# DATABASE ANALYSIS AND DESIGN

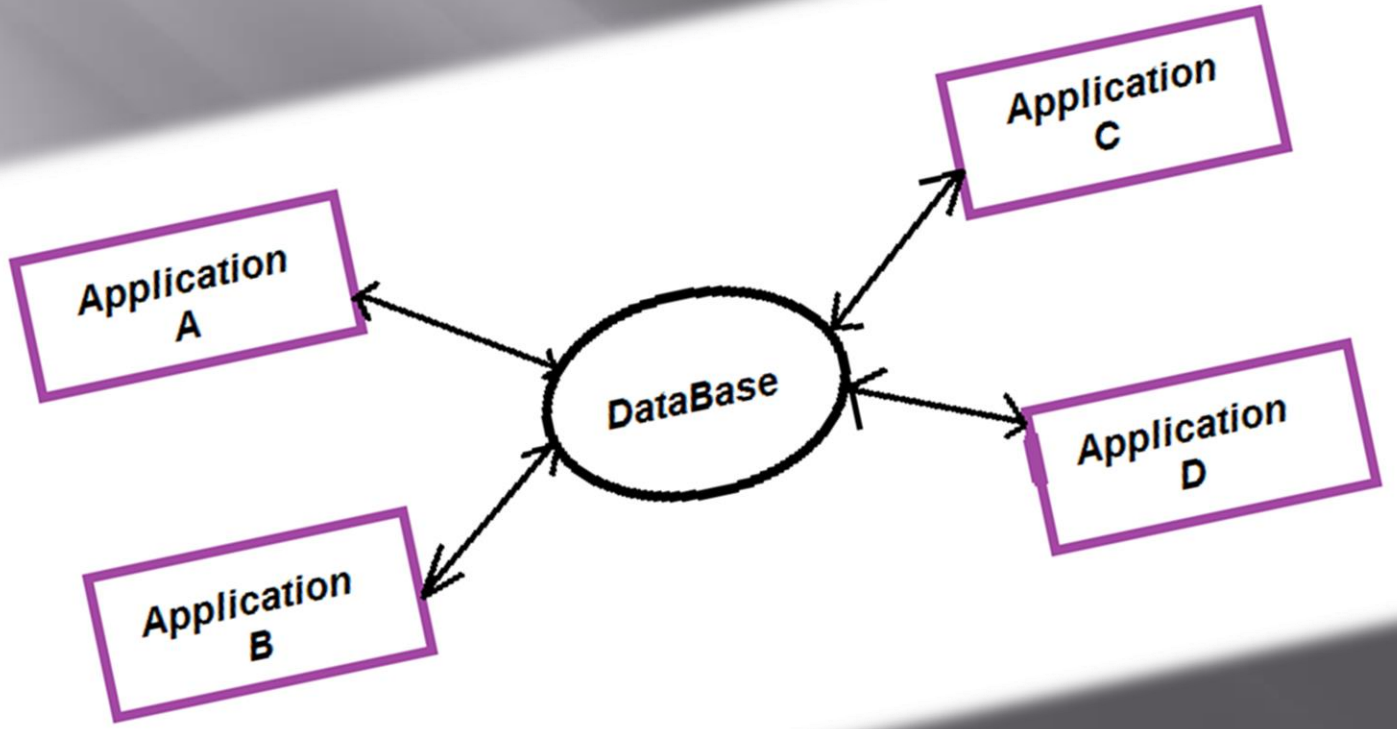
*introduction*

*Database analysis and Database design*

# What is database?

- A **database** is an organized collection of data. It is the collection of schemas, tables, queries, reports, views, and other objects. The data are typically organized to model aspects of reality in a way that supports processes requiring information, such as modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

**Database is a collection of non-redundant data shareable between different application systems.** ***This term means that the data should be stored as a common pool sharable between application systems.***



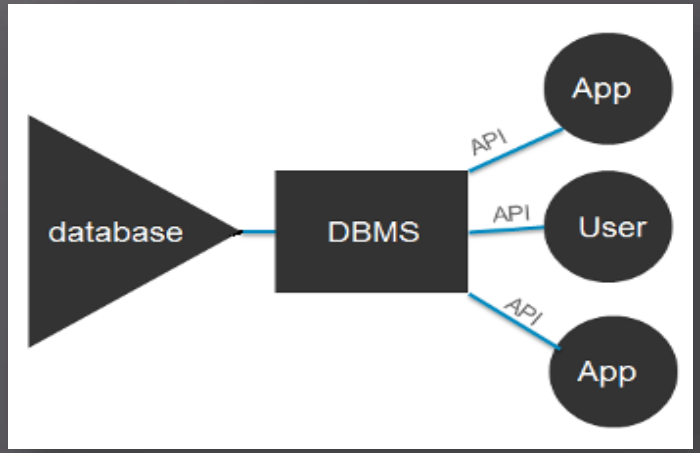
# The goal of using Database is:

- Modifiability : This means the change of the database in which some parts are altered without increasing the complexity
- Understandability: database that is easily understood acts like a bridge between the problem and its solution.
- Reliability: the reliability is a critical for any data base analyses and design.
- Efficiency: the database is efficient when it operates using available resources optimally.

# What is database management system?

- *A **database management system (DBMS)** is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, Microsoft SQL Server, and Oracle.*

Database refers to a set of related data and the way it is organized. Access to these data is usually provided by a "database management system" (DBMS) consisting of an integrated set of computer software that allows users to interact with one or more databases and provides access to all of the data contained in the database (although restrictions may exist that limit access to particular data). The DBMS provides various functions that allow entry, storage and retrieval of large quantities of information and provides ways to manage how that information is organized.



# Introduction to database analysis

Database analysis Concerned of establishing the services that the customer requires from system and the constraints under which it operates and is developed. The requirements themselves are the description of the system services and constraints that are generated . Data analysis is concerned with the NATURE and USE of data. It involves the identification of the data elements which are needed to support the data processing system of the organization, the placing of these elements into logical groups and the definition of the relationships between the resulting groups.

- database analysts often, in practice, go directly from fact finding to implementation dependent data analysis. Their assumptions about the usage of properties and relationships between data elements are embodied directly in record and file designs and computer procedure specifications. The introduction of Database Management Systems (DBMS) has encouraged a higher level of analysis, where the data elements are defined by a logical model or `schema' (conceptual schema).

It is fair to ask why data analysis should be done if it is possible, in practice to go straight to a computerized system design. Data analysis is time consuming; it throws up a lot of questions. Implementation may be slowed down while the answers are sought. It is more expedient to have an experienced analyst `get on with the job' and come up with a design straight away. The main difference is that data analysis is more likely to result in a design which meets both present and future requirements, being more easily adapted to changes in the business or in the computing equipment. It can also be argued that it tends to ensure that policy questions concerning the organizations' data are answered by the managers of the organization, not by the systems analysts. Data analysis may be thought of as the `slow and careful' approach, whereas omitting this step is `quick and dirty'.

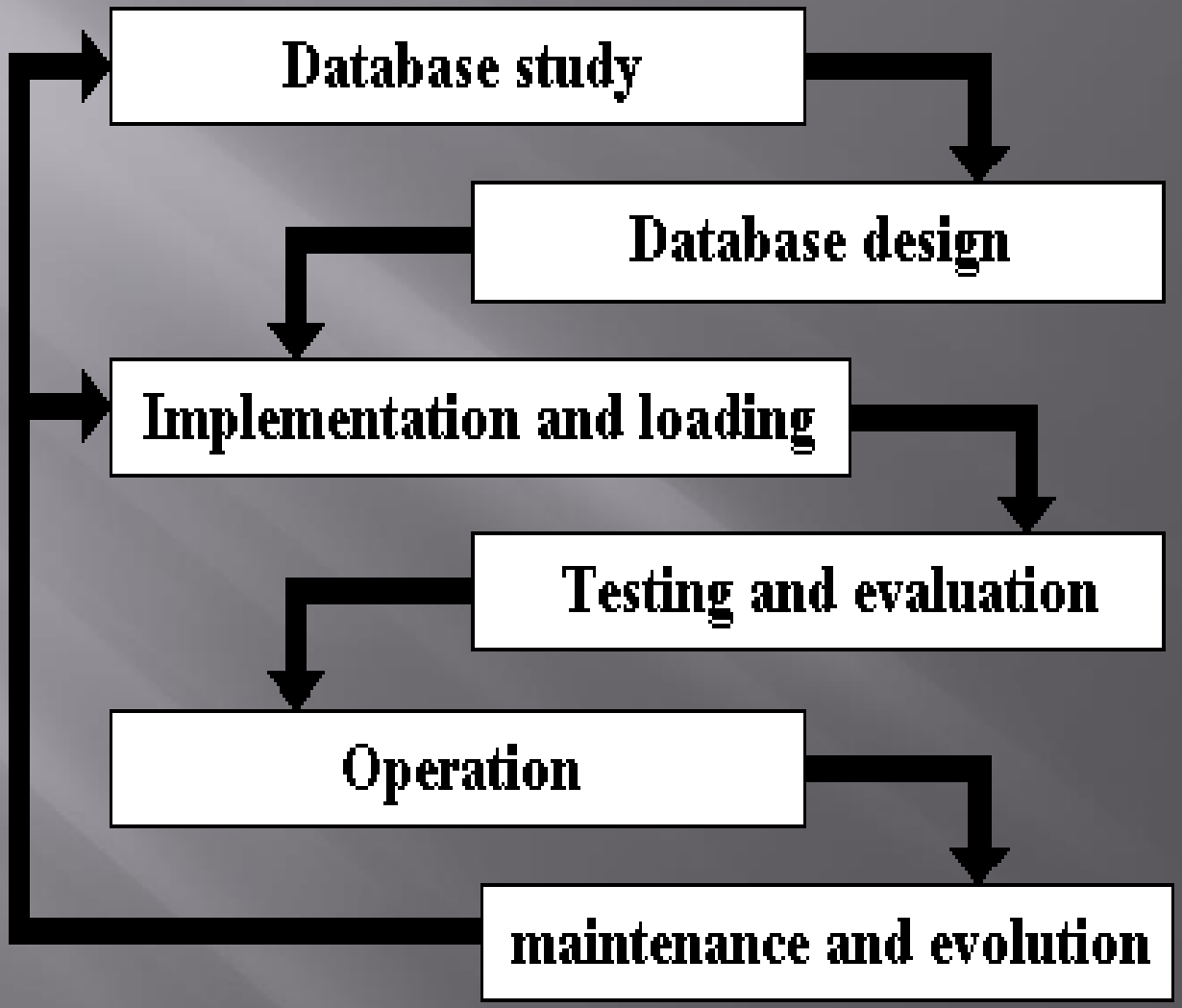
The development of techniques of data analysis has helped to understand the structure and meaning of data in organizations. Data analysis techniques can be used as the first step of extrapolating the complexities of the real world into a model that can be held on a computer and be accessed by many users. The data can be gathered by conventional methods such as interviewing people in the organization and studying documents. The facts can be represented as objects of interest. There are a number of documentation tools available for data analysis, such as entity relationship diagrams. These are useful aids to communication, help to ease the mapping processes that follow data analysis. Some of the documents can be used as source documents for the data dictionary. ensure that the work is carried out in a thorough manner, and

In data analysis we analyze the data and build system representation in the form of a data model (conceptual). A conceptual data model specifies the structure of the data and the processes which use that data.

Data Analysis = establishing the nature of data.

Functional Analysis = establishing the use of data.

# Database Analysis Life Cycle



- **Database study** - here the designer creates a written specification in words for the database system to be built. This involves:
  - analyzing the company situation - is it an expanding company, dynamic in its requirements, mature in nature, solid background in employee training for new internal products, etc. These have an impact on how the specification is to be viewed.
  - Define problems and constraints - what is the situation currently? How does the company deal with the task which the new database is to perform? Any issues around the current method? What are the limits of the new system?
  - Define objectives - what is the new database system going to have to do, and in what way must it be done. What information does the company want to store specifically, and what does it want to calculate. How will the data evolve?
  - Define scope and boundaries - what is stored on this new database system, and what it stored elsewhere. Will it interface to another database? .

- **Database Design** - conceptual, logical, and physical design steps in taking specifications to physical implementable designs. This is looked at more closely in a moment.
- **Implementation and loading** - it is quite possible that the database is to run on a machine which as yet does not have a database management system running on it at the moment. If this is the case one must be installed on that machine. Once a DBMS has been installed, the database itself must be created within the DBMS. Finally, not all databases start completely empty, and thus must be loaded with the initial data set .

- **Testing and evaluation** - the database, once implemented, must be tested against the specification supplied by the client. It is also useful to test the database with the client using mock data, as clients do not always have a full understanding of what they thing they have specified and how it differs from what they have actually asked for! In addition, this step in the life cycle offers the chance to the designer to fine-tune the system for best performance. Finally, it is a good idea to evaluate the database in-situ, along with any linked applications.
- **Operation** - this step is where the database is actually in real usage by the company.
- **Maintenance and evolution** - designers rarely get everything perfect first time, and it may be the case that the company requests changes to fix problems with the system or to recommend enhancements or new requirements.

# What are the requirements?

The requirements of the database are the descriptions of the services provided by the database and its operational constrains, this requirement reflects the need of costumers for the system that helps solve some problems. We can distinguish between two levels of requirements process:

1. User requirements: these are statements in natural language plus diagrams of what services the system is expected to provide and constraints under with it must operate.

2. System requirements: these are the set out the system's functions, services and operational constraints in detail.

# How to specify the database

The specification is usually in the form of written document containing customer requirements, mock reports, screen drawings and the like, written by the client to indicate the requirements which the final system is to have. Often such data has to be collected together from a variety of internal sources to the company and then analyzed to see if the requirements are necessary, correct, and efficient.

# Conceptual Design

Once the Database requirements have been collated, the Conceptual Design phase takes the requirements and produces a high-level data model of the database structure. In this module, we use ER modeling to represent high-level data models, but there are other techniques. This model is independent of the final DBMS which the database will be installed in.

# conceptual schema and physical design

Next, the Conceptual Design phase takes the high-level data model it taken and converted into a conceptual schema, which is specific to a particular DBMS class (e.g. relational). For a relational system, such as Oracle, an appropriate conceptual schema would be relations.

Finally, in the Physical Design phase the conceptual schema is converted into database internal structures. This is specific to a particular DBMS product.

# User requirements and system requirements

1. User requirements : often referred to as user needs, describe what the user does with the system, such as what activities that users must be able to perform. User requirements are generally documented in a User Requirements Document (URD) using narrative text. User requirements are generally signed off by the user and used as the primary input for creating system requirements.

2. System requirements : are the building blocks developers use to build the system. These are the traditional "shall" statements that describe what the system "shall do." System requirements are classified as either functional or supplemental requirements.

# System requirements are:

- Functional requirements: how the system should react to a particular input and how the system should behave in a particular situation.
- Non-functional requirements:
  1. Constraint on the services or function offered by the system such as timing.
  2. Often apply to the system as whole rather than individual features or services.
- Domain requirements: constraints on the system from the domain of operation.

# Functional requirements:

- Describe functionality or system services.
- Depend on the type of software, expected user and the type of the system where the software is used.
- Functional user requirement's may be hight_level statements of what the system should do but functional system requirements should describe system services in details.

# Example of functional requirements

- The library system
- A library system that provides a single interface to a number of databases of articles in different libraries.
- User can search for download and print these articles for personal study.
- The user shall be able to search either all of the initial set of database or select a subset from it.
- The system shall provide appropriate viewers for the user to read document in the document store.
- Every order shall be allocated a unique identifier (ORDER-ID) which the user should be able to copy to the account's permanent area.

# Requirements imprecision

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the terms ‘ appropriate viewers
- 1. User intention – special purpose viewer for each different document type.
- 2. Developer interpretation – provide a text viewer that shows the contents of the document.
- Requirement completeness and consistency
- In principle, requirements should be both complete and consistent.
- Complete, they should include description of all facilities required.
- Consistent, they should be no conflicts or contradiction in the description of the system facilities.
- In practice, it is impossible to produce a complete and consistent requirements document.

# Nonfunctional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements.
- Process requirements may also be specified a particular CASE system, programing language or developments method.
- Nonfunctional requirements may be more critical than the functional requirements. If these are not met the system is useless.

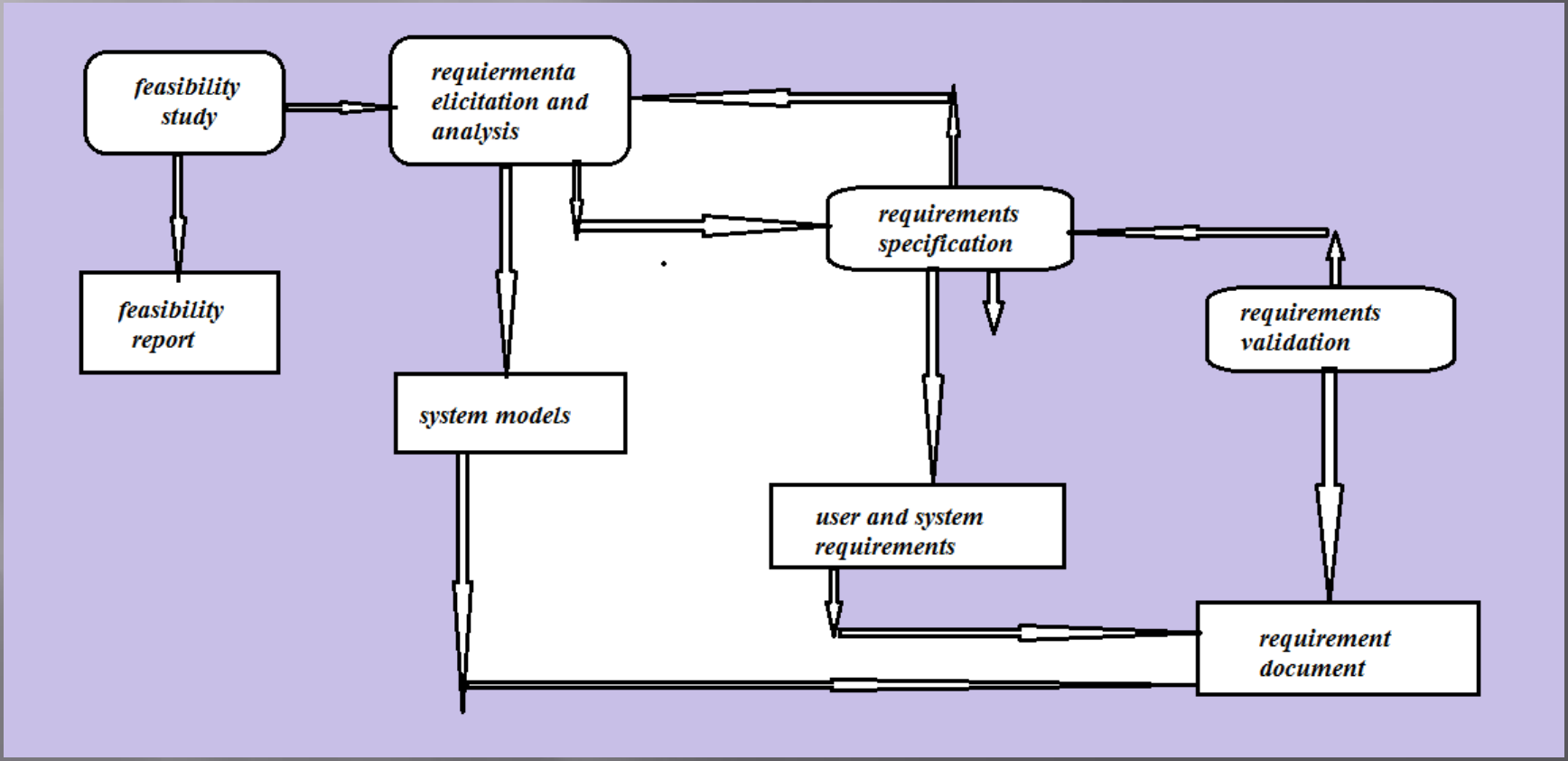
Should describe functional and nonfunctional requirements in such a way that they are understandable by the developer of the high level development.

User requirement are defined using natural language ,table and diagrams as these can be understandable by all users.

# Problems with natural language

- Lack of clarity (document are difficult to read).
- Requirements confusion: the requirements may be mixed-up.
- Requirements amalgamations: several requirements may be expressed together.

# The requirement passes thru a number of steps before designing the database



# DATABASE DESIGN

**Database design** is the process of producing a detailed data model of database. This data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

he term database design can be used to describe many different parts of the design of an overall database system. Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the database management system (DBMS).

# Entities

- The types of information that are saved in the database are called 'entities'. These entities exist in four kinds: people, things, events, and locations. Everything you could want to put in a database fits into one of these categories. If the information you want to include doesn't fit into these categories, than it is probably not an entity but a property of an entity, an attribute.
- To clarify the information given in this article we'll use an example. Imagine that you are creating a website for a shop, what kind of information do you have to deal with? In a shop you sell your products to customers. The "Shop" is a location; "Sale" is an event; "Products" are things; and "Customers" are people. These are all entities that need to be included in your database.

But what other things are happening when selling a product? A customer comes into the shop, approaches the vendor, asks a question and gets an answer. "Vendors" also participate, and because vendors are people, we need a vendors entity.

# Identifying Relationships

- The next step is to determine the relationships between the entities and to determine the cardinality of each relationship. The relationship is the connection between the entities, just like in the real world: what does one entity do with the other, how do they relate to each other? For example, customers buy products, products are sold to customers, a sale comprises products, a sale happens in a shop.
- The cardinality shows how much of one side of the relationship belongs to how much of the other side of the relationship. First, you need to state for each relationship, how much of one side belongs to exactly 1 of the other side. For example: How many customers belong to 1 sale?; How many sales belong to 1 customer?; How many sales take place in 1 shop?

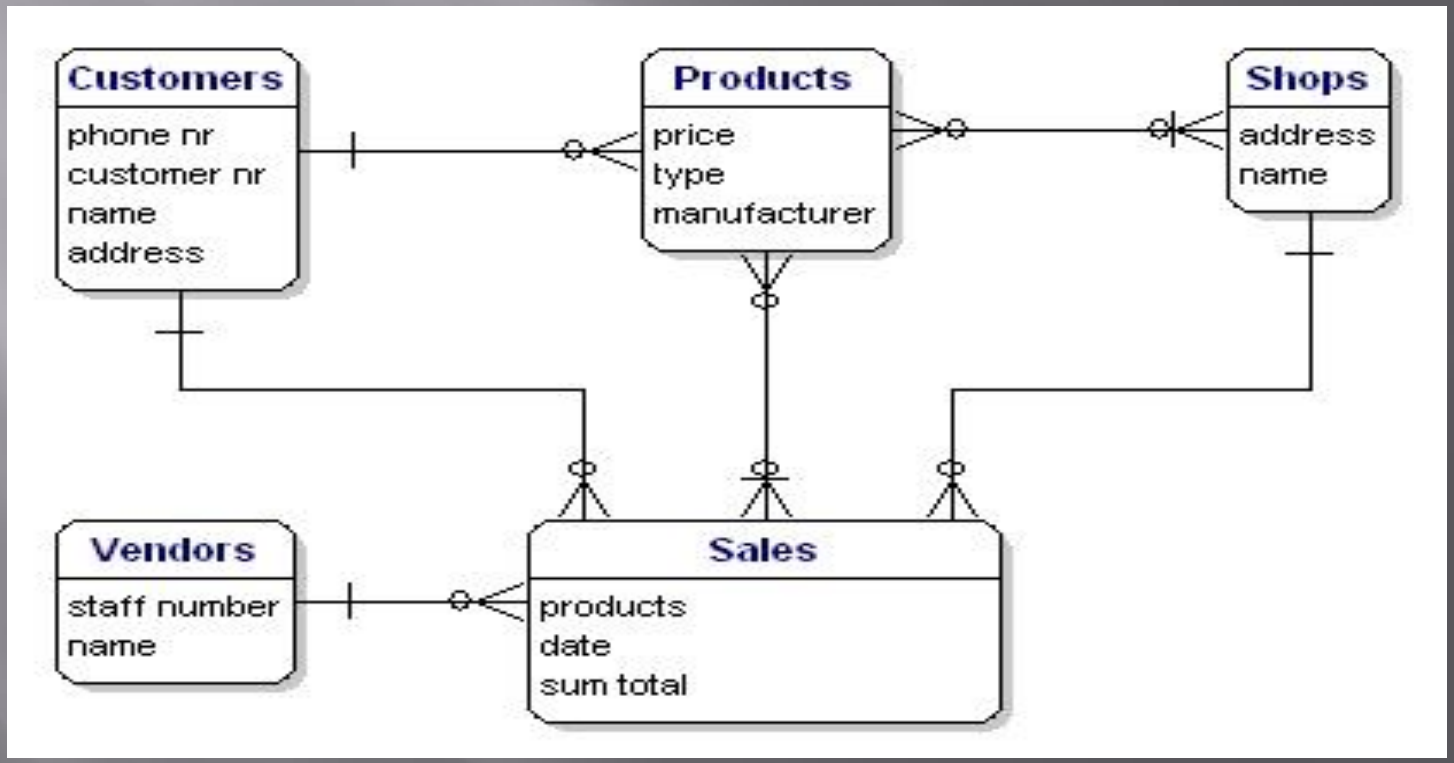
# Relationships

- Customers --> Sales; 1 customer can buy something several times
- Sales --> Customers; 1 sale is always made by 1 customer at the time
- Customers --> Products; 1 customer can buy multiple products
- Products --> Customers; 1 product can be purchased by multiple customers
- Customers --> Shops; 1 customer can purchase in multiple shops
- Shops --> Customers, 1 shop can receive multiple customers
- Shops --> Products; in 1 shop there are multiple products
- Products --> Shops; 1 product (type) can be sold in multiple shops
- Shops --> Sales; in 1 shop multiple sales can me made
- Sales --> Shops; 1 sale can only be made in 1 shop at the time
- Products --> Sales; 1 product (type) can be purchased in multiple sales
- Sales --> Products; 1 sale can exist out of multiple products

# If we shorthand The relationships, we'll get:

- Customers --> Sales; --> 1:N
- Customers --> Products; --> M:N
- Customers --> Shops; --> M:N
- Sales --> Products; --> M:N
- Shops --> Sales; --> 1:N
- Shops --> Products; --> M:N

# So, we have two '1-to-many' relationships, and four 'many-to-many' relationships.



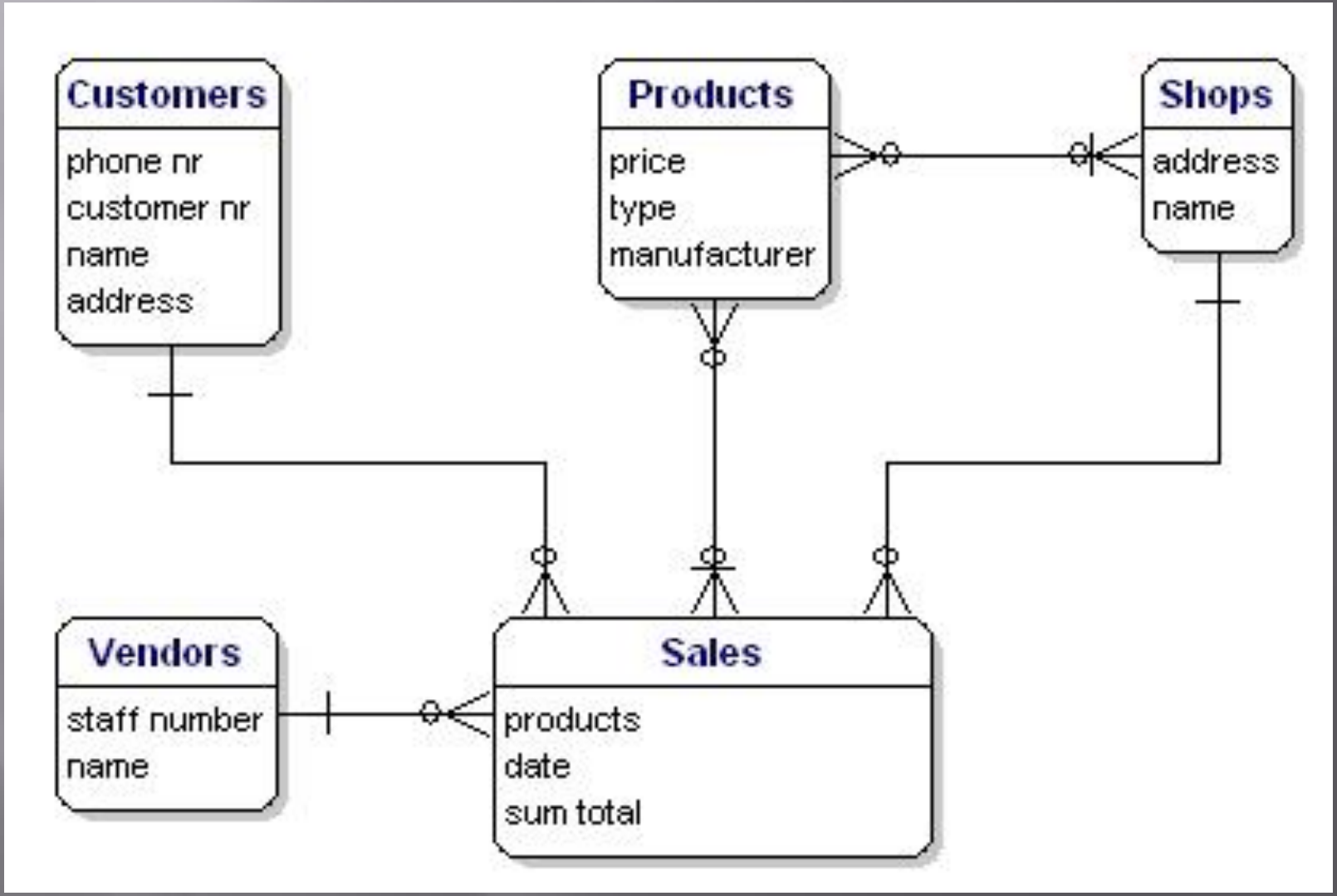
# Recursive Relationships

- Sometimes an entity refers back to itself. For example, think of a work hierarchy: an employee has a boss; and the boss chef is an employee too. The attribute 'boss' of the entity 'employees' refers back to the entity 'employees'.
- In an ERD (see next chapter) this type of relationship is a line that goes out of the entity and returns with a nice loop to the same entity.

# Redundant Relationships

- Sometimes in your model you will get a 'redundant relationship'. These are relationships that are already indicated by other relationships, although not directly.
- In the case of our example there is a direct relationships between customers and products. But there are also relationships from customers to sales and from sales to products, so indirectly there already is a relationship between customers and products through sales. The relationship 'Customers <----> Products' is made twice, and one of them is therefore redundant. In this case, products are only purchased through a sale, so the relationships 'Customers <----> Products' can be deleted.

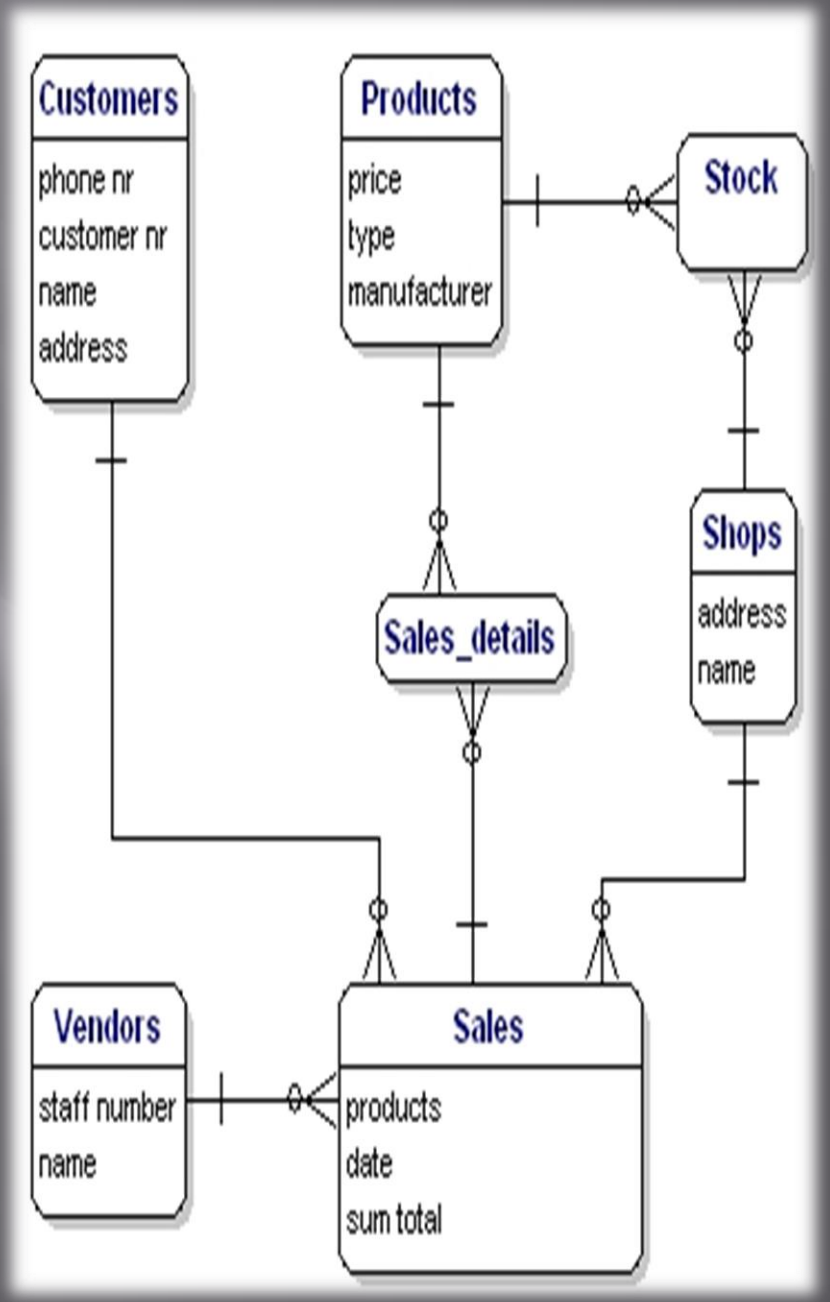
# The model will look like this:



# Solving Many-to-Many Relationships

- Many-to-many relationships (M:N) are not directly possible in a database. What a M:N relationship says is that a number of records from one table belongs to a number of records from another table. Somewhere you need to save which records these are and the solution is to split the relationship up in two one-to-many relationships.
- This can be done by creating a new entity that is in between the related entities. In our example, there is a many-to-many relationship between sales and products. This can be solved by creating a new entity: sales-products. This entity has a many-to-one relationship with Sales, and a many-to-one relationship with Products. In logical models this is called an associative entity and in physical database terms this is called a link table or junction table


- In the example there are two many-to-many relationships that need to be solved: 'Products <----> Sales', and 'Products <----> Shops'. For both situations there needs to be created a new entity, but what is that entity?
- For the Products <----> Sales relationship, every sale includes more products. The relationship shows the content of the sale. In other words, it gives details about the sale. So the entity is called 'Sales details'. You could also name it 'sold products'.
- The Products <----> Shops relationship shows which products are available in which the shops, also known as 'stock'. Our model would now look like this:



# Identifying Attributes

- The data elements that you want to save for each entity are called 'attributes'.
- About the products that you sell, you want to know, for example, what the price is, what the name of the manufacturer is, and what the type number is. About the customers you know their customer number, their name, and address. About the shops you know the location code, the name, the address. Of the sales you know when they happened, in which shop, what products were sold, and the sum total of the sale. Of the vendor you know his staff number, name, and address. What will be included precisely is not of importance yet; it is still only about what you want to save.

# Assigning Keys (Primary Keys , Foreign keys)

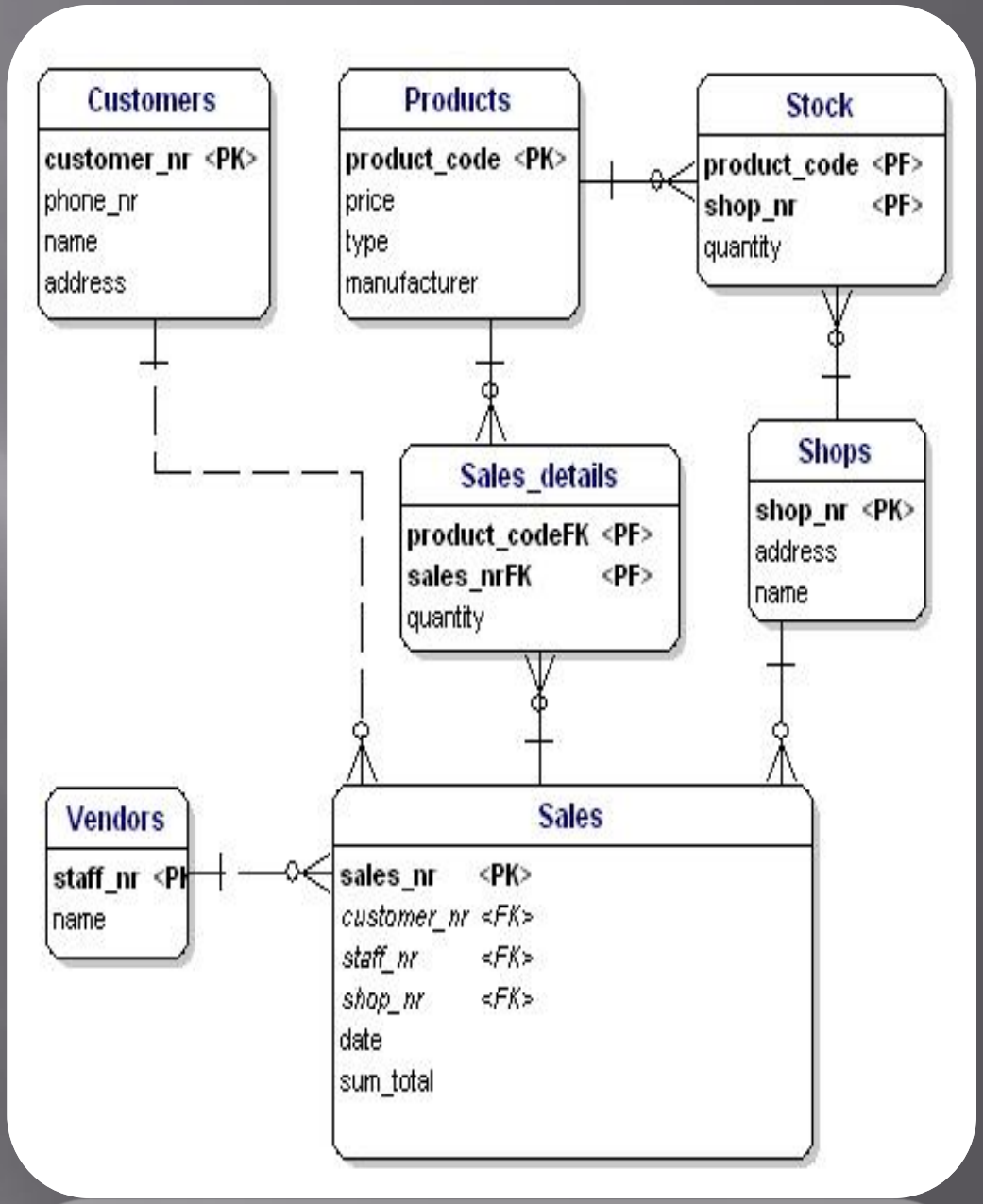
**Primary Keys**

- A primary key (PK) is one or more data attributes that uniquely identify an entity. A key that consists of two or more attributes is called a composite key. All attributes part of a primary key must have a value in every record (which cannot be left empty) and the combination of the values within these attributes must be unique in the table.
- In the example there are a few obvious candidates for the primary key. Customers all have a customer number, products all have a unique product number and the sales have a sales number. Each of these data is unique and each record will contain a value, so these attributes can be a primary key. Often an integer column is used for the primary key so a record can be easily found through its number.

# Foreign Keys

The Foreign Key (FK) in an entity is the reference to the primary key of another entity. In the ERD that attribute will be indicated with 'FK' behind its name. The foreign key of an entity can also be part of the primary key, in that case the attribute will be indicated with 'PF' behind its name. This is usually the case with the link-entities, because you usually link two instances only once together (with 1 sale only 1 product type is sold 1 time).

If we put all link-entities, PK's and FK's into the ERD, we get the model as shown



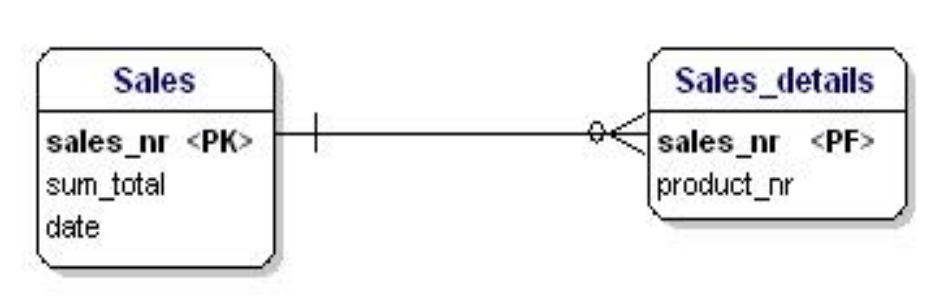
# Normalization

- Normalization makes your data model flexible and reliable. It does generate some overhead because you usually get more tables, but it enables you to do many things with your data model without having to adjust it.

# Normalization, the First Form

- The first form of normalization states that there may be no repeating groups of columns in an entity. We could have created an entity 'sales' with attributes for each of the products that were bought. This would look like this:

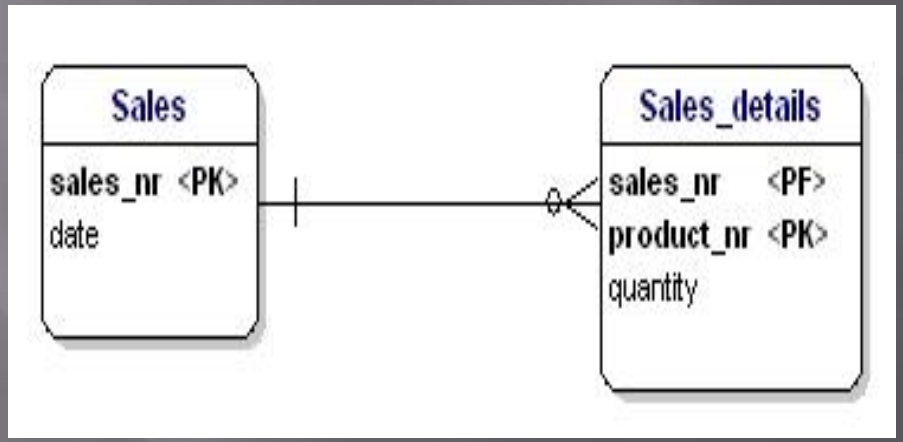
- What is wrong about this is that now only 3 products can be sold. If you would have to sell 4 products, than you would have to start a second sale or adjust your data model by adding 'product4' attributes. Both solutions are unwanted. In these cases you should always create a new entity that you link to the old one via a one-to-many relationship.



# Normalization, the Second Form

- The second form of normalization states that all attributes of an entity should be fully dependent on the whole primary key. This means that each attribute of an entity can only be identified through the whole primary key. Suppose we had the date in the Sales_details entity:

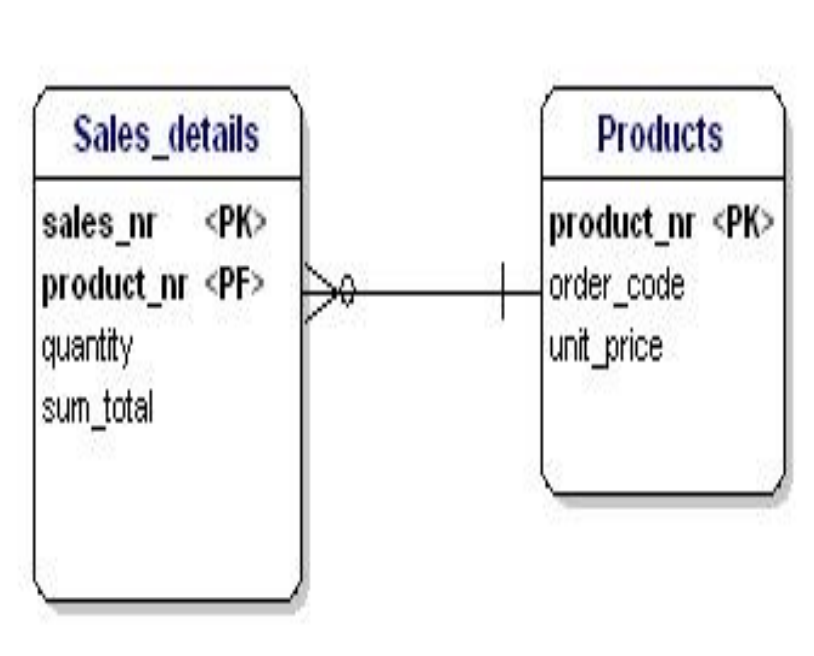
- This entity is not according the second normalization form, because in order to be able to look up the date of a sale, I do not have to know what is sold (product-nr), the only thing I need to know is the sales number. This was solved by splitting up the tables into the sales and the Sales details table:



# Normalization, the Third Form

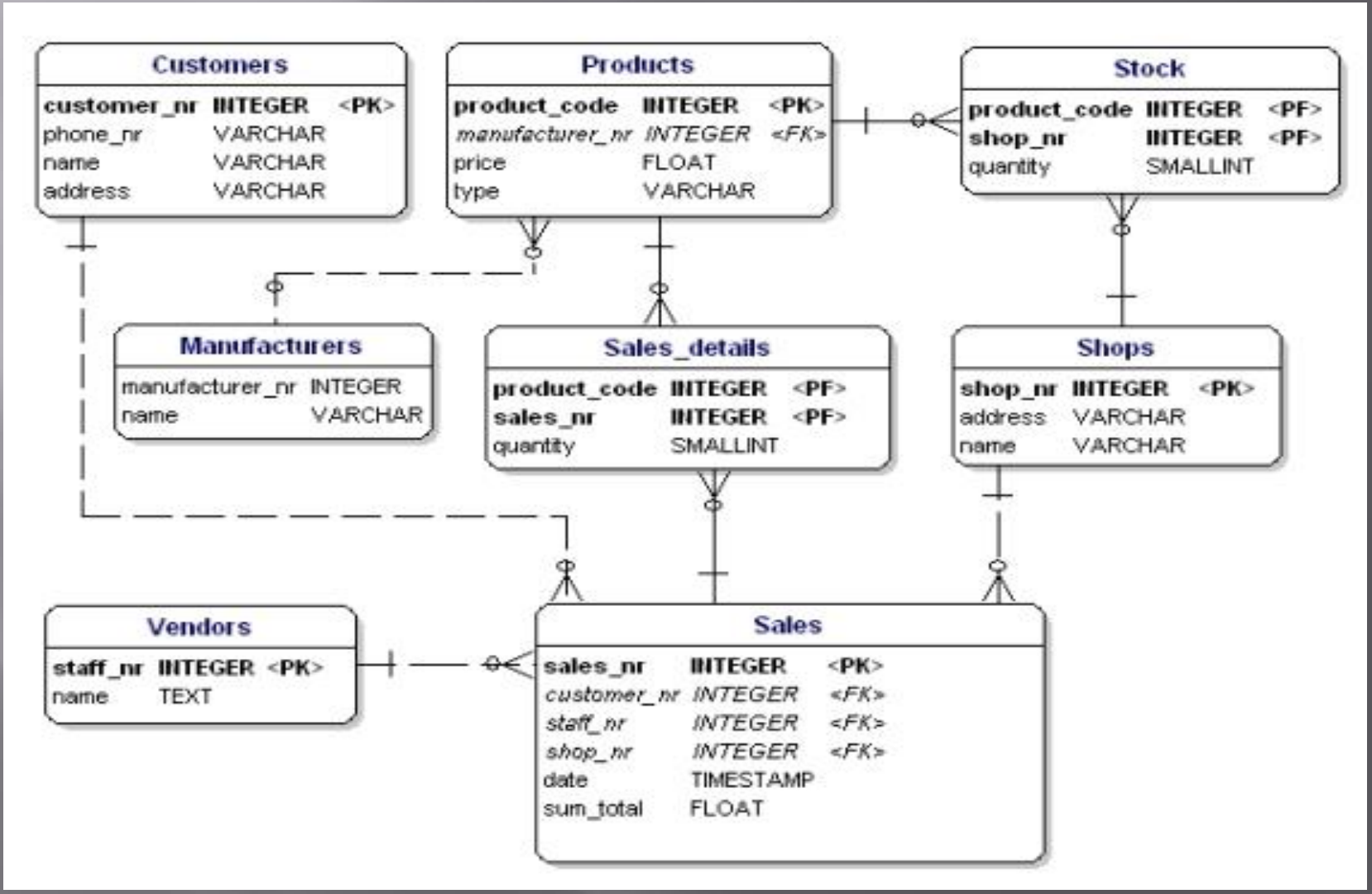
- The third form of normalization states that all attributes need to be directly dependent on the primary key, and not on other attributes. This seems to be what the second form of normalization states, but in the second form is actually stated the opposite. In the second form of normalization you point out attributes through the PK, in the third form of normalization every attribute needs to be dependent on the PK, and nothing else.

- In this case the price of a loose product is dependent on the ordering number, and the ordering number is dependent on the product number and the sales number. This is not according to the third form of normalization. Again, splitting up the tables solves this.



# Normalized Data Model

If you apply the normalization rules, you will find that the 'manufacturer' in de product table should also be a separate table:



# Glossary

- Attributes - detailed data about an entity, such as price, length, name.
- Cardinality - the relationship between two entities, in figures. For example, a person can place multiple orders.
- Entities - abstract data that you save in a database. For example: customers, products.
- Foreign key (FK) - a referral to the Primary Key of another table. Foreign Key-columns can only contain values that exist in the Primary Key column that they refer to.
- Key - a key is used to point out records. The most well-known key is the Primary Key (see Primary Key).
- Normalization - A flexible data model needs to follow certain rules. Applying these rules is called normalizing.
- Primary key - one or more columns within a table that together form a unique combination of values by which each record can be pointed out separately. For example: customer numbers, or the serial number of a product.