above, the degree is 4.

Properties of Tables

- Table has a name that is distinct from all other tables in the database.
- There are no duplicate rows, each row is distinct.
- Entries in columns are atomic (no repeating groups or multivalued attributes)
- Entries from columns are from the same domain based on their data type:

number (numeric, integer, float, smallint,…)
character (string)
date
logical (true or false)

- Operations combining different data types are disallowed
- Each attribute has a distinct name
- sequence of columns is insignificant
- sequence of rows is insignificant

Terminology

We have used many different terms. Here is a summary. Alternative 1 is the most common.

Formal Terms (Codd)	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

8

Chapter 8 Entity Relationship Model

The Entity Relationship Data Model (ER) has existed for over 35 years.

The ER model is well-suited to data modelling for use with databases because it is fairly abstract and is easy to discuss and explain. ER models are readily translated to relations.

ER modelling is based on two concepts:

- Entities, that is, things. E.g. Prof. Ba, Course Database System.
- Relationships, that is, associations or interactions between entities.

E.g. Prof. Ba teaches course Database Systems

ER models (or ER schemas) are represented by ER diagrams.

The example database

For the next section we will look at a sample database called the COMPANY database to illustrate the concepts of Entity Relationship Model.

This database contains the information about employees, departments and projects:

- There are several departments in the company. Each department has a unique identification, a name, location of the office and a particular employee who manages the department.
- A department controls a number of projects, each of which has unique name, a unique number and the budget.
- Each employee has name, an identification number, address, salary, birthdate. An employee is assigned to one department but can join in several projects. We need to record the start date of the employee in each project. We also need to know the direct supervisor of each employee.
- We want to keep track of the dependents of the employees. Each dependent has name, birthdate and relationship with the employee.

Entity, Entity Set and Entity Type

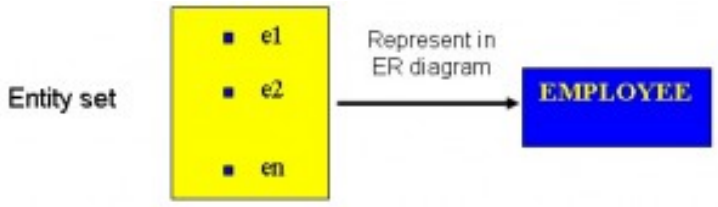
An entity is an object in the real world with an independent existence and can be differentiated from other objects

An entity might be

- An object with physical existence. E.g. a lecturer, a student, a car
- An object with conceptual existence. E.g. a course, a job, a position

An Entity Type defines a collection of similar entities.

An Entity Set is a collection of entities of an entity type at a point of time. In ER diagrams, an entity type is represented by a name in a box.



Source: http://cnx.org/content/m28250/latest/

Types of Entities

Independent entities, also referred to as Kernels, are the backbone of the database. It is what other tables are based on. Kernels have the following characteristics:

- they are the 'building blocks' of a database
- the primary key may be simple or composite
- the primary key is not a foreign key
- they do not depend on another entity for their existence

Example: Customer table, Employee table, Product table
Dependent Entities, also referred to as derived, depend on other tables for their meaning.

- Dependent entities are used to connect two kernels together.
- They are said to be existent dependent on two or more tables.
- Many to many relationships become associative tables with at least two foreign keys.
- They may contain other attributes.
- The foreign key identifies each associated table.
- There are three options for the primary key:
  - use a composite of foreign keys of associated tables if unique
  - use a composite of foreign keys and qualifying column
  - create a new simple primary key

Characteristic entities provide more information about another table. These entities have the following characteristic.

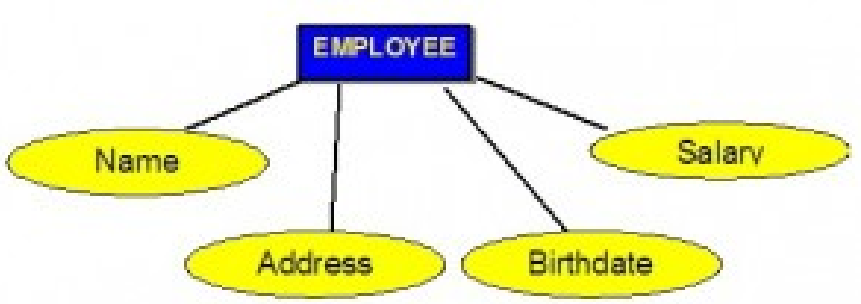
- They represent multi-valued attributes.
- They describe other entities.
- They typically have a one to many relationship.
- The foreign key is used to further identify the characterized table.
- Options for primary key are as follows:
  - foreign key plus a qualifying column
  - or create a new simple primary key
    - Employee(<u>EID,</u> Name, Address, Age, Salary)
    - EmployeePhone(<u>EID, Phone</u>)

Attributes

Each entity is described by a set of attributes. E.g. Employee = (Name, Address, Age, Salary).

Each attribute has a name, associated with an entity and is associated with a domain of legal values. However the information about attribute domain is not presented on the ER diagram.

In the diagram, each attribute is represented by an oval with a name inside.



Source: http://cnx.org/content/m28250/latest/

Types of Attributes

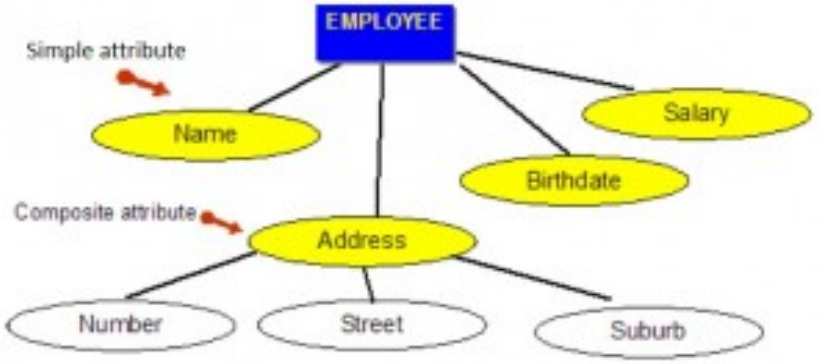
There are a few types of attributes you need to be familiar with. Some of these are to be left as is, but some need to be adjusted to facilitate representation in the relational model. This first section will discuss the types of attributes. Later on we will discuss fixing the attributes to fit correctly into the relational model.

Simple attributes are attributes that are drawn from the atomic value domains

e.g. Name = {John} ; Age = {23} , also called Single valued.

Composite attributes: Attributes that consist of a hierarchy of attributes

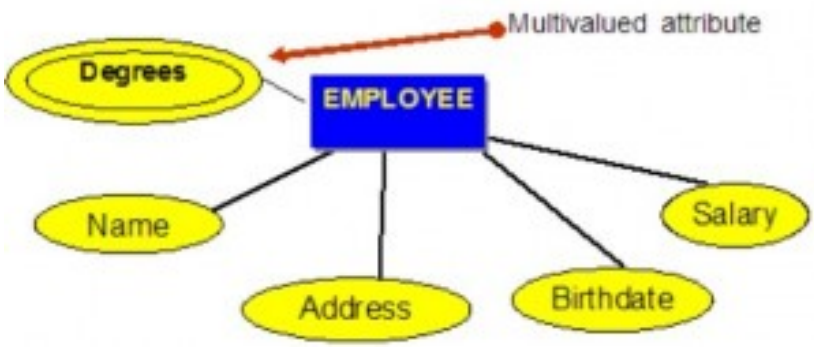
e.g. Address may consists of "Number", "Street" and "Suburb" → Address = {59 + 'Meek Street' + 'Kingsford'}



Source: http://cnx.org/content/m28250/latest/

Multivalued attributes: Attributes that have a set of values for each entity

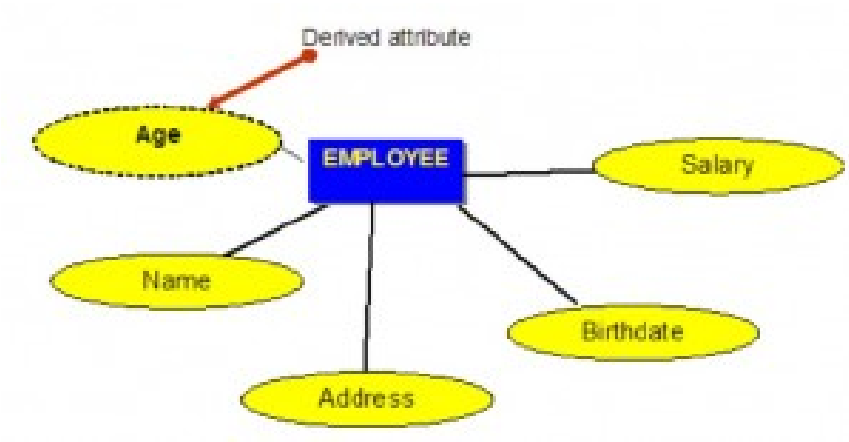
e.g. Degrees of a person: ' BSc' , 'MIT', 'PhD'



Source: http://cnx.org/content/m28250/latest/

Derived attributes: Attributes contain values that are calculated from other attributes

e.g. Age can be derived from attribute DateOfBirth. In this situation, DateOfBirth might be called Stored Attribute.



Source: http://cnx.org/content/m28250/latest/

Source: http://cnx.org/content/m28250/latest/

Keys

An important constraint on the entities is the key. The key is an attribute or a group of attributes whose values can be used to uniquely identify an individual entity in an entity set.

Candidate key

- a simple or composite key that is unique and minimal
- unique – no two rows in a table may have the same value at any time
- minimal – every column must be necessary for uniqueness
- For example, for the entity

  Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

  Possible candidate keys are EID, SIN

FirstName and Last Name – assuming there is no one else in the company with the same name, Last Name and DepartmentID – assuming two people with the same last name don't work in the same department.

EID, SIN are also candidate keys

Composite key

- Composed of more than one attribute
- For example, as discussed earlier

First Name and Last Name – assuming there is no one else in the company with the same name, Last Name and Department ID – assuming two people with the same last name don't work in the same department.

- A composite key can have two or more attributes, but it must be minimal.

Primary key

- A candidate key is selected by the designer to uniquely identify tuples in a table. It must not be null.
- A key is chosen by the database designer to be used as an identifying mechanism for the whole entity set. This is referred to as the primary key. This key is indicated by underlining the attribute in the ER model.
- For example **Employee**(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID) - EID is the Primary key.

Secondary key

- an attribute used strictly for retrieval purposes (can be composite), for example: Phone number, Last Name and Phone number, etc.
- all other candidate keys not chosen as the primary key
- An attribute in one table that references the primary key of another table OR it can be null.
- Both foreign and primary keys must be of the same data type
- For example: **Employee**(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID) - DepartmentID is the Foreign key.

Alternate key

- all other candidate keys not chosen as the primary key

Foreign key

- An attribute in one table that references the primary key of another table OR it can be null.
- Both foreign and primary keys must be of the same data type
- For example: **Employee**(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID) - DepartmentID is the Foreign key.

Nulls

This is a special symbol, independent of data type, which means either unknown or inapplicable. (*it does not mean zero or blank*)

- No data entry
- Not permitted in primary key
- Should be avoided in other attributes
- Can represent
  - An unknown attribute value
  - A known, but missing, attribute value
  - A "not applicable" condition
  - Can create problems when functions such as COUNT, AVERAGE, and SUM are used
  - Can create logical problems when relational tables are linked

NOTE: the result of a comparison operation is null when either argument is null. The result of an arithmetic operation is null when either argument is null (except functions which ignore nulls)

Salary_tbl

emp#	jobName	salary	commission
E10	Sales	12500	32090
E11	Null	25000	8000
E12	Sales	44000	0
E13	Sales	44000	Null

PROBLEM: Find all employees (EMP#) in Sales whose salary plus commission is greater than 30,000.

```
SELECT emp# FROM salary_tbl
WHERE jobname = 'Sales' AND
(commission + salary) > 30000                    –> E10 and E12
```

This result does not include E13 because of the Null value in Commission. To ensure the row with the null value is included, we need to look at the individual fields. By adding commission and Salary for employee E13, the result will be a null value. The solution is shown below.

```
SELECT emp# FROM salary_tbl
WHERE jobName = 'Sales' AND
(commission > 30000 OR salary > 30000 OR
(commission + salary) > 30000 )                  –>E10 and E12 and E13
```

Entity Types

Existence Dependency

- An entity's existence is dependent on the existence of the related entity.
- It's existence-dependent if it has a mandatory foreign key. That is a foreign key attribute that cannot be null.

- Spouse entity is existence dependent on the employee

Weak Entities

- These tables are existence dependent.
- They cannot exist without entity with which it has a relationship.
- Primary key is derived from the primary key of the parent entity
  - The spouse table is a weak entity because its PK is dependent on the employee table. Without a corresponding employee record, the spouse record could not exist

Strong Entities

- If an entity can exist apart from all of its related entities it's considered a strong entity.
- Kernels are strong entities.
- A table without a foreign key is or a table that contains a foreign key which can contain NULLS is a strong entity.

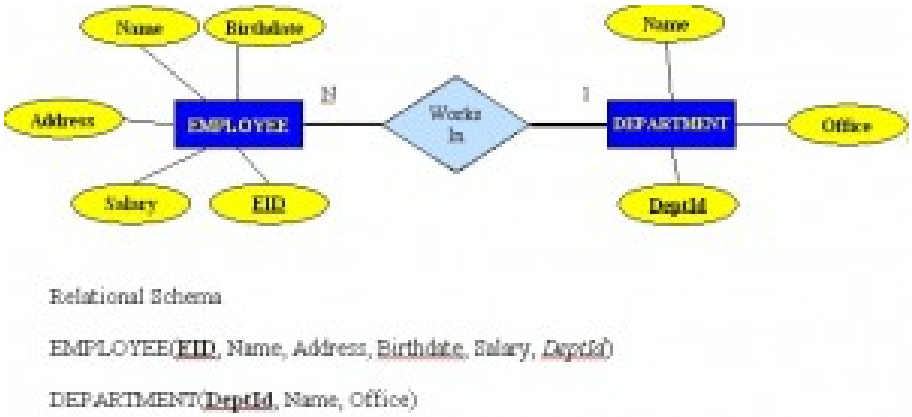
Relationships

These are the glue that holds the tables together. They are used to connect together related information in other tables.

1:M relationship

- Should be the norm in any relational database design
- Relational database norm
- Found in any database environment

e.g. One department has many employees



Source: http://cnx.org/content/m28250/latest/

1:1 relationship

- Should be rare in any relational database design
- One entity can be related to only one other entity, and vice versa
- Could indicate that two entities actually belong in the same table

e.g. A professor chairs one department and a department has only one chair.

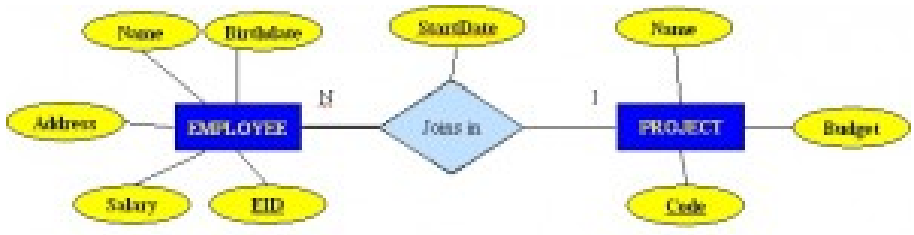
1:1 Example:

- An employee is associated with one spouse, and one spouse is associated with one employee.
- Cannot be implemented as such in the relational model
- M:N relationships can be changed into two 1:M relationships
- Can be implemented by breaking it up to produce a set of 1:M relationships
- Can avoid problems inherent to M:N relationship by creating a composite entity or bridge entity
- A student has many classes and a class can hold many students.
- Implementation of a composite entity
- Yields required M:N to 1:M conversion
- Composite entity table must contain at least the primary keys of original tables
- Linking table contains multiple occurrences of the foreign key values
- Additional attributes may be assigned as needed

M:N relationships

Mapping M-N Binary Relationship Type: For each M-N Binary Relationship, identify two relations A, B represent two entity type participating in R. Create a new relation S to represent R. Include in S as foreign keys the primary keys of A and B and all the simple attributes of R. The combination of primary keys of A and B will make the primary key of S.

M:N relationships



Relational Schema

EMPLOYEE (EID, Name, Address, Birthdate, Salary)

PROJECT (Code, Name, Budget)

JOIN(EID, Code, StartDate)

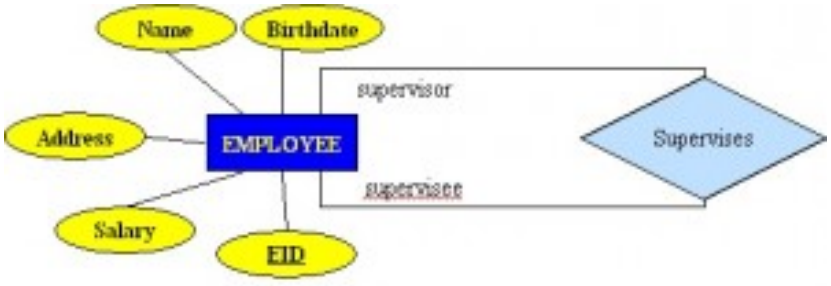
Source: http://cnx.org/content/m28250/latest/

Unary Relationships (recursive)

One in which a relationship exists between occurrences of the same entity set

The two keys (primary key and foreign key) are the same but they represents two entities of different roles relate to this relationship.

In some entities, a separate column can be created that refers to the primary key of the same entity set.



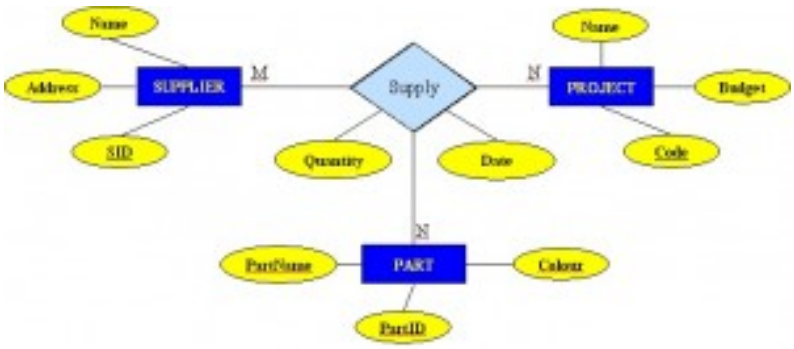
Relational Schema

EMPLOYEE (EID, Name, Address, Birthdate, Salary, *Super-EID*)

Source: http://cnx.org/content/m28250/latest/

Ternary Relationships

Mapping Ternary Relationship Type: For each n-ary (> 2) Relationships create a new relation to represent the relationship. The primary key of the new relation is the combination of the primary keys of the participating entities that hold the N (many) side. In most cases of an n-ary relationship all the participating entities hold a many side. Source: http://cnx.org/content/m28250/latest/



Relational Schema

SUPPLIER (SID, Name, Address)
PROJECT(Code, Name, Budget)
PART(PartID, PartName, Colour)
Supply(SID, Code, PartID, Quantity, Date)

Source: http://cnx.org/content/m28250/latest/

Relationship Strength

Relationship strength is based on how the primary key of a related entity is defined.

Weak Relationships (non-identifying relationship)

- Exists if the PK of the related entity does not contain a PK component of the parent entity.
- Customer(**custid**, custname)
- Order(**orderID**, custid, date)

Strong Relationship (identifying)

- Exists when the PK of the related entity contains the PK component of the parent entity.
- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode, Section**, ClassTime…)

9

Chapter 9 Integrity Rules and Constraints

Constraints are a very important feature in a relational model. In fact, the relational model supports well defined theory of constraints on attributes or tables. Constraints are useful because they allow a designer to specify the semantics of data in the database and constraints are the rules to enforce DBMSs to check that data satisfies the semantics.

Domain Integrity

Domain restricts the values of attributes in the relation and it is a constraint of the relational model. However, there are real –world semantics on data that cannot specified if used only with domain constraints. We need more specific ways to state what data values are/are not allowed and what format is suitable for an attributes. For example, the employee ID must be unique, the employee birthday is in the range [Jan 1, 1950, Jan 1, 2000]. Such information is provided in logical statements called integrity constraints.

There are several kinds of integrity constraints:

Entity Integrity – Every table requires a primary key. The primary key, nor any part of the primary key, can contain NULL values. This is because NULL values for the primary key means we cannot identify some rows. For example, in the EMPLOYEE table, Phone cannot be a key since some people may not have a phone.

Referential integrity – a foreign key must have a matching primary key or it must be null

This constraint is specified between two tables (parent and child); it maintains the correspondence between rows in these tables. It means the reference from a row in one table to other table must be valid. Examples of Referential integrity constraint:

Referential integrity Examples

In the Customer/Order database:

- Customer(**custid**, custname)
- Order(**orderID**, custid, OrderDate)

To ensure that there are no orphan records, we need to enforce referential integrity.

An orphan record is one whose foreign key value is not found in the corresponding entity – the entity where the PK is located. Recall that a typical join is between a PK and FK.

The referential integrity constraint states that the CustID in the Order table must match a valid CustiD in the Customer table. Most relational databases have declarative referential integrity. In other words, when the tables are created the referential integrity constraints are set up.

In the Course/Class database:

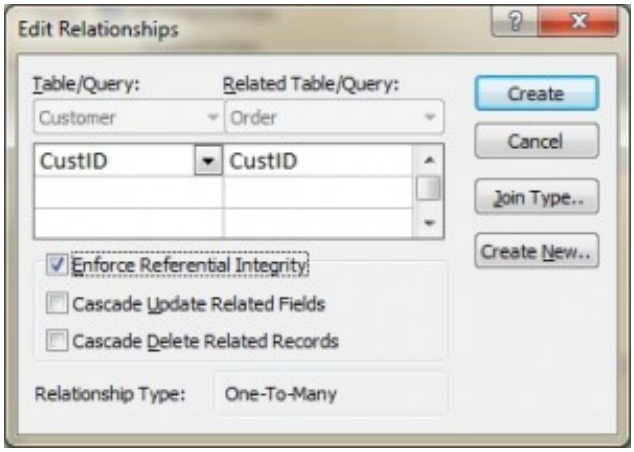
- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode, Section**, ClassTime)

The referential integrity constraint states that CrsCode in the Class table must match a valid CrsCode in the Course table. In this situation, it's not enough that the CrsCode and Section in the Class table make up the PK, we must also enforce referential integrity.

When setting up referential integrity it is important that the PK and FK have the same data types and come from the same domain. Otherwise the RDBMS will not allow the join.

Referential Integrity in MS Access

In MS Access referential integrity is set up by joining the PK in the Customer table to the CustID in the Order table.



Referential Integrity using Transact SQL (MS SQL Server)

```
CREATE TABLE Customer
      ( CustID INTEGER PRIMARY KEY,
CustName CHAR(35)
)
      CREATE TABLE Orders
      ( OrderID INTEGER PRIMARY KEY,
CustID INTEGER REFERENCES Customer(CustID),
OrderDate DATETIME
)
```

The referential integrity is set when creating the table (Orders) with the FK.

Foreign Key Rules

Additional foreign key rules may be added, such as what to do with the child rows (Orders table) when the record with the PK – the parent (Customer) is deleted or changed (updated). The relationship window in Access shows two additional options for foreign keys rules. Cascade Update and Cascade Delete. If they are not selected, the system would prevent the deletion or update of PK values in the parent table (Customer) if a child record exists. The child record is any record with a matching PK.

DELETE
- RESTRICT
- CASCADE
- SET TO NULL

UPDATE
- RESTRICT
- CASCADE

In some databases, an additional option exists when selecting the Delete option. That is 'Set to Null'. In these situations, the PK row is deleted, but the FK in the child table is set to Null. Though this creates an orphan row, it is acceptable.

Enterprise Constraints – sometimes referred to as Semantic constraints. They are additional rules specified by users or database administrators. i.e. A class can have a maximum of 30 students. A teacher can teach a maximum of 4 classes a semester. An employee cannot take a part in more than 5 projects. Salary of an employee cannot exceed the salary of the employee's manager.

Business Rules

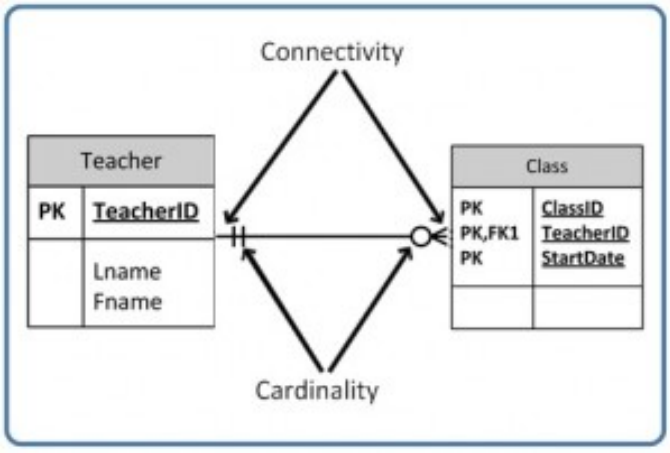
Another term we've used is semantics. Business rules are obtained from users when gathering requirements. The requirements gathering process is very important and should be verified by the user before the database design is built. If the business rules are incorrect, the design will be incorrect and ultimately the application built will not function as expected by the users.

Some examples of business rules are:

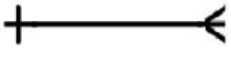
- A teacher can teach many students
- A class can have a maximum of 35 students
- A course can be taught many times, but by only one instructor
- Not all teachers teach classes, etc.

Cardinality

Expresses minimum and maximum number of entity occurrences associated with one occurrence of related entity. Business rules are used to determine cardinality.



Relationship Participation

There are two sides to each of the relationships when it comes to participation. Each side can have two options for participation. It is either 0 (zero), 1 (one), or many. The outer most symbol represents the connectivity. i.e. one to many would be represented by 

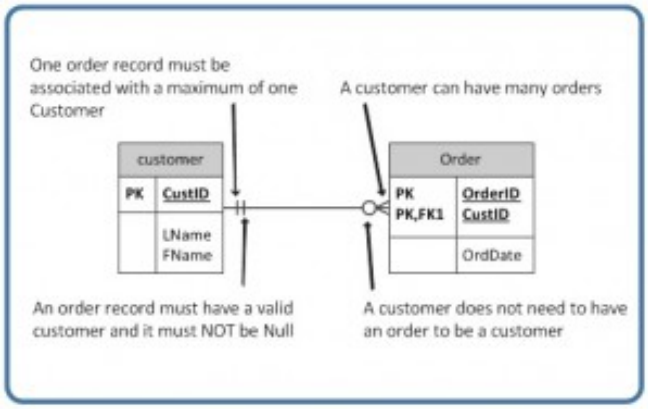
The inner most symbols represent the cardinality which indicates the minimum number of instances in the corresponding entity. i.e. on the right hand side it is read as: minimum 1 maximum many. On the left hand side it's read as minimum 1 and maximum 1. 

See below for more examples.

Optional

One entity occurrence does not require a corresponding entity occurrence in a relationship

 This shows a zero or many. The many side is optional.

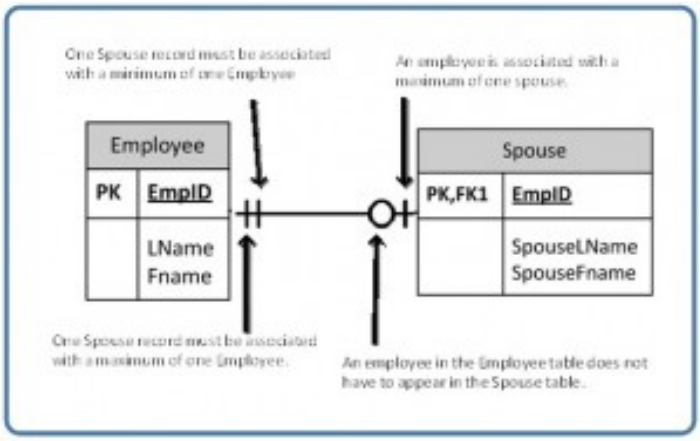


This can also be read as:

Right hand side: A customer can be in a minimum of 0 orders or a maximum of many orders.

Left hand side: The order entity must contain a minimum of one related entity in the customer table and a maximum of 1 related entity.

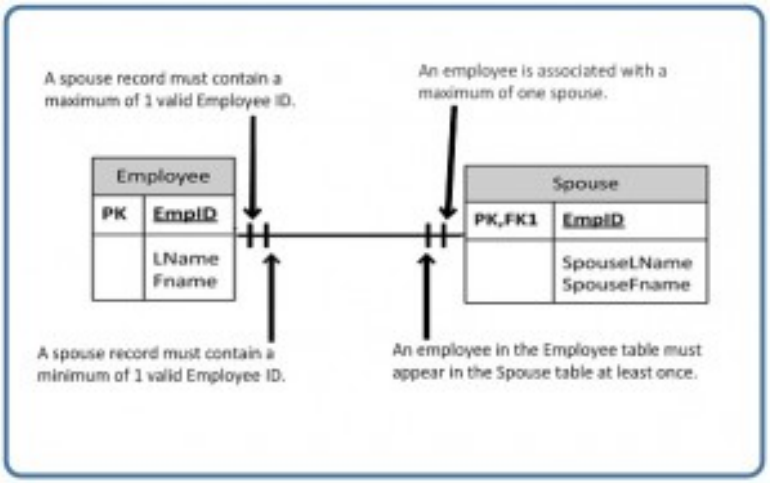
 This shows a zero or one. The 1 side is optional.



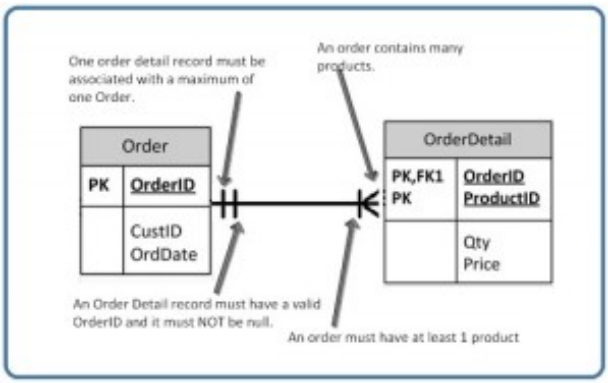
Mandatory

One entity occurrence requires a corresponding entity occurrence in a relationship

 This shows one and only one. 1 side is mandatory.



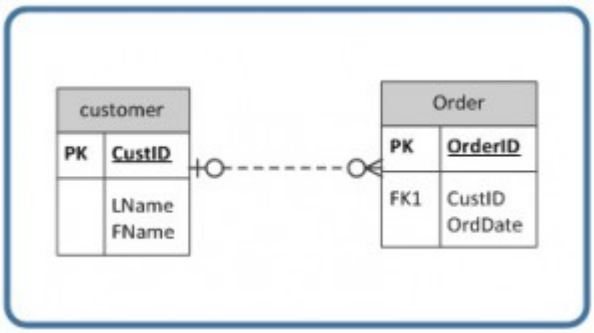
 This example shows one or many. The many side is mandatory.



So far we have seen that the right hand side can have  a 0 (zero) cardinality and a connectivity of many or one.

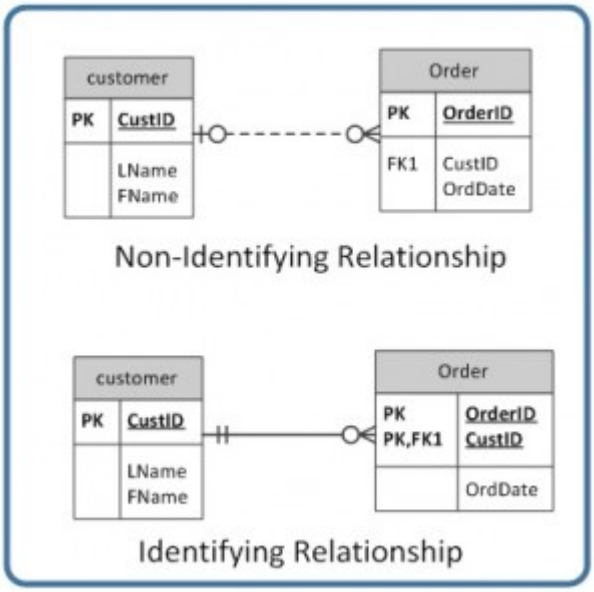
The left hand side can also have a cardinality  of 0 (zero).

However, it cannot have a connectivity of 0 only 1 (as shown). The connectivity symbols show maximums. So if you think about it logically, if the connectivity symbol on the left hand side shows 0, then there would be no connection between the tables. The way to read the left hand side is: The CustID in the Order table must be found in the Customer table a minimum of 0 and a maximum of 1 time. The 0 means that the CustID in the Order table may be null. The left most 1 (right before the 0 representing connectivity) says that if there is a CustID in the Order table it can only be in the Customer table once. When you see the 0 symbol for cardinality you can assume two things. The FK in the Order table allows nulls, and the FK is not part of the PK since PKs must not contain null values.



Relationship Types

You may have noticed in the previous image is the dashed line that connects the two tables. This is referred as the relationship type. It's either identifying or non-identifying. Please see the section that discussed weak and strong relationships for more explanation. The only thing you need to understand for this section is the way the relationship is depicted in the ERD. Identifying relationships will have a solid line (PK contains the FK). The non-Identifying relationship does not contain the FK in the PK.



10

Chapter 10 ER Modelling

One important theory developed for the relational model involves the notion of functional dependency (fd). Like constraints, functional dependencies are drawn from the semantics of the application domain. Essentially, fd's describe how individual attributes are related. Functional dependencies are a kind of constraint among attributes within a relation and that have implications for "good" relational schema design. Here we will look at:

- basic theory and definition of functional dependencies
- methodology for improving schema designs (normalization)

The aim of studying this is to improve understanding of relationships among data and to gain enough formalism to assist practical database design.

Relational Design and Redundancy

Generally, a good relational database design must capture all of the necessary attributes/associations and should do this with a minimal amount of stored information (it means there is no redundant data).

In database design, redundancy is generally a "bad thing" because it causes problems maintaining consistency after updates. However, it can sometimes lead to performance improvements e.g. may be able to avoid a join to collect bits of data together.

Consider the following table defining bank accounts/branches:

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Bank Accounts

Source: http://cnx.org/content/m28252/latest/

Insertion anomaly

When we insert a new record, we need to check that branch data is consistent with existing rows.

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000
A-306	800	1111111	Round Hill	Horseneck	8000800

Insertion anomaly - Insert account A-306 at Round Hill

Source: http://cnx.org/content/m28252/latest/

Update anomaly

If a branch changes address, we need to update all rows referring to that branch.

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Palo Alto	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Update Anomaly - Round Hill branch address

Source: http://cnx.org/content/m28252/latest/

Deletion anomaly

If we remove information about the last account at a branch (Downtown), all of the branch information disappears

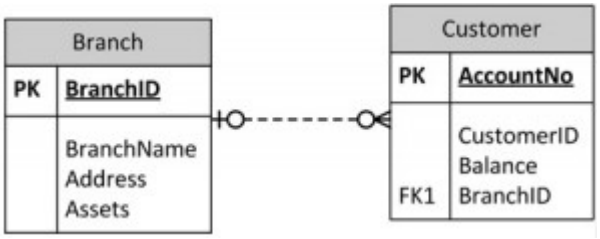
accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Deletion anomaly - Bank Account

Source: http://cnx.org/content/m28252/latest/

The problem is that after deleting the row, we don't know where the Downtown branch is located and we lose all information regarding customer '1313131'. To avoid these kinds of update/delete problems we need to decompose the original table into several smaller tables where each table has minimal overlap with other tables. Typically, each table contains information about one entity (e.g. branch, customer, …)

The Bank Accounts tables should appear as follows



This will ensure that when branch information is added or updated it will only affect one record. When customer information is added or deleted, the Branch information will not be accidently modified or incorrectly recorded.

Source: http://cnx.org/content/m28252/latest/

Another Example:

Employee Project Table

Assumptions: EmpID and ProjectID is a composite PK

Project ID determines Budget. i.e. Project P1 has a budget of 32 hours.

EmpID	Budget	ProjectID	Hours
S75	32	P1	7
S75	40	P2	3
S79	32	P1	4
S79	27	P3	1
S80	40	P2	5
	17	P4	

Let's look at some possible anomalies:

Add row {S85,35,P1,9)

Problem: Two tuples with conflicting budgets

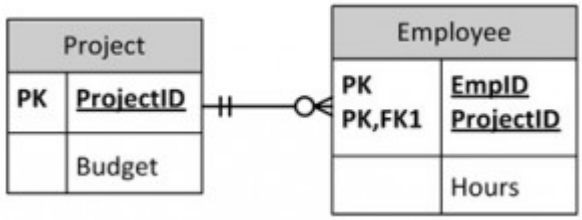
Delete tuple {S79, 27, P3, 1}

Problem: Deletes the budget of project P3

Update tuple {S75, 32, P1, 7} to {S75, 35, P1, 7)

Problem: Two tuples with different values for project P1's budget

To fix this, we need a table for employees and a table for projects.



Tables with Data

Project table

ProjectID	Budget
P1	32
P2	40
P3	27
P4	17

Employee table

EmpID	ProjectID	Hours
S75	P1	7
S75	P2	3
S79	P1	4
S79	P3	1
S80	P2	5

1. No anomalies will be created if a budget is changed
2. No dummy values are needed for projects that have no employees assigned
3. If an employee's contribution is deleted, no important data is lost
4. No anomalies are created if an employee's contribution is added.

The best approach to creating tables without anomalies is to ensure tables are normalized and that's accomplished by understanding functional dependencies (FD). FD ensures that all attributes in a table belong to that table. In other words, it will eliminate redundancies and anomalies.

11

Chapter 11 Functional Dependencies

A functional dependency is a relationship between two attributes. Typically between the PK and other non-key attributes with in the table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y.

X ————> Y

The left-hand side of the FD is called the determinant, and the right-hand side is the dependent.

Examples:

SIN ————-> Name, Address, Birthdate

SIN determines names and address and birthdays. Given SIN, we can determine any of the other attributes within the table.

Sin, Course ————> DateCompleted

Sin and Course determine date completed. This must also work for a composite PK.

ISBN ————> Title

ISBN determines title.

Rules of Functional Dependencies

Consider the following instance r(R) of the relation schema R(ABCDE):

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b1	C2	d2	e1
a3	b2	C1	d1	e1
a4	b2	C2	d2	e1
a5	b3	C3	d1	e1

Table R

What kind of dependencies can we observe among the attributes in Table R?

- Since the values of A are unique, it follows from the FD definition that:
  A →B, A →C, A →D, A →E
- It also follows that A →BC (or any other subset of ABCDE).
- This can be summarized as A →BCDE
- From our understanding of primary keys, A is a Primary Key.
  Since the values of E are always the same, it follows that: A →E, B →E, C →E, D →E
  However, we cannot generally summarized above by ABCD →E
  In general, A →E, B →E AB →E

Other observations:

- combinations of BC are unique, therefore BC →ADE
- combinations of BD are unique, therefore BD →ACE
- if C values match, so do D values, therefore C →D however, D values don't determine C values, so C does not determine D, and D does not determine C.

When looking at the data, it makes a lot more sense in terms of which attributes are dependent and which are determinants.

Inference Rules

Armstrong's axioms are a set of axioms (or, more precisely, inference rules) used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong.

Let R(U) be a relation scheme over the set of attributes U. We will use the letters X, Y, Z to represent any subset of and, for short, the union of two sets of attributes and by instead of the usual X U Y.

Source: http://en.wikipedia.org/wiki/Armstrong%27s_axioms

Axiom of reflexivity

$\text{If } Y \subseteq X, \text{ then } X \to Y$ If Y is a subset of X, then X determines Y

e g. PartNo —> NT123 – composed of category (NT) and partID (123)

X **X** **Y**

Axiom of augmentation

- Also known as a Partial dependency

$\text{If } X \to Y, \text{ then } XZ \to YZ \text{ for any } Z$ Studentno, course —> studentName, address, city, prov, pc, grade, dateCompleted

This situation is not desirable, because every non key attribute has to be fully dependent on the PK. In this situation Student information is only 'partially' dependent on the PK; StudentNo.

To fix this problem, we need to break down the table into two as follows:

StudentNo, course à grade, dateCompleted

StudentNo à studentName, address, city, prov, pc

Axiom of transitivity

$\text{If } X \to Y \text{ and } Y \to Z, \text{ then } X \to Z$ If X determines Y and Y determines Z, then X must also determine Z.

*StudentNoà studentName,*address, city, prov, pc, *ProgramID, ProgramName*

X Y Z

This situation is not desirable, because a non key attributes depends on another non key attribute.

To fix this problem, we need to break this table into two; one to hold information about the student and the other to hold information about the program. However we still need to leave a FK in the student table, so that we can determine which program the Student is enrolled in.

*StudentNo—> studentName,*address, city, prov, pc, *ProgramID*

ProgramID —> ProgramName

Additional rules

Union

$\textbf{If } X \rightarrow Y \textbf{ and } X \rightarrow Z \textbf{ then } X \rightarrow YZ$ **If X determines Y and X determines Z then X must also determines Y and Z.**

This is suggesting that if two tables are separate, and the PK is the same, you may want to consider putting them together.

SIN —> EmpName

SIN —> SpouseName

You may want to join these two tables into one.

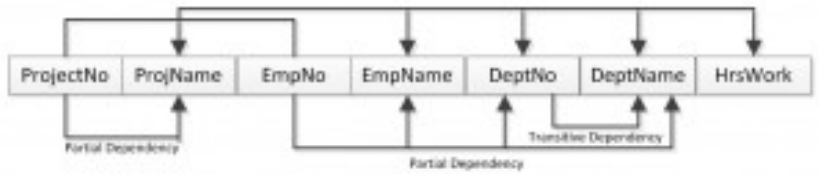
SIN –> EmpName, SpouseName

Some DBAs would leave them separated for a couple of reasons. They describes two entities, so should be separated. If in one table, the spouse name may be left NULL most of the time, so there is no need to have it in the same table.

Decomposition

$\textbf{If } X \rightarrow YZ \textbf{ then } X \rightarrow Y \textbf{ and } X \rightarrow Z$ If X determines Y and Z, then X determines Y and X determines Z separately. This is the reverse of Union. If you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables.

Dependency Diagram



A dependency diagram illustrates the various dependencies that may exist in a non normalized table. The following dependencies are identified:

ProjectNo, and EmpNo combined is the PK.

Partial Dependencies:

ProjectNo —> ProjName

EmpNo —> EmpName, DeptNo, HrsWork

Transitive Dependency:

DeptNo —> DeptName

12

Chapter 12 Normalization

Normalization is the branch of relational theory providing design insights. The goals of normalization are:

- be able to characterize the level of redundancy in a relational schema
- provide mechanisms for transforming schemas to remove redundancy

Normalization draws heavily on the theory of functional dependencies. Normalization theory defines six normal forms (NFs). Each normal form involves a set of dependency properties that a schema must satisfy and each gives guarantees about presence/absence of update anomalies. That means higher normal forms have less redundancy so less update problems.

Normal Forms

All the tables in any database can be in one of the normal forms we will discuss next. Normalization is the process of determining how much redundancy exists in your tables. Idealy we only want minimal redundancy. That is PK to FK. Everything else should be derived from other tables. There are 6 normal forms, but we will only look at the first 4. The last two are rarely used.

1st Normal Form

2nd Normal Form

3rd Normal Form

BCNF

First Normal Form

In the first normal form only single values are permitted at the intersection of each row and column hence, there are no repeating groups.

To normalize a relation that contains a repeating group, remove the repeating group and form two new relations.

The primary key of the new relation is a combination of the primary key of the original relation plus an attribute from the newly created relation for unique identification.

School database

Student_Grade_Report(Studentno, student_name, major,course_no, course_name, instructor_no, instructor_name, instructor_location, grade)

Process for 1st NF

- In the Student Grade Report table the repeating group is the course information. A student can take many courses.
- Remove the repeating group. In this case it's the Course information for each student.
- Identify the Primary Key in your new table.
- Primary key must uniquely identify attribute value (StudentNo, and CourseNo)
- After removing all the attributes related to the Course and Student, what's left is the Student_course table. Student (**Student_no**,student_name,major)
- The Student table is in First Normal Form (repeating group removed)

Student (Student_no, student_name, major)
Student_Course (Studentno, course_no, course_name,instructor_no,instructor_name, instructor_location, grade)

Update anomalies in 1st normal form

Student_Course (Studentno, course_no, course_name, instructor_no, instructor_name, instructor_location, grade)

- To add a new course, we need a student
- When Course information needs to be updated, we may have inconsistencies
- To delete a student, we might also delete critical information about a course

2nd Normal Form

- The relation must first be in 1st NF.
- The relation is automatically in 2nd NF, if and only if the primary key comprises a single attribute.
- If the relation has a composite primary key, then each non-key attribute must be fully dependent on the entire primary key and not on a subset of the primary key (i.e. there must be no partial dependency – augmentation).

Process for 2nd NF

To move on to 2nd NF, the table must be in 1st NF.

- The student table is already in 2nd NF because it has a single column PK.
- When examining the Student Course table, we see that not all the attributes are fully dependent on the PK. Specifically, all course information. The only attribute that is fully dependent is grade.
- Identify the new table that contains the course information.
- Identify the PK for the new table.

Student (Studentno,student_name,major)
Course_Grade (studentno,course_no,grade)
Course_Instructor(Course_no, course_name, instructor_no, instructor_name, instructor_location)

Update anomalies in 2nd normal form

- When adding a new instructor we need a course
- Updating course information could lead to inconsistencies for instructor information
- Deleting a course may also delete instructor information

3rd Normal Form

Remove Transitive Dependencies

- The relation has to be in 2nd normal form.
- Remove all transitive dependencies – a non-key attribute may not be functionally dependent on another non-key attribute
- Eliminate all dependent attributes in transitive relationship(s) from each of the tables that have a transitive relationship
- Create new table(s) with removed dependency.
- Check new table(s) as well as table(s) modified to make sure that each table has determinant and that no table contains inappropriate dependencies

Student (**Studentno**, student_name, major)
Course_Grade (**Studentno, course_no**, grade)
Course (**Course_no**,course_name,instructor_no)
Instructor (**Instructor_no**, instructor_name, instructor_location)

Update anomalies in 3rd normal form

At this stage there should be no anomalies in 3rd normal form.

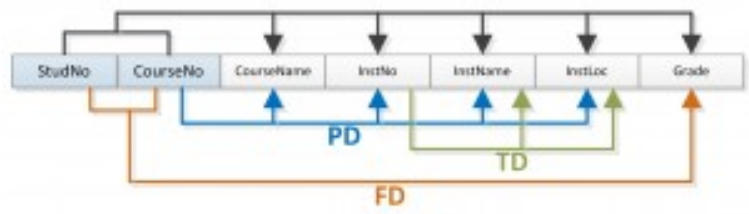
Lets look at the dependency diagrams for this example.

The first step is to remove repeating groups. As discussed above.

Student (Student_no, student_name, major)

Student_Course (Studentno, course_no, course_name,instructor_no,instructor_name, instructor_location, grade)

To recap the normalization process for the School database, the following dependencies are shown in the diagram:



PD – Partial Dependency
TD – Transitive Dependency
FD – Full Dependency

Boyce-Codd Normal Form (BCNF)

When a table has more than one candidate key, anomalies may result even though the relation is in 3NF. Boyce-Codd normal form is a special case of 3NF. A relation is in BCNF if and only if every determinant is a candidate key.

Consider the following table

St_Maj_Adv table

Student_id	Major	Advisor
123	Physics	Einstein
123	Music	Mozart
456	Biology	Darwin
789	Physics	Bohr
999	Physics	Einstein

Semantic Rules:

1. Each student may major in several subjects
2. For each major, a given student has only one advisor
3. Each major has several advisors
4. Each advisor advises only one major
5. Each advisor advises several students in one major

Functional dependencies:

Fd1: Student_id,Major ——> Advisor (candidate Key)

Fd2: Advisor ——> Major (not a candidate key)

ANOMALIES

Deleting student deletes advisor info

Insert a new advisor – need a student

Update – inconsistencies

Note: no single attribute is a candidate key

Primary key can be **student_id,major** or **student_id,advisor**

To reduce the St_Maj_Adv relation to BCNF, create two new tables:

1. St_Adv (Student_id,Advisor)
2. Adv_Maj (Advisor,Major)

St_Adv

Student_id	Advisor
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

Adv_Maj

Advisor	Major
Einstein	Physics
Mozart	Music
Darwin	Biol
Bohr	Physics

BCNF Example 2

Client Interview Table

ClientNo	**InterviewDate**	InterViewTime	StaffNo	RoomNo
CR76	13-May-02	10.30	SG5	G101
CR56	13-May-02	12.00	SG5	G101
CR74	13-May-02	12.00	SG37	G102
CR56	1-July-02	10.30	SG5	G102

Fd1 – ClientNo, InterviewDate –> interviewTime, staffNo, roomNo (PK)
Fd2 – staffNo, InterviewDate, InterviewTime –> ClientNO (CK)
Fd3 – roomNo, InterviewDate, InterviewTime –> StaffNo, clientNo (CK)
Fd4 – staffNo, interviewDate –> roomNo

A relation is in BCNF if and only if every determinant is a candidate key. We need to create a table that incorporates the first 3 Fds and another table for the 4th Fd.

Interview

ClientNo	**InterviewDate**	InterViewTime	StaffNo
CR76	13-May-02	10.30	SG5
CR56	13-May-02	12.00	SG5
CR74	13-May-02	12.00	SG37
CR56	1-July-02	10.30	SG5

StaffRoom

StaffNo	**InterviewDate**	RoomNo
SG5	13-May-02	G101
SG37	13-May-02	G102
SG5	1-July-02	G102

Source: http://db.grussell.org/section008.html

Normalization and Database Design

Normalization should be part of design process. Make sure that proposed entities meet required normal form before table structures are created. Many real-world databases have been improperly designed or burdened with anomalies if improperly modified during the course of time. You may be asked to redesign and modify existing databases. This could be a really big endeavor if the tables are not properly normalized.

Use an Entity Relation Diagram (ERD) to provide a big picture. The ERD shows a macro view of an organization's data requirements and operations. The ER Diagram is created through an iterative process which involves identifying relevant entities, their attributes and their relationships.

Normalization procedure focuses on characteristics of specific entities and it represents the micro view of entities within ERD.

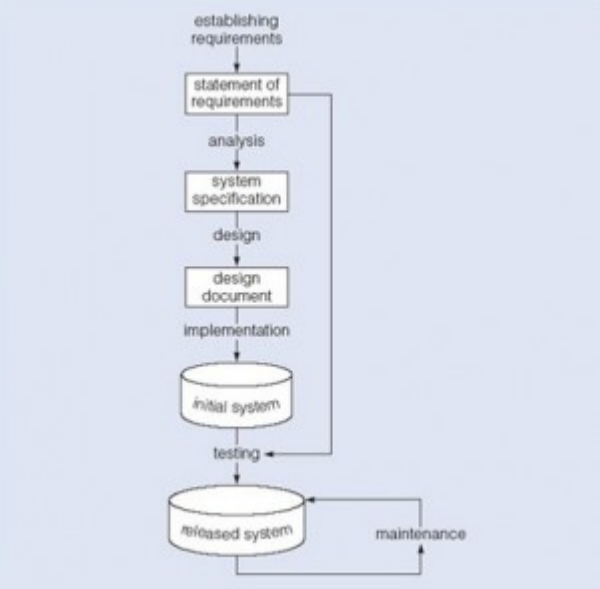
It is difficult to separate normalization process from ER modeling process. The two techniques should be used concurrently.

13

Chapter 13 Database Development Process

A core aspect of software engineering is the subdivision of the development process into a series of phases, or steps, each of which focuses on one aspect of the development. The collection of these steps is sometimes referred to as a development life cycle. The software product moves through this life cycle (sometimes repeatedly as it is refined or redeveloped) until it is finally retired from use. Ideally, each phase in the life cycle can be checked for correctness before moving on to the next phase.

Let us start with an overview of the waterfall model such as you will find in most software engineering text books.



Source: http://www.oercommons.org/courses/the-database-development-life-cycle/view

SDLC – Waterfall

The waterfall figure above illustrates a general waterfall model which could apply to any computer system development. It shows the process as a strict sequence of steps where the output of one step is the input to the next and all of one step has to be completed before moving onto the next.

We can use the waterfall process as a means of identifying the tasks that are required, together with the input and output for each activity. What is important is the scope of the activities, which can be summarised as follows:

- **Establishing requirements** involves consultation with, and agreement among, **stakeholders** as to what they want of a system, expressed as a statement of requirements.
- **Analysis** starts by considering the statement of requirements and finishes by producing a system specification. The specification is a formal representation of what a system should do, expressed in terms that are independent of how it may be realized.
- **Design** begins with a system specification and produces design documents, and provides a detailed description of how a system should be constructed.

- **Implementation** is the construction of a computer system according to a given design document and taking account of the environment in which the system will be operating (for example specific hardware or software available for the development). Implementation may be staged, usually with an initial system than can be validated and tested before a final system is released for use.
- **Testing** compares the implemented system against the design documents and requirements specification and produces an acceptance report or, more usually, a list of errors and bugs that require a review of the analysis, design and implementation processes to correct (testing is usually the task that leads to the waterfall model iterating through the life cycle).
- **Maintenance** involves dealing with changes in the requirements, or the implementation environment, bug fixing or porting of the system to new environments (for example migrating a system from a standalone PC to a UNIX workstation or a networked environment). Since maintenance involves the analysis of the changes required, design of a solution, implementation and testing of that solution over the lifetime of a maintained software system, the waterfall life cycle will be repeatedly revisited.

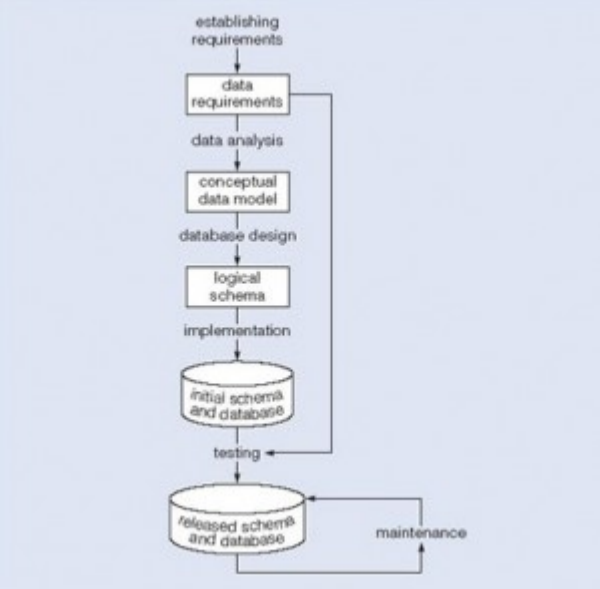
http://cnx.org/content/m28139/latest/

Database Life Cycle

We can use the waterfall cycle as the basis for a model of database development which incorporates three assumptions:

- We can separate the development of a database – that is, specification and creation of a schema to define data in a database – from the user processes that make use of the database.
- We can use the three-schema architecture as a basis for distinguishing the activities associated with a schema.
- We can represent the constraints to enforce the semantics of the data once, within a database, rather than within every user process that uses the data.

http://www.oercommons.org/courses/the-database-development-life-cycle/view



Source: http://www.oercommons.org/courses/the-database-development-life-cycle/view

Using these assumptions, the diagram above represents a model of the activities and their outputs for database development. It is applicable to any class of **DBMS** (database management system), not just a relational approach.

http://cnx.org/content/m28139/latest/

Database Application Development is the process of obtaining real-world requirements, analyzing requirements, designing the data and functions of the system and then implementing the operations in the system.

http://www.oercommons.org/courses/the-database-development-life-cycle/view

Requirements gathering

The first step is Requirements Gathering. During this step, the database designers have to interview the customers (database users) to understand the proposed system, obtain and document the data and functional requirements. The result of this step is a document including the detail requirements provided by the users.

Establishing requirements involves consultation with, and agreement among, all the users as to what persistent data they want to store along with an agreement as to the meaning and interpretation of the data elements. The data administrator plays a key role in this process as they overview the business, legal and ethical issues within the organization that impact on the data requirements.

The data requirements document is used to confirm the understanding of requirements with users. To make sure that it is easily understood, it should not be overly formal or highly encoded. The document should give a concise summary of all users' requirements – not just a collection of individuals' requirements – as the intention is to develop a single shared database.

The requirements should not describe how the data is to be processed, but rather what the data items are, what attributes they have, what constraints apply and the relationships that hold between the data items.

Analysis

Data analysis begins with the statement of data requirements and then produces a **conceptual data model**. The aim of analysis is to obtain a detailed description of the data that will suit user requirements so that

both high and low level properties of data and their use are dealt with. These include properties such as the possible range of values that can be permitted for attributes such as, in the School Database example; for instance, the Student course code, course title and credit points.

The conceptual data model provides a shared, formal representation of what is being communicated between clients and developers during database development – it is focused on the data in a database, irrespective of the eventual use of that data in user processes or implementation of the data in specific computer environments. Therefore, a conceptual data model is concerned with the meaning and structure of data, but not with the details affecting how they are implemented.

The conceptual data model then is a formal representation of what data a database should contain and the constraints the data must satisfy. This should be expressed in terms that are independent of how the model may be implemented. As a result, analysis focuses on 'What is required?' not 'How is it achieved?

http://www.oercommons.org/courses/the-database-development-life-cycle/view

Logical Design

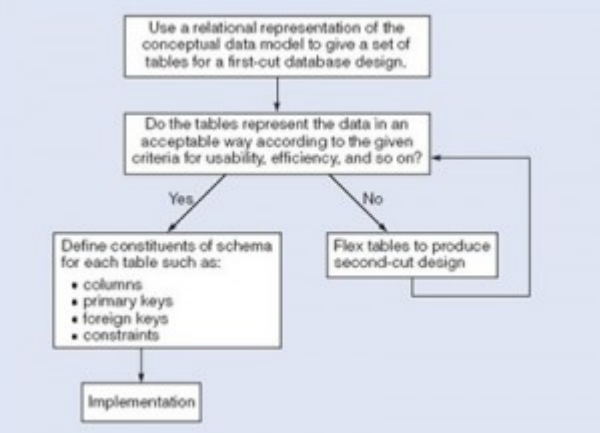
Database design starts with a conceptual data model and produces a specification of a logical schema; this will determine the specific type of database system (network, relational, object-oriented) that is required. The relational representation is still independent of any specific DBMS, it is another conceptual data model.

We can use a relational representation of the conceptual data model as input to the logical design process. The output of this stage is a detailed relational specification, the logical schema, of all the tables and constraints needed to satisfy the description of the data in the conceptual data model. It is during this design activity that choices are made as to which tables are most appropriate for representing the data in a database. These choices must take into account various design criteria including, for example, flexibility for change, control of duplication and how best to represent the constraints. It is the tables defined by the logical schema that determine what data are stored and how they may be manipulated in the database.

Database designers familiar with relational databases and SQL might be tempted to go directly to implementation after they have produced a conceptual data model. However, such a direct transformation of the relational representation to SQL tables does not necessarily result in a database that has all the desirable properties: completeness, integrity, flexibility, efficiency and usability. A good conceptual data model is an essential first step *towards* a database with these properties, but that does not mean that the direct transformation to SQL tables automatically produces a good database. This first step will accurately represent the tables and constraints needed to satisfy the conceptual data model description, and so satisfies the completeness and integrity requirements, but it may be inflexible or offer poor usability. The first design is then **flexed** to improve the quality of the database design. Flexing is a term that is intended to capture the simultaneous ideas of bending something for a different purpose and weakening aspects of it as it is bent.

The figure below summarizes the iterative (repeated) steps involved in database design, based on the overview given. Its main purpose is to distinguish the general issue of what tables should be used from the detailed definition of the constituent parts of each table – these tables are considered one at a time, although they are not independent of each other. Each iteration that involves a revision of the tables would lead to a new design; collectively they are usually referred to as **second-cut designs**, even if the process iterates for more than a single loop.

http://www.oercommons.org/courses/the-database-development-life-cycle/view



Source: http://www.oercommons.org/courses/the-database-development-life-cycle/view

First, for a given conceptual data model it is not necessary that all the user requirements it represents have to be satisfied by a single database. There can be various reasons for the development of more than one database, such as the need for independent operation in different locations or departmental control over 'their' data. However, if the collection of databases contains duplicated data and users need to access data in more than one database, then there are possible reasons that the one database can satisfy multiple requirements or issues related to data replication and distribution need to be examined.

Second, one of the assumptions about database development is that we can separate the development of a database from the development of user processes that make use of it. This is based on the expectation that, once a database has been implemented, all data required by currently identified user processes have been defined and can be accessed; but we also require flexibility to allow us to meet future requirements changes. In developing a database for some applications it may be possible to predict the common requests that will be presented to the database and so we can optimize our design for the most common requests.

Third, at a detailed level, many aspects of database design and implementation depend on the particular DBMS being used. If the choice of DBMS is fixed or made prior to the design task, that choice can be used to determine design criteria rather than waiting until implementation. That is, it is possible to incorporate design decisions for a specific DBMS rather than produce a generic design and then tailor it to the DBMS during implementation.

It is not uncommon to find that a single design cannot simultaneously satisfy all the properties of a good database. So it is important that the designer has prioritized these properties (usually using information from the requirements specification), for example, to decide if integrity is more important than efficiency and whether usability is more important than flexibility in a given development.

At the end of our design stage the logical schema will be specified by SQL data definition language (DDL) statements, which describe the database that needs to be implemented to meet the user requirements.

http://www.oercommons.org/courses/the-database-development-life-cycle/view

Implementation

Implementation involves the construction of a database according to the specification of a logical schema. This will include the specification of an appropriate storage schema, security enforcement, external schema, and so on. Implementation is heavily influenced by the choice of available DBMS, database tools and operating environment. There are additional tasks beyond simply creating a database schema and implementing the constraints – data must be entered into the tables, issues relating to the users and user processes need to be addressed and the management activities associated with wider aspects of corporate data management

need to be supported. In keeping with the DBMS approach we want as many of these concerns as possible to be addressed within the DBMS. We look at some of these concerns briefly now.

In practice, implementation of the logical schema in a given DBMS requires a very detailed knowledge of the specific features and facilities that the DBMS has to offer. In an ideal world, and in keeping with good software engineering practice, the first stage of implementation would involve matching the design requirements with the best available implementing tools and then using those tools for the implementation. In database terms, this might involve choosing vendor products who's DBMS and SQL variants are most suited to the database we need to implement. However, we don't live in an ideal world and more often than not, hardware choice and decisions regarding the DBMS will have been made well in advance of consideration of the database design. Consequently, implementation can involve additional flexing of the design to overcome any software or hardware limitations.

Realising the design

After the Logical Design has been created, we need our database to be created according to the definitions we have produced. For an implementation with a relational DBMS, this will probably involve the use of SQL to create tables and constraints that satisfy the logical schema description and the choice of appropriate storage schema (if the DBMS permits that level of control).

One way to achieve this is to write the appropriate SQL DDL statements into a file that can be executed by a DBMS so that there is an independent record, a text file, of the SQL statements defining the database. Another method is to work interactively using a database tool like SQL Server Management Studio or Microsoft Access. Whatever mechanism is used to implement the logical schema, the result is that a database, with tables and constraints, is defined but will contain no data for the user processes.

Populating the database

After a database has been created, there are two ways of populating the tables – either from existing data, or through the use of the user applications developed for the database.

For some tables, there may be existing data from another database or data files. For example, in establishing a database for a hospital you would expect that there are already some records of all the staff that have to be included in the database. Data might also be bought in from an outside agency (address lists are frequently brought in from external companies) or produced during a large data entry task (converting hard-copy manual records into computer files can be done by a data entry agency). In such situations the simplest approach to populate the database is to use the import and export facilities found in the DBMS.

Facilities to import and export data in various standard formats are usually available (these functions are also known in some systems as loading and unloading data). Importing enables a file of data to be copied directly into a table. When data are held in a file format that is not appropriate for using the import function then it is necessary to prepare an application program that reads in the old data, transforms them as necessary and then inserts them into the database using SQL code specifically produced for that purpose. The transfer of large quantities of existing data into a database is referred to as a **bulk load**. Bulk loading of data may involve very large quantities of data being loaded, one table at a time so you may find that there are DBMS facilities to postpone constraint checking until the end of the bulk loading.

http://www.oercommons.org/courses/the-database-development-life-cycle/view

Guidelines For Developing An ER Diagram

NOTE: These are general guidelines which will assist in developing a strong basis for the actual database design (the logical model)

1. Document all entities discovered during the information gathering stage.
2. Document all attributes that belong to each entity. Select candidate and primary keys. Ensure that all non-key attributes for each entity are full-functionally dependent on the primary key.
3. Develop an initial E-R diagram and review with appropriate personnel. (Remember that this is an iterative process).
4. Create new entities (tables) for multi-valued attributes and repeating groups. Incorporate these new entities (tables) in the E-R diagram. Review with appropriate personnel.
5. Verify E-R modeling by normalizing tables.

14

Chapter 14 Database Users

End users

These are people whose jobs require access to a database for querying, updating and generating reports. An end user might be one of the following:

The application user uses the existing application programs to perform their daily tasks.

Sophisticated users are those who have their own way of accessing the database. This mean they do not use the application program provided in the system. Instead, they might define their own application or describe their need directly using query languages. Specialized users maintain their personal database by using ready –made program packages that provide easy-to-use menu driven commands – such as MS Access.

Application Programmers

People implement specific application programs to access to the stored data. This kind of user needs to be familiar with the DBMSs to accomplish their task.

Database Administrators

A person or a group of people in the organization who is responsible for authorizing the access to the database, monitoring its use and managing all the resource to support the use of the whole database system.

Appendix A University Registration Data Model example

Here is a statement of the data requirements for a product to support the registration of and provide help to students of a fictitious e-learning university.

An e-learning university needs to keep details of its students and staff, the courses that it offers and the performance of the students who study its courses. The university is administered in four geographical regions (England, Scotland, Wales and Northern Ireland).

Information about each student should be initially recorded at registration. This includes the student's identification number issued at the time, name, year of registration and the region in which the student is located. A student is not required to enroll on any courses at registration; enrolment on a course can happen at a later time.

Information recorded for each member of the tutorial and counseling staff must include the staff number, name and the region in which he or she is located. Each staff member may act as a counselor to one or more students, and may act as a tutor to one or more students on one or more courses. It may be the case that, at any particular point in time, a member of staff may not be allocated any students to tutor or to counsel.

Each student has one counselor, allocated at registration, who supports the student throughout his or her university career. A student is allocated a separate tutor for each course on which he or she is enrolled. A staff member may only counsel or tutor a student who is resident in the same region as that member of staff.

Each course that is available for study must have a course code, a title, and a value in terms of credit points. A course is either a 15-point course or a 30-point course. A course may have a quota for the number of students enrolled on it on any one presentation. A course need not have any students enrolled on it (such as a course that has just been written and offered for study).

Students are constrained in the number of courses they can be enrolled on at any one time. They may not take courses simultaneously if their combined points total exceeds 180 points.

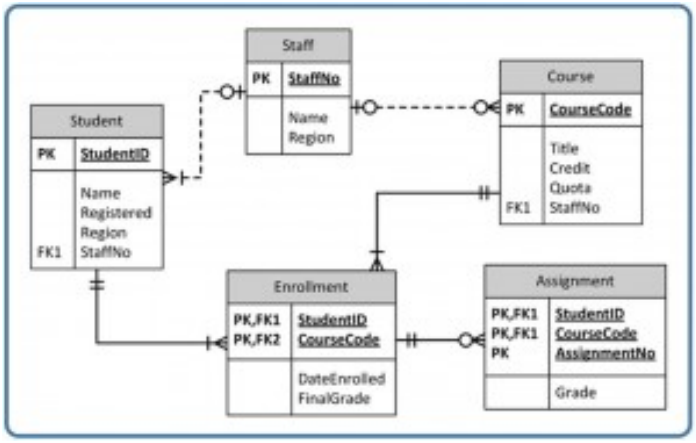
For assessment purposes, a 15-point course may have up to three assignments per presentation and a 30-point course may have up to five assignments per presentation. The grade for an assignment on any course is recorded as a mark out of 100.

University Database below is one possible data model that describes the above set of requirements. The model has several parts, beginning with an E–R diagram and followed by a written description of entity types, constraints and assumptions.

http://openlearn.open.ac.uk/mod/oucontent/view.php?id=397581&section=8.2

Design Process

1. The first step is to determine the kernels. These are typically nouns. Staff, Course, Student, and Assignment.
2. The next step is to document all attributes for each entity. This is where you need to ensure that all tables are properly normalized.
3. The initial ER diagram is created and reviewed with the users.
4. Changes may need to be made after reviewing ERD.
5. Verify the ER Model with users to finalize the design.



Source: http://openlearn.open.ac.uk/mod/oucontent/view.php?id=397581&section=8.2

University ERD – A data model for a student and staff records system

Entity

Student (StudentId, Name, Registered, Region, StaffNo)

Staff (StaffNo, Name, Region) – This table contains instructors and other staff members

Course (CourseCode, Title, Credit, Quota)

Enrolment (StudentId, CourseCode, Date Enrolled, FinalGrade)

Assignment (StudentId, CourseCode, AssignmentNo, Grade)

Constraints

- A staff member may only tutor or counsel students who are located in the same region as the member of staff.
- Students may not enroll for more than 180 points worth of courses at any one time.
- The attribute Credit (of Course) has a value of 15 or 30 points.
- A 30-point course may have up to five assignments; a 15-point course may have up to three assignments.
- The attribute Grade (of Assignment) has a value that is a mark out of 100.

Assumptions

- A student has at most one enrolment on a course as only current enrolments are recorded.
- An assignment may be submitted only once.

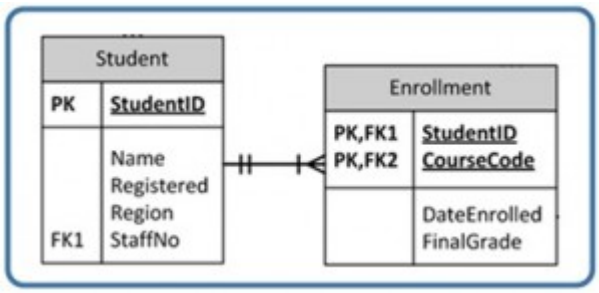
http://openlearn.open.ac.uk/mod/oucontent/view.php?id=397581&section=8.2

Relationships (includes cardinality)

A student (record) is associated with (enrolled) with a minimum of 1 to a maximum of many courses.

Each enrollment must have a valid Student.

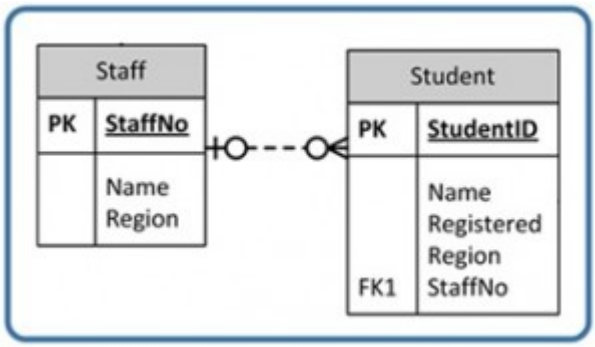
Note: Since the StudentID is part of the PK, it can't be null. Therefore, any StudentID entered, must exist in the Student table at least one to a maximum of one time. This should be obvious since the PK cannot have duplicates.



A Staff record (a tutor) is associated with a minimum of 0 students to a maximum of many students.

A Student record may or may not have a tutor.

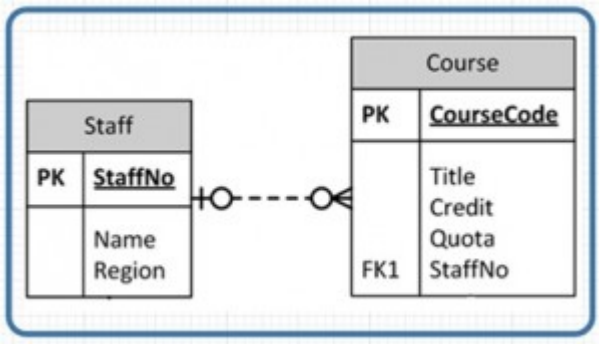
Note: The staffNo field in Student table allows null values – represented by the 0 on the left hand side. However, if a StaffNo exists in the student table it must exist in the Staff table maximum once – represented by the 1.



A staff record (instructor) is associated with a minimum of 0 courses to a maximum of many courses.

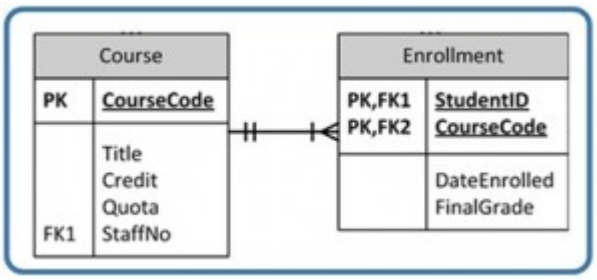
A course may or may not be associated with an Instructor.

Note: The StaffNo in Course table is the FK, and it can be Null. This represents the 0 on the left hand side of the relationship. If the StafNo has data, it has to be in the Staff table a maximum of once. That is represented by the 1 on the left hand side of the relationship.



A course must be offered (in enrollment) at least once to a maximum of many times.

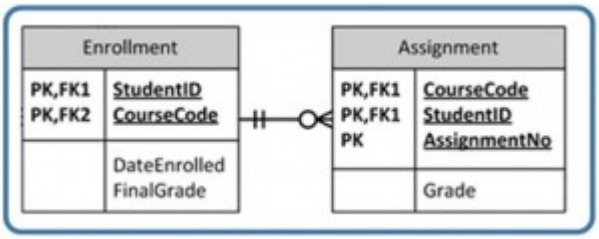
The enrollment table must contain at least one valid Course to a maximum of many.



An enrollment can have a minimum of 0 assignments or a maximum of many.

An assignment must be associated with at least one and a maximum of one enrollment.

Note: Every record in the Assignment table must contain a valid Enrollment record. One enrollment record can be associated with multiple assignments.



Appendix B ERD Exercises

A manufacturing company produces products. Product information stored is product name, id, quantity on hand. These products are made up of many components. Each component can be supplied by one or more suppliers. Component information kept is component id, name, description, suppliers who supply them, and which products they are used in.

Create an ERD to show how you would track this information.

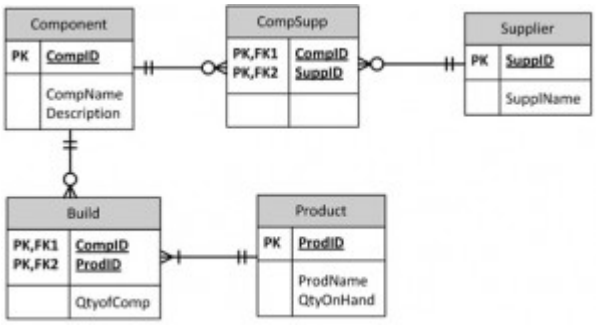
Show entity names, primary keys, attributes for each entity, relationships between the entities and cardinality.

Assumptions:

- A supplier can exist without providing components.
- A component does not have to be associated with a supplier.
- A component does not have to be associated with product. Not all components are used in products.
- A product cannot exist without components.

ERD Answer

Component(CompID, name, description) pk=CompID
Product(ProdID, name, QTYonHand) pk=ProdID
Supplier(SuppID, Coname) pk = SuppID
CompSupp(CompID,SuppID) pk = CompID,SuppID
Build(CompID, PID, QtyOfComp) pk= CompID,ProdID



2nd Exercise

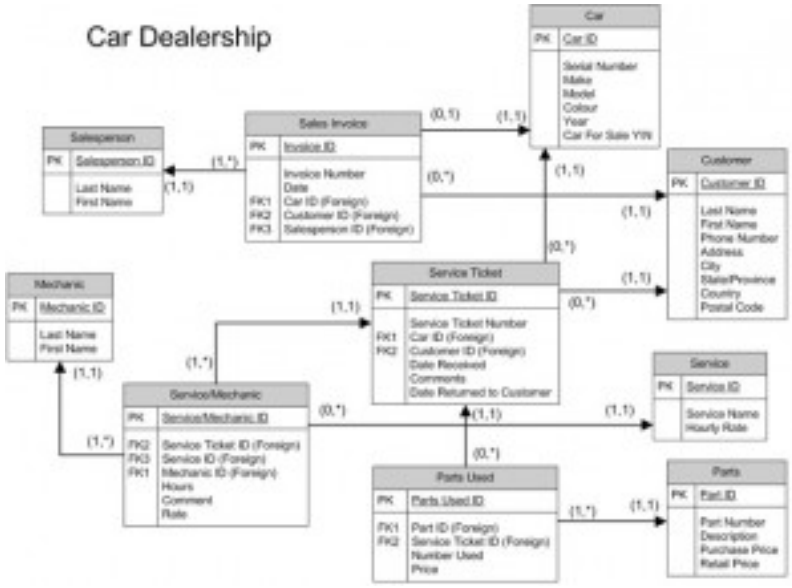
Car Dealership

Create an E-R diagram for a car dealership. The dealership sells both new and used cars, and it operates a service facility. Base your design on the following business rules:

- A salesperson may sell many cars, but each car is sold by only one salesperson.
- A customer may buy many cars, but each car is sold to only one customer.
- A salesperson writes a single invoice for each car he or she buys.
- A customer gets an invoice for each car he or she buys.

- A customer may come in just to have his or her car serviced, that is, one need not buy a car to be classified as a customer.
- When a customer takes one or more cars in for repair or service, one service ticket is written for each car.
- The car dealership maintains a service history for each of the cars serviced. The service records are referenced by the car's serial number.
- A car brought in for service can be worked on by many mechanics, and each mechanic may work on many cars.
- A car that is serviced may or may not need parts. (For example, adjusting a carburetor or
- cleaning a fuel injector nozzle does not require the use of parts).

Solution



References and Permissions

Cover image – DATABASE at Postmasters, March 2009 by Michael Mandiberg / CC BY-SA

http://cnx.org/content/m28150/latest/
http://en.wikipedia.org/wiki/Data_independence
http://cnx.org/content/m28252/latest/
http://en.wikipedia.org/wiki/Armstrong%27s_axioms
http://db.grussell.org/section008.html
http://openlearn.open.ac.uk/course/view.php?name=M359_1
http://cnx.org/content/m28139/latest/
http://openlearn.open.ac.uk/mod/oucontent/view.php?id=397581&section=8.2

About the Author

Adrienne Watt holds a Computer Systems Diploma (BCIT), a Bachelors in Technology (BCIT) and a Master's in Business Administration (City University).

Since 1989, Adrienne have worked as an educator and gained extensive experience developing and delivering business and technology curriculum to post-secondary students. During that time she ran a successful software development business. In the business she worked as an IT Professional in a variety of senior positions including Project Manager, Database Designer, Administrator and Business Analyst. Recently she has been exploring a wide range of technology related tools and processes to improve delivery methods and enhance learning for her students.