1.- What is a Database?

In essence a database is nothing more than a collection of information that exists over a long period of time, often many years.

2.- What is a DBMS?

A DBMS is characterized by the ability to support efficient access to large amounts of data, which persists over time. It is also characterized by support for powerful query languages and durable transactions that can execute concurrently in a manner that apears atomic and independent of other transaction.

2.1.- Early DBMS

The first commercial database management system appeared in 1960's. These systems evolved from file systems.

Some of applications of these systems was: Airline systems, Banking systems, Corporate records.

2.2.- DBMS

Ted Codd in 1970's changed significantly database systems. Codd proposed that database systems should present the user with a view of data organized as tables called relations.

SQL (Structured Query Language) is the most important query language based on relational model.

By 1990, relational database systems were the norm. Yet the database field continues to evolve, and the new issues and approaches to the management of data surface regularly.

2.3.- Outline of DBS studies

DESIGN OF DB:

1. How does one develop a useful DB?
2. What kind of information go into the database?
3. How is the information structured
4. What assumptions are made about types or values of data items?
5. How do the data items connect?

DATABASE PROGRAMMING:

1. How does one express queries and other operations on the DB?
2. How one use other capabilities of a DBMS such as transactions or constraints, in an application?
3. How is database programming combined with conventional progr.?

DATABASE SYSTEM IMPLEMENTATION:

1. How does one build a DBMS, including such matters as query processing, transaction processing and organizing storage for efficient access?

1.- Introduction to databases

Databases today are essential to every business. They are used to maintain internal records, to present data to customers and clients on the World-Wide-Web, and to support many other commercial processes.

Databases are likewise found at the core of many scientific investigations. They represent the data gathered by astronomers, by investigators of the human genome, and by biochemists exploring the medicinal properties of proteins, along with many other scientists.

2.- Databases and Databases Management Systems (DBMS)

2.1.- The DATA and DATABASE

There are three scopes or worlds of data:

- **The real world:** they are the objects of the reality, and whose will be manage.
- **The conceptual world:** Set knowledges obtained from the real world. These sets depends on the viewer.
- **The representations world**: it is the set of computer representations of the world. They are required to work with data.

2.2.- Data and its representation.

DATA are computer representation of available information. It makes reference to interesting for us real world objects. REPRESENTATIONS WORLD is set up by the worked computerized data.

Two stages are necessaries in order to convert the real world to computer data; they are:

1. **Logic Design:** It works with the abstract model of data. That model is obtained at the end of the conceptual model stage. The aim of this stage is translate the real world to the data model used by DBMS.
2. **Physical Design:** It is the stage where we increase the operation efficiency.

2.3.- Entity, attribute and value.

The information is defined by three elements:

1. **Entity:** The ENTITY word is some object of the reality conceptualised and we are interested in some characteristics of it. There are two entities: Entity type is a generic word used in the real language and Entity request is a specific object. For example: car is an Entity type. Your car with ID is 1234 BCD is an Entity Request.
2. **Attribute:** entity properties which are interested. Example: the car entity 1234 BCD is type, color, glass, hp, cc.
3. **Value:** are the values of the attributes. Example: the car entity 1234 BCD is sportive, black, dark, 147hp, 3.5 cc.

About ENTITY there are two types:

- **Type entity:** is a generic entity or, more correctly, is an abstraction of a real things set. F.i.: a car
- **Instance entity:** is a real world object conceptualization. F.i.: the car licence plate

3.- Attributes, data type and domain

All the value set that an attribute can take is called domain.

A data type define a value set with some common characteristics, these characteristics make compatibilities, then operations about that could be defined.

Example of type of data

Let integer set be as a type of data. Operation like plus, minus, times can be defined but the exact division can't be defined on this set.

Example of type of domain

You consider the spanish licence driver (DL). When you lose all the points, the DL is revoked and you must pass the exam another time. The number of points is from 0 to 15, this is the domain in this case.

4.- Null value

The expression null value indicate that any value is associated to that attribute of an entity. For instance, a person who don't have the DL, doesn't have points, neither 0.

5.- Keys and ID attributes

An identificator attribute (ID) permits identify unmistakably an entity from the rest. It value is unique. For instance, the number of ID card is unique for everyone.

The set of ID attributes is called keys. For example: The postal code is a key on shopping of department stores.

6.- Tabular representations and its implementation

The information is the result of analyze and of make concepts from the real world. There are representations which aren't efficient to see the information, and it is necessary use a good representation. The most used is the tabular representation.

On the tabular representation every row represents an instance entity, every column represents an attribute and every cell contains the value pertinent.

Most of these features, however, depend at least in part on good database design. If you don't craft a good design, you'll miss out on some or all of the benefit of these features.

The following describe some of the features that a good database system should provide and explain to what degree they depend on good database design.

1.- CRUD

CRUD stands for the four fundamental database operations that any database should provide: Create, Read, Update, and Delete.

CRUD is more a feature of databases in general than it is a feature of good database design, but a good database design provides CRUD efficiently.

When you create a new record (the C in CRUD), the database must validate the new State entry. Similarly when you update a record (the U in CRUD), the database must validate the modified State entry.

When you delete an entry in the States table (the D in CRUD), the database must verify that no Participant records use that state. Finally when you read data (the R in CRUD), the database design determines whether you find the data you want in seconds, hours, or not at all.

2.- Retrieval

Retrieval is another word for ''read,'' the R in CRUD. The database should allow you to find every piece of data. There's no point putting something in the database if there's no way to get it back later. (That would be a ''data black hole,'' not a database.)

The database should allow you to structure the data so you can find particular pieces of data in one or more specific ways.

Ideally the database will also allow you to structure the data so it is relatively quick and easy to fetch data in a particular manner.

In contrast, you probably don't need to search for customers by middle name too frequently.

3.- Consistency

Another aspect of the R in CRUD is consistency. The database should provide consistent results. If you perform the same search twice in a row, you should get the same results. Another user who performs the same search should also get the same results.

A well-built database product can ensure that the exact same query returns the same result but design also plays an important role. If the database is poorly designed, you may be able to store conflicting data in different parts of the database.

4.- Validity

Validity is closely related to the idea of consistency. Consistency means different parts of the database don't hold contradictory views of the same information. Validity means data is validated where possible against other pieces of data in the database. In CRUD terms, data can be validated when a record is created, updated, or deleted.

Just like physical data containers, a computerized database can hold incomplete, incorrect, or contradictory data. You can never protect a database from users who can't spell or who just plain enter the wrong information, but a good database design can help prevent some kinds of errors that a physical database cannot prevent.

The database can also verify that a value entered by the user is present in another part of the database.

The database can also check some kinds of conditions on the data.

5.- Easy error connection

Even a perfectly designed database cannot ensure perfect validity. How can the database know that a customer's name is supposed to be spelled Pheidaux not Fido as typed by the user?

Correcting a single error in a notebook is fairly easy. Just cross out the wrong value and write in the new one.

Correcting systematic errors in a notebook is a lot harder. Suppose you hire a summer intern to go door-to-door selling household products and he writes up a lot of orders for ''Duck Tape'' not realizing that the actual product is ''Duct Tape.'' Fixing all of the mistakes could be tedious and time-consuming.

In a computerized database, this sort of correction is trivial. A simple database command can update every occurrence of the product name ''Duck Tape'' throughout the whole system.

6.- Speed

An important aspect of all of the CRUD components is speed. A well-designed database can create, read, update, and delete records quickly.

There's no denying that a computerized database is a lot faster than a notebook or a filing cabinet. Instead of processing dozens of records per hour, a computerized database can process dozens or hundreds per second.

Good design plays a critical role in database efficiency. A poorly organized database may still be faster than the paper equivalent but it will be a lot slower than a well-designed database.

7.- Atomic transactions

Recall that an atomic transaction is a possibly complex series of actions that is considered as a single operation by those not involved directly in performing the transaction. If you transfer $100 from Alice's account to Bob's account, no one else can see the database while it is in an intermediate state where the money has been removed from Alice's account and not yet added to Bob's.

The transaction either happens completely or none of its pieces happen — it cannot happen halfway.

Atomic transactions are important for maintaining consistency and validity, and are thus important for the R and U parts of CRUD.

Physical data containers such as notebooks support atomic transactions because typically only one person at a time can use them.

These databases also automatically rollback any transaction that is open if the database halts unexpectedly.

8.- ACID

This section provides some more detail about the transactions described in the previous section rather than discussing a new feature of physical data containers and computerized databases.

ACID is an acronym describing four features that an effective transaction system should provide. ACID stands for Atomicity, Consistency, Isolation, and Durability. Atomicity means transactions are atomic. The operations in a transaction either all happen or none of them happen.

Consistency means the transaction ensures that the database is in a consistent state before and after the transaction. In other words, if the operations within the transaction would violate the database's rules, the transaction is rolled back.

Isolation means the transaction isolates the details of the transaction from everyone except the person making the transaction.

Durability means that once a transaction is committed, it will not disappear later. If the power fails, when the database restarts, the effects of this transaction will still be there.

9.- Persistence backups

The data must be persistent. It shouldn't change or disappear by itself. If you can't trust the database to keep the data safe, the database is pretty much worthless.

Database products do their best to keep the data safe, and in normal operation you don't need to do much to get the benefit of data persistence.

10.- Low cost & extensibility

Ideally the database should be easy to obtain and install, inexpensive, and easily extensible. If you discover that you need to process a lot more data per day than you had expected, you should be able to somehow increase the database's capacity.

Although some database products are quite expensive, most of them have reasonable upgrade paths so you can buy the least expensive license that will handle your needs, at least in the beginning.

11.- Easy of use

Notebooks and filing cabinets have simple user interfaces so almost anyone can use them effectively.

A computer application's user interface determines how usable it is by average users. User interface design is not part of database design, so you may wonder why ease of use is mentioned here.

12.- Portability

A computerized database allows for a portability that is even more powerful than the portability of a notebook. It allows you to access the data from anywhere you have access to the Web without actually moving the physical database.

You can access the database from just about anywhere while the data itself remains safely at home, far from the dangers of pickpockets, being dropped in a puddle, and getting forgotten on the bus.

13.- Security

A notebook is relatively easy to lose or steal but a highly portable database can be even easier to compromise. If you can access your database from all over the world, then so can cyber banditos and other ne'er-do-wells.

Locking down your database is mostly a security issue that you should address by using your network's and database's security tools. However, there are some design techniques that you can use to make securing the database easier.

If you separate the data into categories that different types of users need to manipulate, you can grant different levels of permission to the different kinds of users.

Yet another novel aspect to database security is the fact that users can access the database remotely without actually holding a copy of the database locally.

This is more an application architecture issue than a database design issue (don't store the data locally on laptops) but using a database design that restricts users' access to what they really need to know can help.

14.- Sharing

It's not easy to share a notebook or envelope full of business cards among a lot of people. No two people can really use a notebook at the same time and there's some overhead in shipping the notebook back and forth among users.

Taking time to walk across the room a dozen times a day would be annoying; express mailing a notebook across the country every day would be just plain silly.

Modern networks can let hundreds or even thousands of users access the same database at the same time from locations scattered across the globe. Though this is largely an exercise in networking and the tools provided by a particular database product, some design issues come into play.

15.- Ability to perform complex calculations

Compared to the human brain, computers are idiots. It takes seriously powerful hardware and frighteningly sophisticated algorithms to perform tasks that you take for granted such as recognizing faces, speaker-independent speech recognition, and handwriting recognition.

The human brain is also self-programming, so it can learn new tasks flexibly and relatively quickly.

When it comes to balancing checkbooks, searching for accounts with balances less than zero, and performing a host of other number crunching tasks, the computer is much faster.

The computer is naturally faster at these sorts of calculations, but even its blazing speed won't help you if your database is poorly designed. A good design can make the difference between finding the data you need in seconds rather than hours, days, or not at all.

1.- Relational database

Without getting into too much detail, a relational-database contains tables that hold rows and columns. Each row holds related data about a particular entity (person, vehicle, sandwich, or whatever). Each column represents a piece of data about that entity (name, street address, number of pickles, and so forth).

Sometimes a piece of data naturally has more than one value. For example, a single customer might place lots of orders. To make it easy to add multiple values, those values are stored in a separate table linked to the first by some value that the corresponding records share.

For example, suppose you build a relational database to track your favorite street luge racers. The Racers table stores information about individual racers. Each row corresponds to a particular racer. The columns represent basic information for a racer such as name, age, height, weight, and so forth. A very important column stores each racer's ID number.

Over time, each racer will have lots of race results (although there will probably be lots of blank spots for bale chuckers — see www.skateluge.com/lugetalk.htm). You store race results in a separate RaceResults table. Each row records the final standings for a single racer in a single race. The columns record the racer's ID number, the race's name and date, the racer's finishing position, and the points that position is worth for overall ranking.

To find all of the finishing positions and points for a particular racer, you look up the racer's row in the Racers table, find the racer's ID number, and then find all of the rows in the RaceResults table that have this racer ID.

Racers Table

RacerName	Nationality	RacerId
Michael Serek	Austria	1
Chris McBride	United States	2
Sebastien Tournissac	France	3

RaceResults Table

RaceName	Division	Dates	RacerId	FinishingPosition	Points
Go Fast Speed Days	Pro Classic Luge Mass	9/1/2007-9/2/2007	1	1	450.0024
Rock and Roll	Pro Classic Luge Mass	7/27/2007-7/28/2007	1	1	450.0024
Almabtrieb World Championships	Pro Classic Luge Mass	7/11/2007-7/14/2007	1	6	403.3633
Almabtrieb World Championships	Pro Classic Luge Mass	7/11/2007-7/14/2007	2	24	321.1366
Almabtrieb World Championships	Pro Classic Luge Mass	7/11/2007-7/14/2007	3	2	432.6154
Go Fast Speed Days	Pro Classic Luge Mass	9/1/2007-9/2/2007	2	2	432.6154
Go Fast Speed Days	Pro Classic Luge Mass	9/1/2007-9/2/2007	3	3	424.3687
Top Challenge	Pro Street Luge Mass	8/25/2007-8/26/2007	2	13	0

Relational databases have been around for a long time. (Edgar Codd started laying the foundations in 1970.) They are the most commonly used kind of database today and have been for years, so a lot of very powerful companies have spent a huge amount of time building them. All of that means that relational databases have been thoroughly studied and have evolved over time to the point where they are quite useful and effective.

Relational databases provide a number of features that make working with databases such as the street luge database easier. Some of the features they provide include:

- **Data types:** Each column has a particular data type (text, numeric, date, and so forth) and the database will not allow values of other types in a column.

- **Basic constraints:** The database can enforce constraints such as requiring that a luge racer's top speed be between 50 and 250mph (no one with a top speed less than 50 is worth recording) or it can require certain fields.
- **Referential integrity:** The database can prevent you from adding a RaceResults record for a racer who doesn't exist in the Racers table. Similarly, the database can prevent you from modifying a racer's ID if that would leave rows in the RaceResults table with invalid racer IDs, and it could prevent you from modifying a RaceResults row's racer ID to an invalid value.
- **Cascading deletes and updates:** If you delete a racer from the Racers table, the database can automatically delete all of that racer's RaceResults records. Similarly if you change a racer's ID number, the database can update the ID numbers in that racer's RaceResults records.
- **Joins:** The database can quickly gather related records from different tables. For example, it can easily list every racer with his or her corresponding finishing positions sorted alphabetically and by race date.
- **Complex queries:** Relational databases support all sorts of interesting query and aggregation functions such as SUM, AVG, MIN, COUNT, STDEV, and GROUP BY.

Relational databases work well if:

- You need to perform complicated queries and joins among different tables.
- You need to perform data validations such as verifying that related rows in other tables exist.
- You need to allow for any number of values for a particular piece of data (for example, race finishing positions).
- You want to be able to flexibly build new queries that you didn't plan when you started designing the project.

Relational databases don't work well if:

- You need to use a special data topology to perform the application's main function. For example, you can beat a hierarchy or network with a brick until it fits in a relational database but you may get better performance using a more specialized type of database.

Unless you have special needs, relational databases are usually an excellent choice.

2.- XML

XML (eXtensible Markup Language) is a language for storing hierarchical data. XML itself doesn't provide any tools for building, searching, updating, validating, or otherwise manipulating data and anyonewho tells you otherwise is trying to sell you something.

XML Basics

An XML file is a relatively simple text file that uses special tokens to define a structure for the data that it contains. People often compare XML to the Web language HTML (HyperText Markup Language) because both use tokens surrounded by pointy brackets, but the two languages have several large differences.

XML Structures

In practice I typically see XML files used most often in one of three ways.

First, XML files are hierarchical so it's natural to use them to hold hierarchical data. It's straightforward to map purely hierarchical data such as a simple family tree or organizational chart into an XML file.

Second, XML files are often used to hold table-like data. The basic structure closely follows the structure of a relational database. The root element holds several table elements. Each of those elements holds ‘‘records’’ that hold ‘‘fields.’

For example, the following XML document holds data about a simple company’s customers and their orders:

```
<AllData>
	<Customers>
		<Customer ID="1">
			<FirstName>Alfred</FirstName>
			<LastName>Gusenbauer</LastName>
		</Customer>
		<Customer ID="2">
			<FirstName>David</FirstName>
			<LastName>Thompson</LastName>
		</Customer>
		<Customer ID="3">
			<FirstName>Alberto</FirstName>
			<LastName>Selva</LastName>
		</Customer>
	</Customers>
	<Products>
		<Product ID="273645" Description="Toothbrush" Price="$1.95" />
		<Product ID="78463" Description="Pencil" Price="$0.15" />
		<Product ID="48937" Description="Notepad" Price="$0.75" />
	</Products>
	<CustomerOrders>
		<CustomerOrder Date="12/27/2008" CustomerId="2">
			<Item ID="1" ProductId="78463" Quantity="12" />
			<Item ID="2" ProductId="48937" Quantity="2" />
		</CustomerOrder>
	</CustomerOrders>
</AllData>
```

The file starts with an AllData root element. That element contains three more elements that define table-like structures holding customer, product, and customer order information.

Each of these ‘‘tables’’ defines ‘‘records.’’ For example, the Customers element includes Customer ‘‘records’’ that hold FirstName and LastName values.

This XML document uses ID numbers to link records in different ‘‘tables’’ together. In this example, the single CustomerOrder element represents an order placed by customer 2 (David Thompson) who ordered 12 items with ID 78463 (pencils) and 2 items with ID 48937 (notepads).

The third XML file structure I’ve seen regularly is a simple list of values. The following XML document uses this structure to hold configuration settings for an application:

```
<Settings>
	<NormalColor>Black</NormalColor>
	<WarningColor>Green</WarningColor>
	<ErrorColor>Yellow</ErrorColor>
	<PanicSound>panic.wav</PanicSound>
	<BugEmail>bugs@panic.com</BugEmail>
</Settings>
```

This kind of XML file gives a little more structure than a flat text file used to hold settings and lets a program use XML tools to easily load and read setting values.

XML files work well if:

1. The data is naturally hierarchical.
2. Available XML tools provide the features you need.
3. You want the kinds of validation that schema files can provide.

4. You want to import and export the data in products that understand XML.

XML files don't work well if:

1. You use non-hierarchical data such as networks (described in the following section).
2. You need more complex data validation than schema files can provide.
3. You need to perform relational rather than hierarchical queries.
4. The database is very large so rewriting the entire file to update a small bit of data in the middle is cumbersome.
5. You need to allow multiple users to frequently update the database without interfering with each other.

You can find lots of free tutorials covering XML and its related technologies such as XSL, XSLT, XPath, XQuery, and others on the W3 Schools Web site (www.w3schools.com).

3.- Object-relational

An object-relational database (ORD) or object-relational database management system (ORDBMS) is a relational database that provides extra features for integrating object types into the data. Like a relational database, it can perform complex queries relatively quickly. Like an object database, it uses some special syntax to simplify the creation of objects.

Over time, many of the features originally designed for use by object-relational databases have been added to relational databases.

Object-relational databases and object-relational mappings work well if:

1. Your programming environment and architecture favors using objects.
2. You need to perform complicated relational-style queries.
3. You need to perform relational-style data validations.
4. Your program needs to interact with external tools where storing the data in a common relational format is an advantage.
5. You have separate programmers and database developers so maintaining a strict separation can make the project more manageable.

Object-relational databases and object-relational mappings don't work well if:

- You aren't using an object-oriented language (for example, if a Microsoft Access database can do everything you need without any programming).