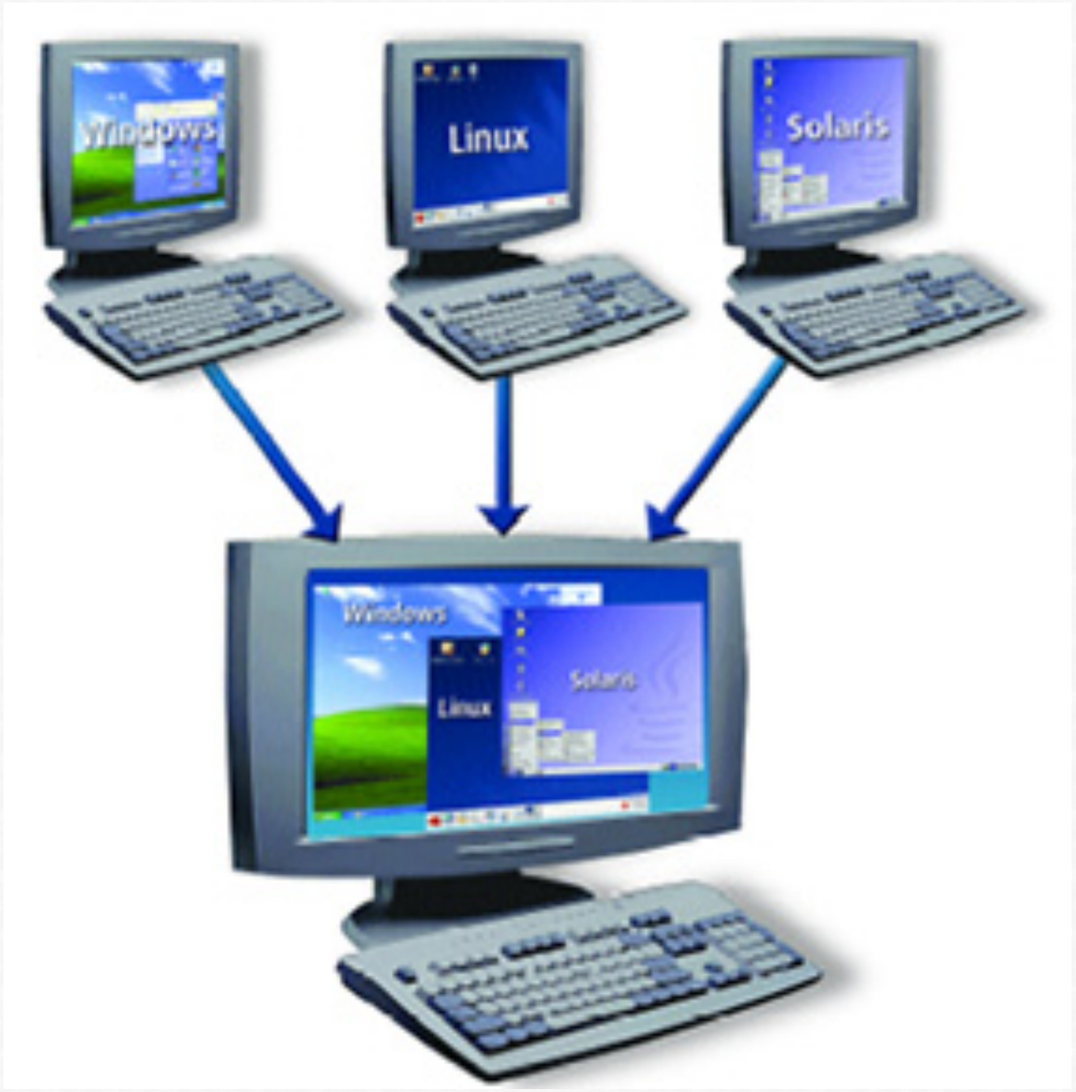
**UNIVERSITATEA „TRANSILVANIA” DIN BRAŞOV**
FACULTATEA DE INGINERIE ELECTRICĂ
ŞI ŞTIINŢA CALCULATOARELOR

# *Retele de senzori*

Curs 2 - 1st edition

# Virtual Machine – short introduction

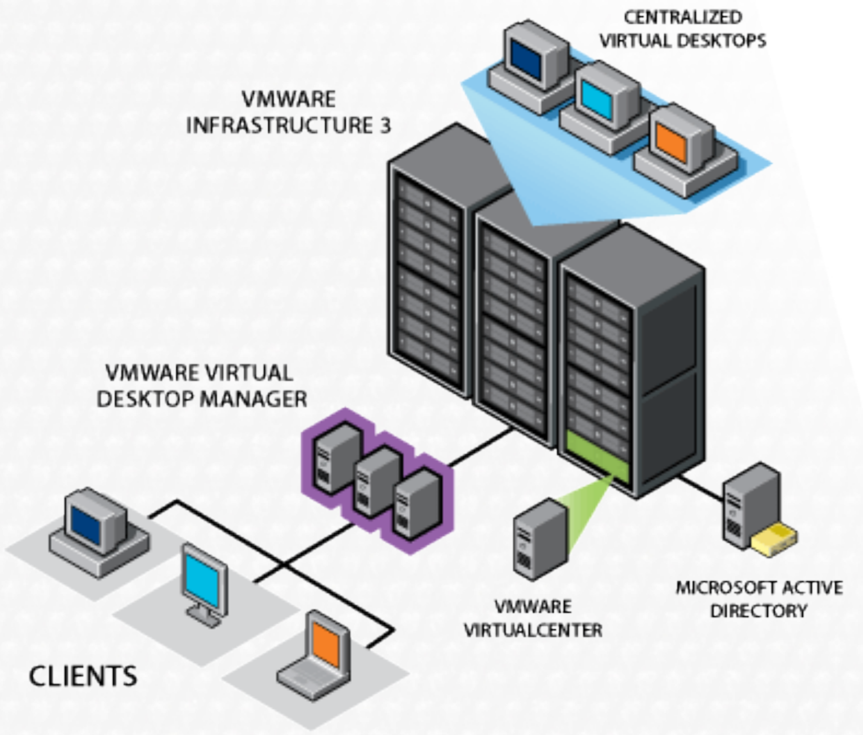


It’s like installing software applications on a real computer, but in this case that computer doesn’t exist. Anyway it could be executed only on a real computer (a virtual computer inside a real computer). On a real computer could be launched as many virtual machines as needed, and as soon there are resources (computing, memory) available.

## Virtual Machine – short introduction

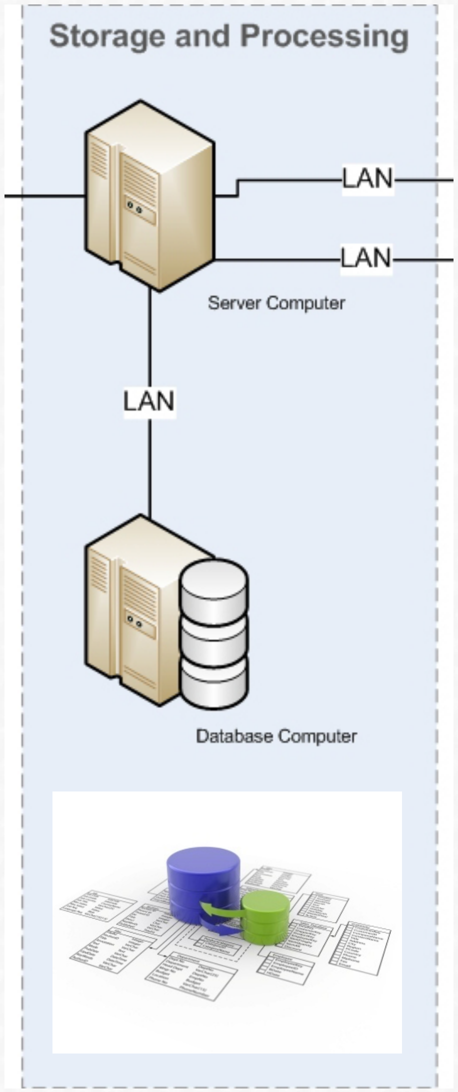
A virtual machine could be created with an application like *VMWare Workstation 7* and executed with *VMWare Player.*

- when it is initially created it is required to specify the operating system (OS) which will be installed on it and the path to the OS kit.
- after the OS is installed it could be installed on the virtual machine any applications the users desires (more applications installed, more space occupied by the virtual machine on the real computer's hard disk).
- then it could be easily moved or copied to another real computer and executed properly.


CENTRALIZED VIRTUAL DESKTOPS
VMWARE INFRASTRUCTURE 3
VMWARE VIRTUAL DESKTOP MANAGER
CLIENTS
VMWARE VIRTUALCENTER
MICROSOFT ACTIVE DIRECTORY

A virtual machine is represented on a real computer by a set of files and when it is executed, use, by sharing, the resources of the real computer.

# Database – role and components



The main role of a database it to store properly the information, ensure a fast processing and data integrity.

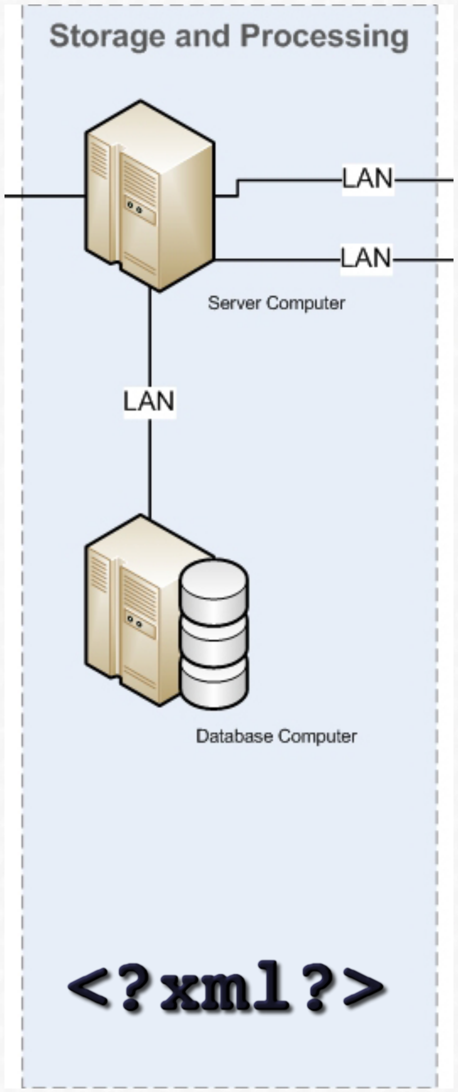
It could store different type of information:

- strings (e.g. “brasov”);
- integer numbers (e.g. 117);
- real numbers (e.g. 23.54);
- date & time (e.g. 27/11/2010 12:55:00);
- XML.

SQL (Structured Query Language) is a language defined for processing the information from a database. The main operation which could be executed with information are: INSERT, UPDATE, DELETE, SELECT.

e.g. SELECT * FROM users_table ORDER BY name.

## XML



*XML (eXtensible Markup Language)*

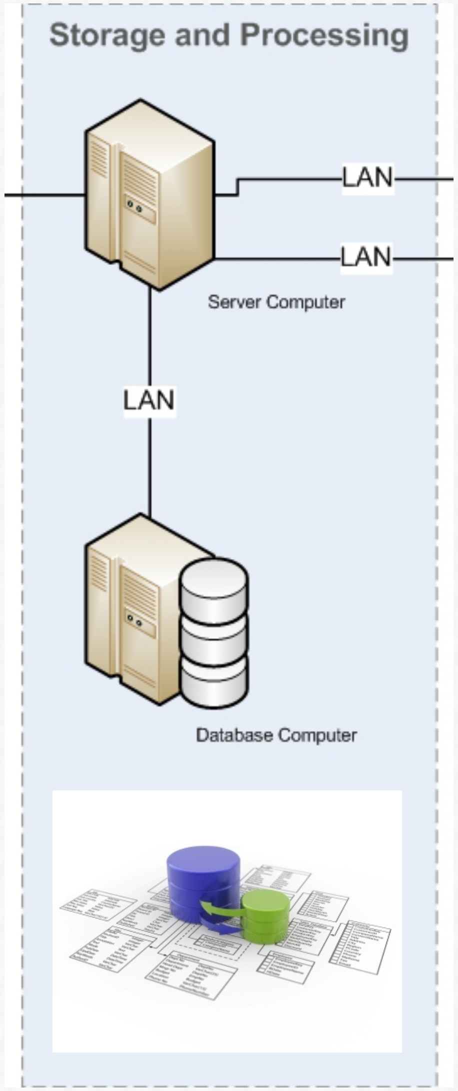
• it was designed for permitting the transfer of information between software applications and have a structured format and easily understandable.

• the useful information is posted between the tags or as a attribute for a tag.

e.g.

```
<note id=“1”>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

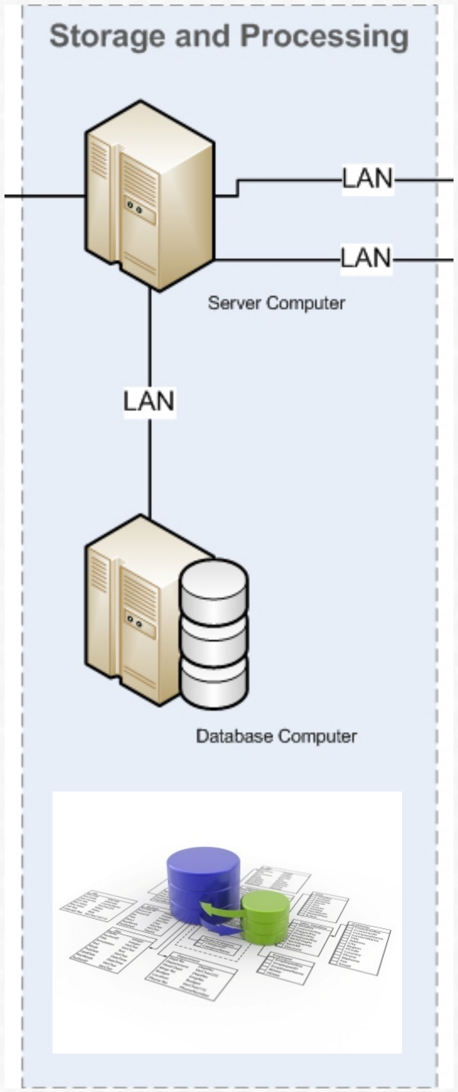
# Database – role and components



It permit to define inside it:

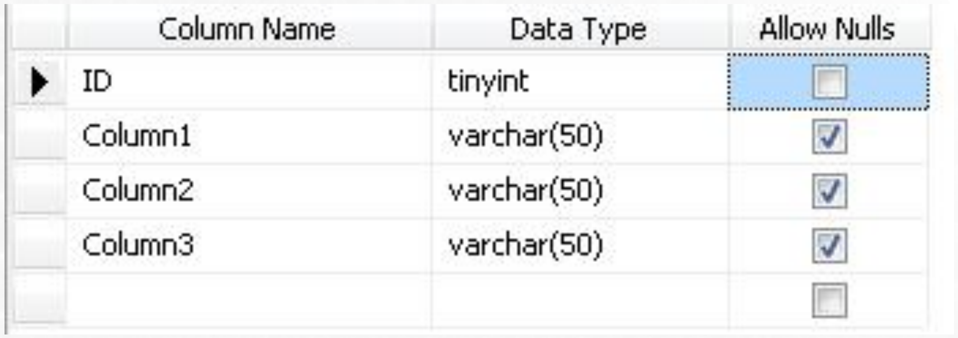
- tables (for a better organization of the information);
- diagrams (with relationships between tables);
- views (for a fast processing of the information);
- stored procedures and triggers (the stored procedures could be called when it is necessary to execute some processing; the triggers are executed on different types of events – insert, update, delete);
- security rules (who can access the database and which type of functionalities a user could call; there are defined specific users for the database).

## Database – tables



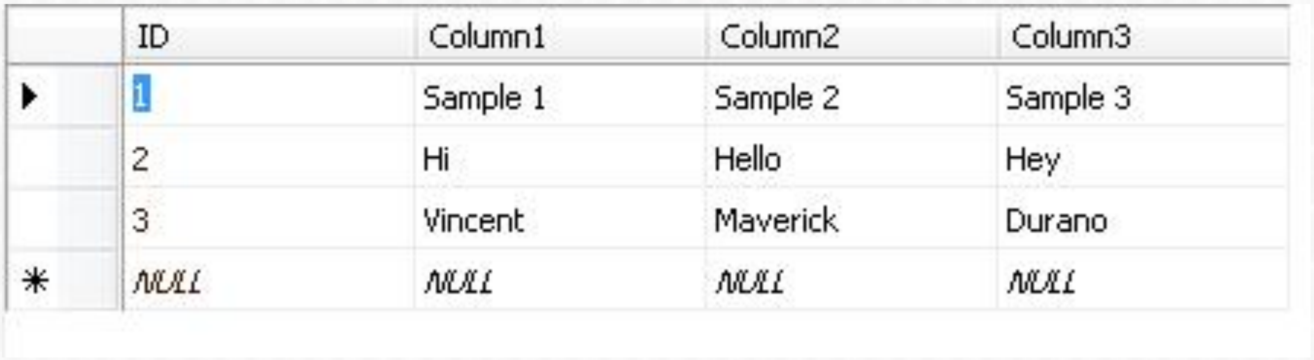
- each type of information is stored into a separate table.
- a table is defied as a set of columns (each column could be of different type).

Table definition



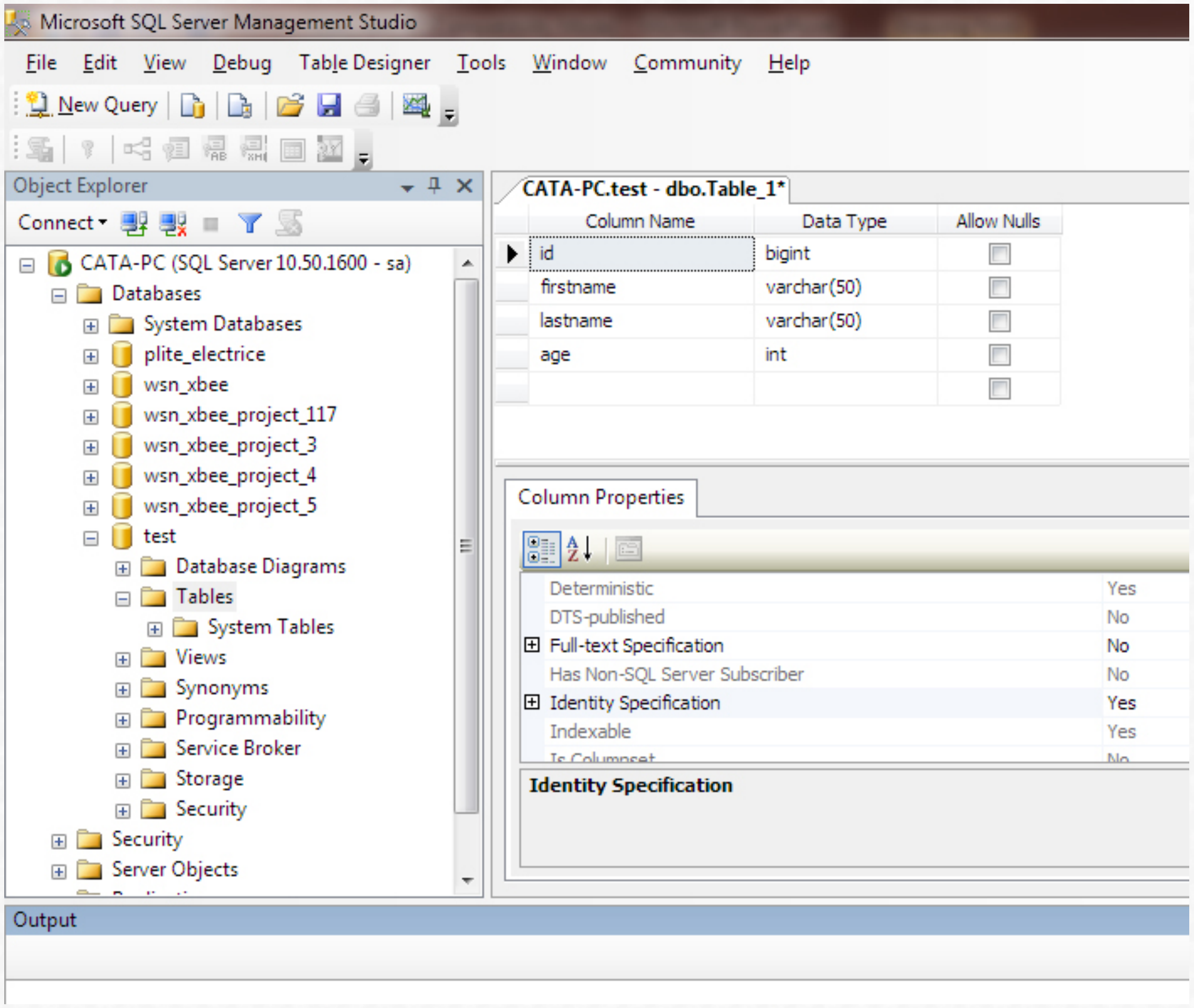
| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| ▶ | ID | tinyint | ☐ |
| | Column1 | varchar(50) | ✓ |
| | Column2 | varchar(50) | ✓ |
| | Column3 | varchar(50) | ✓ |
| | | | ☐ |

Table data



| | ID | Column1 | Column2 | Column3 |
|---|---|---|---|---|
| ▶ | 1 | Sample 1 | Sample 2 | Sample 3 |
| | 2 | Hi | Hello | Hey |
| | 3 | Vincent | Maverick | Durano |
| * | NULL | NULL | NULL | NULL |

# Laboratory



Microsoft SQL Server Management Studio

File Edit View Debug Table Designer Tools Window Community Help

Object Explorer

CATA-PC (SQL Server 10.50.1600 - sa)
- Databases
  - System Databases
  - plite_electrice
  - wsn_xbee
  - wsn_xbee_project_117
  - wsn_xbee_project_3
  - wsn_xbee_project_4
  - wsn_xbee_project_5
  - test
    - Database Diagrams
    - Tables
      - System Tables
    - Views
    - Synonyms
    - Programmability
    - Service Broker
    - Storage
    - Security
- Security
- Server Objects

CATA-PC.test - dbo.Table_1*

| Column Name | Data Type | Allow Nulls |
|---|---|---|
| id | bigint | ☐ |
| firstname | varchar(50) | ☐ |
| lastname | varchar(50) | ☐ |
| age | int | ☐ |
| | | ☐ |

Column Properties

| | |
|---|---|
| Deterministic | Yes |
| DTS-published | No |
| Full-text Specification | No |
| Has Non-SQL Server Subscriber | No |
| Identity Specification | Yes |
| Indexable | Yes |

Identity Specification

Output

• use the SQL Server Management Studio to create a database in MsSQL (the name of the database will be *test*);

• into the database a single table will be created (*students* – *id, firstname, lastname, age*)

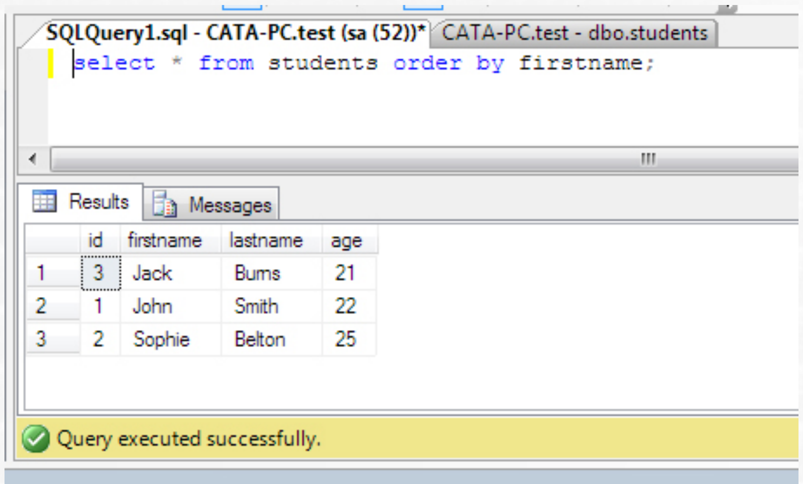
• *id* - is used to identify a record and should be set with the property *Identity Specification: Yes* (this will make him to increment automatically at every record inserted).

# Laboratory

- *New Query* option will be selected from the SQL Server management Studio application for being able to execute SQL commands.

- using the table just created insert some records into it using the INSERT command:
    - e.g. INSERT into students (firstname, lastname, age) values (‘John’, ‘Smith’, 23);

- select the records from the table *students* using the SELECT command
    - e.g. SELECT * FROM students ORDER BY firstname;
    - return the list of all students ordered by the firstname column.



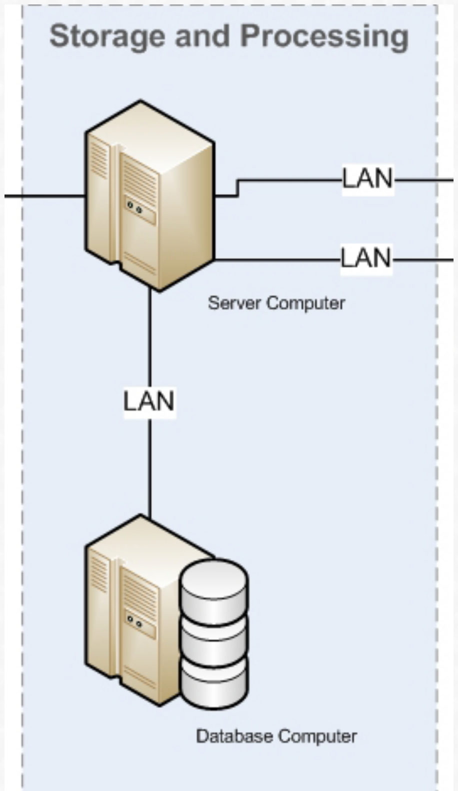
SQLQuery1.sql - CATA-PC.test (sa (52))*
CATA-PC.test - dbo.students

```
select * from students order by firstname;
```

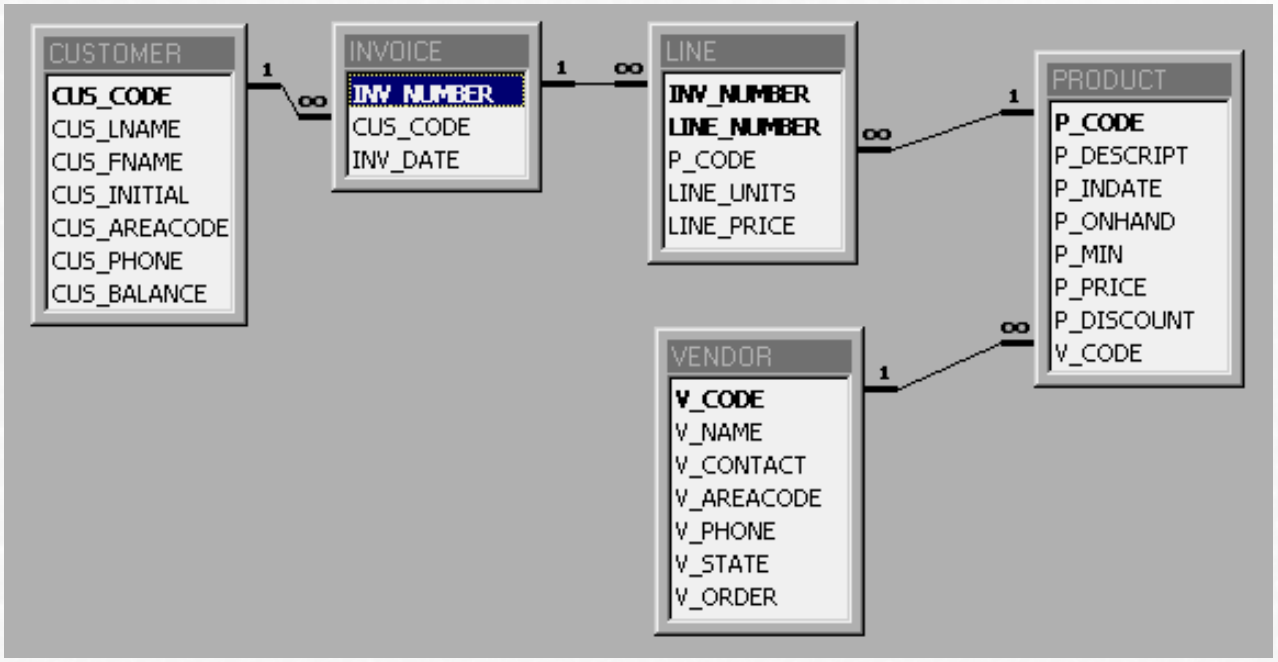
| | id | firstname | lastname | age |
|---|---|---|---|---|
| 1 | 3 | Jack | Bums | 21 |
| 2 | 1 | John | Smith | 22 |
| 3 | 2 | Sophie | Belton | 25 |

Query executed successfully.

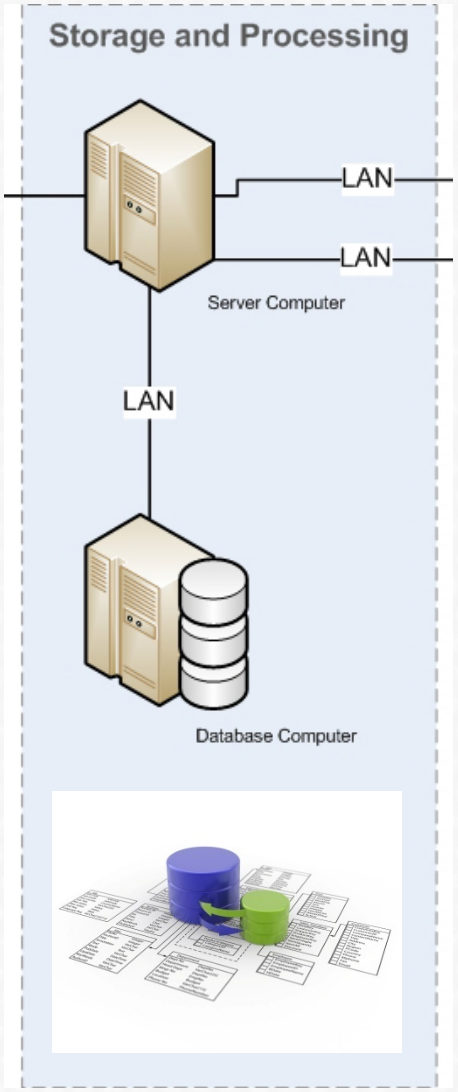
# Database – tables and diagrams



- the tables are interconnected with relationships (one-to-one; one-to-many; many-to-one; like in the diagram presented below);
- e.g. the *customer* table is linked to *invoice* table (relation one-to-many), which means that a customer could have multiple invoices and an invoice is only for one customer.



# Database – views



A view is a virtual table composed of multiple information from one or more tables.

The advantages of views over tables:

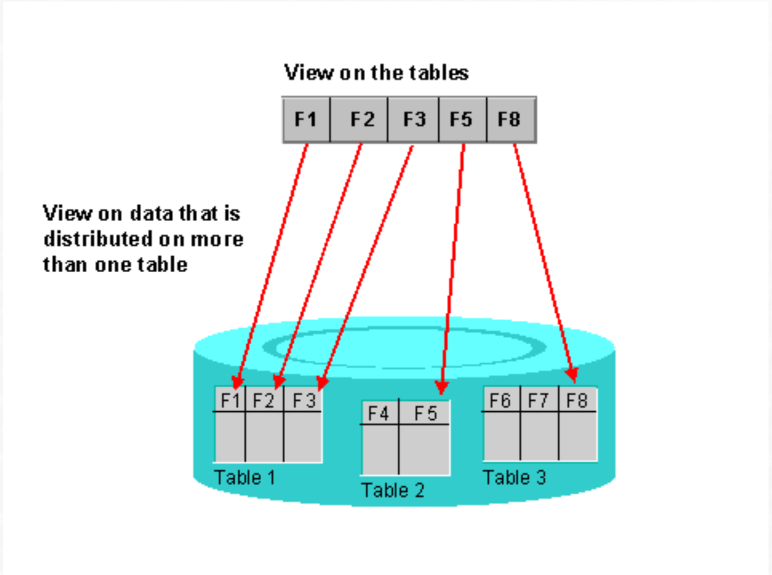
• can represent a subset of data contained in a table;

• can join information from multiple table into a single virtual table;

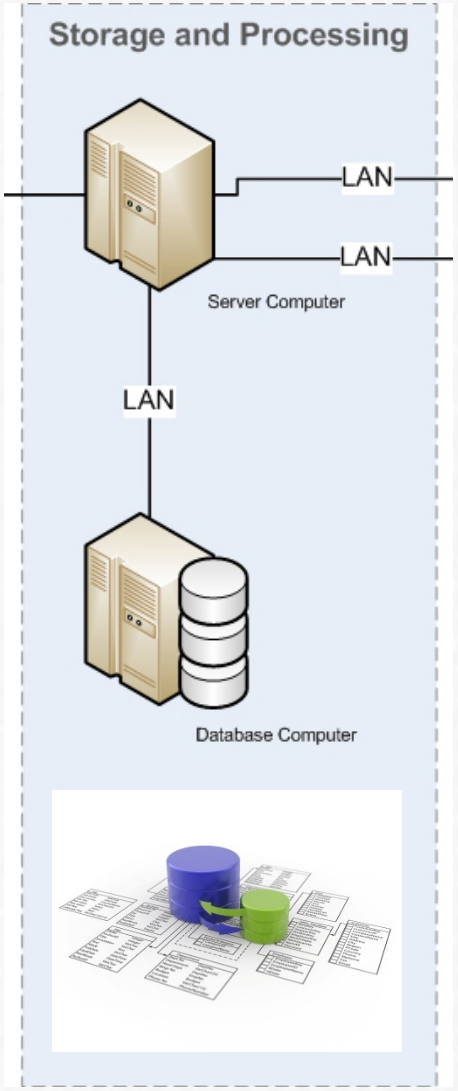
• simplify the complexity of the data;

• they require small space to store (only the definition of the view is saved by the database, not all the information which the view retrieves);

• could provide extra security.



# Database – stored procedures and triggers

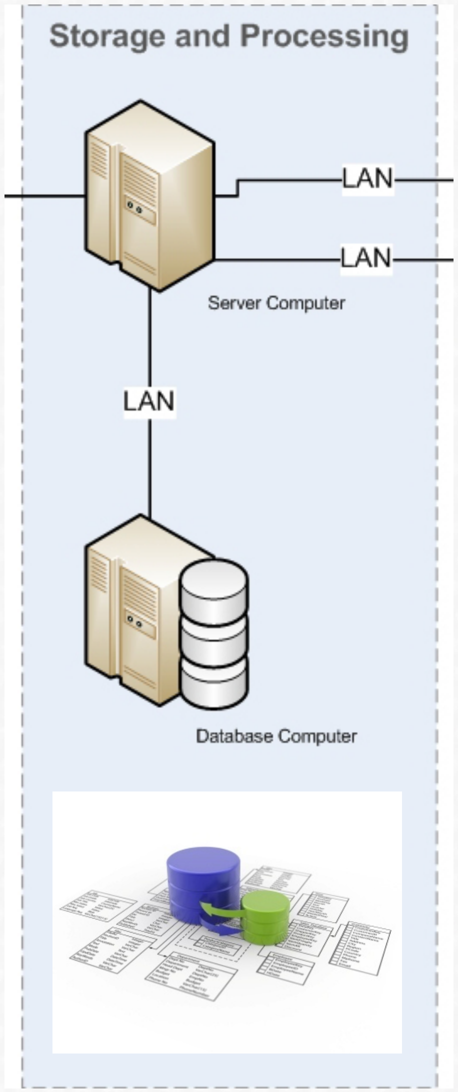


Stored procedure example (it retrieves the information stored in *column1* and *column2* from table *table1*):

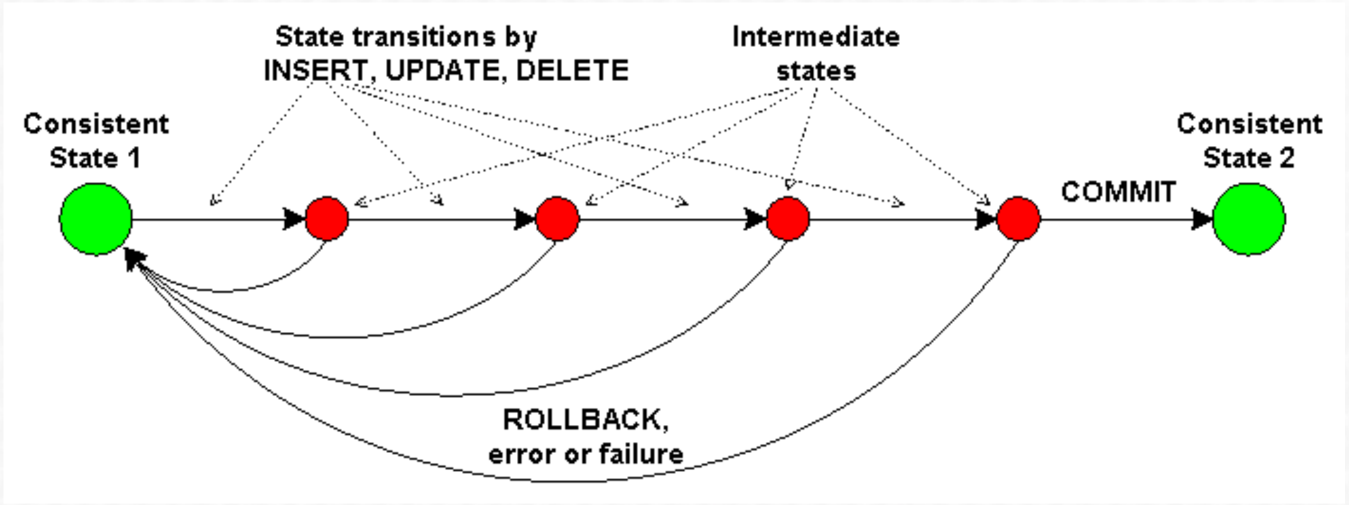
```
CREATE PROCEDURE sp_myStoredProcedure
AS
Select column1, column2 From Table1
Go
```

Triggers: execute procedures on different types of events (insert, update, delete). E.g. when information about a new user is inserted into the *users* table, into the table *emails* is generated automatically an email address.

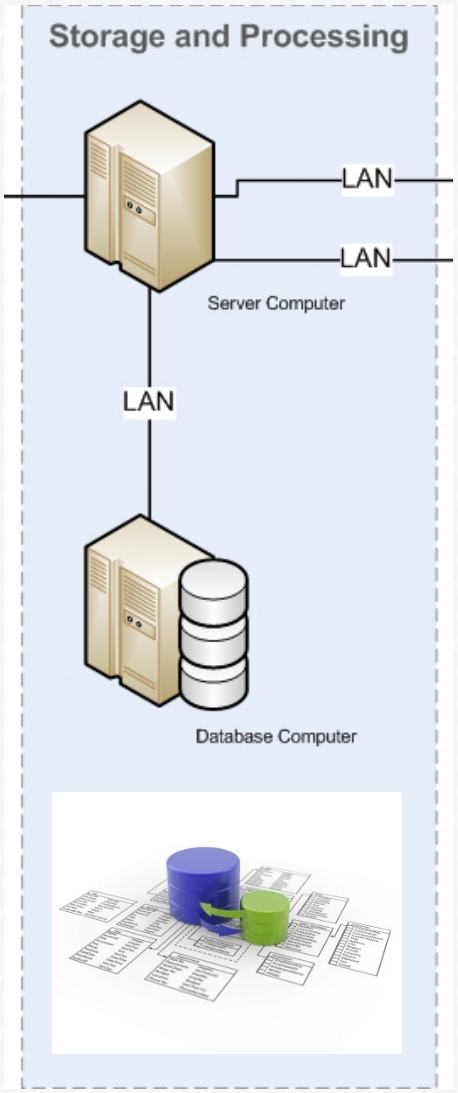
# Database – transactions



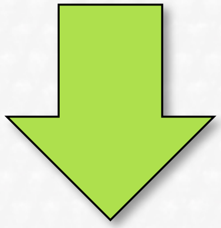
The transactions are very useful when we have to modify information in multiple tables in the same time. A transaction is represented by at least 2 database operations (if one operations fails all other operations *roll back* – their modifications to database are not saved; if all operations are successfully the modifications to the database are saved).



# Database



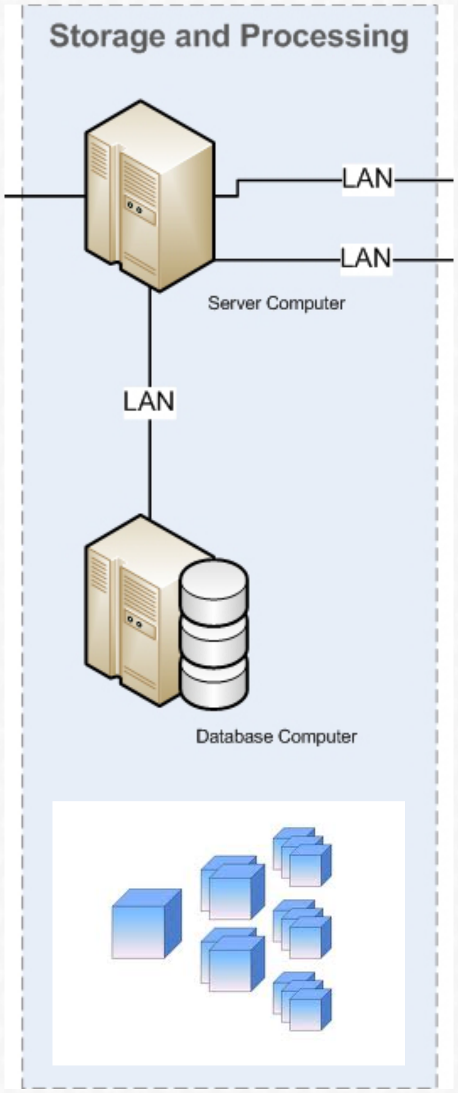
from *database theory*



(we go back to continue with)

*database optimization* (course 1)

# Database optimization - Normalization



It supposes:

- avoid similar information;
- avoid update anomalies;
- ensure the relation between attributes;
- the verification of the updates should not affect the integrity of the database.

The normalization should simplify the work and reduce the possibility to get errors at the update of the information from the database.

The normalization method permit to split a table in two or more tables and a set of normal forms (rules) should be followed for accomplish this task.

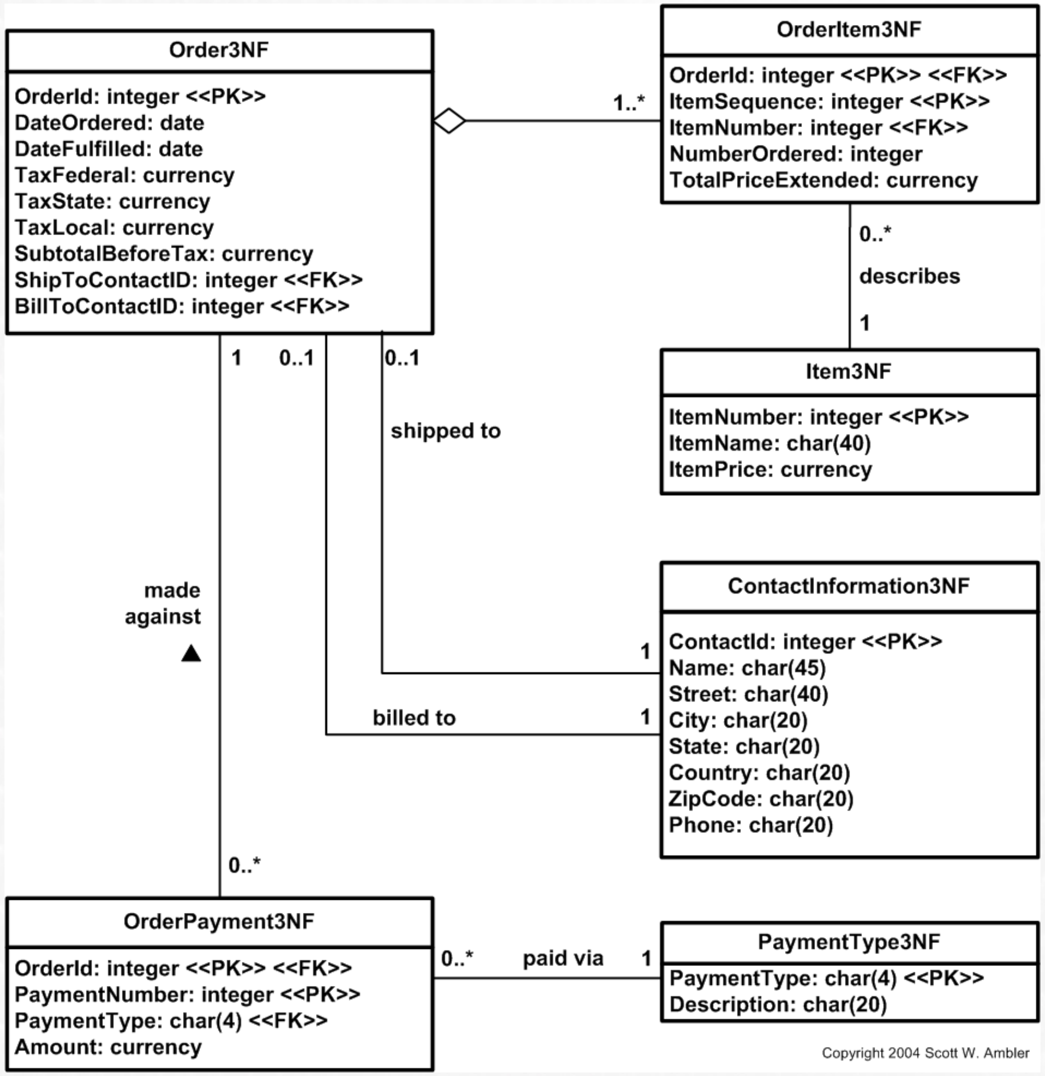
# Database optimization - Normalization

*before*

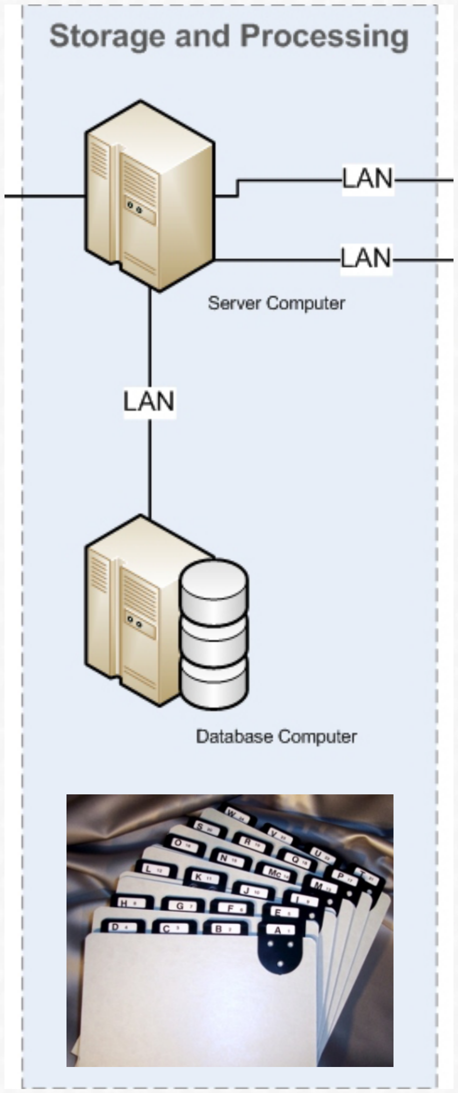
**Order0NF**

OrderId: integer <<PK>>
DateOrdered: date
DateFulfilled: date
Payment1Amount: currency
Payment1Type: char(4)
Payment1Description: char(40)
Payment2Amount: currency
Payment2Type: char(4)
Payment2Description: char(40)
TaxFederal: currency
TaxState: currency
TaxLocal: currency
SubtotalBeforeTax: currency
ShipToName: char(45)
ShipToStreet: char(40)
ShipToCity: char(20)
ShipToState: char(20)
ShipToCountry: char(20)
ShipToZipCode: char(20)
ShipToPhone: char(20)
BillToName: char(45)
BillToStreet: char(40)
BillToCity: char(20)
BillToState: char(20)
BillToCountry: char(20)
BillToZipCode: char(20)
BillToPhone: char(20)
ItemName1: char(40)
ItemNumber1: integer
NumberOrdered1: integer
InitialItemPrice1: currency
TotalPriceExtended1: currency
ItemName2: char(40)
ItemNumber2: integer
NumberOrdered2: integer
InitialItemPrice2: currency
TotalPriceExtended2: currency
. . .
ItemName9: char(40)
ItemNumber9: integer
NumberOrdered9: integer
InitialItemPrice9: currency
TotalPriceExtended9: currency

*after*



# Database optimization - Indexes

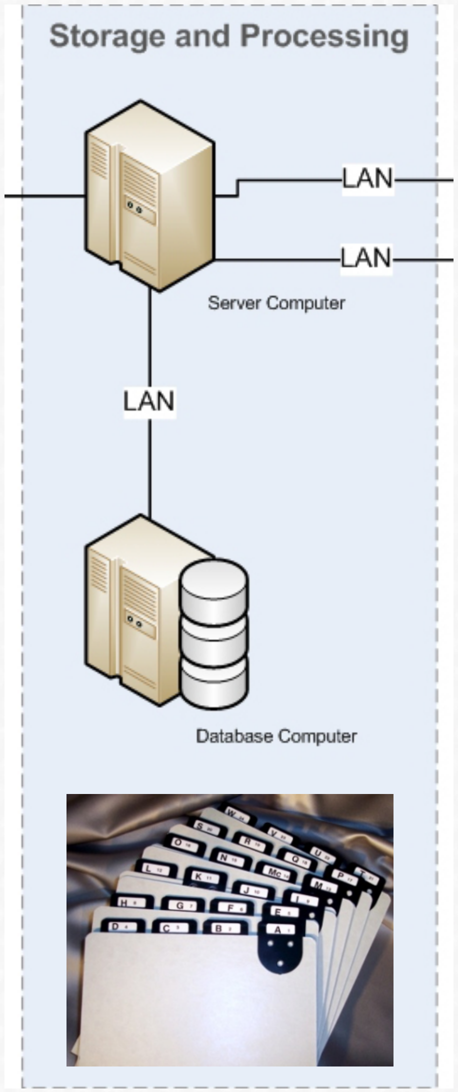


Is the method with the higher impact on the queries execution. It could contain only one column or multiple columns from the same table (the order matters). The definition of the index depends on the most used queries executed on that table.

Rules at creating an index:

- the column with the fewest unique values should be inserted first;
- for a table is preferable to have maximum 4-5 index objects. Since each index should be maintained and updated it have a proper cost.
- is preferable to have columns with integer values than columns with text;
- the index slows down the operation of insert, delete or update;
- it is recommended to use a clustered index than a non-clustered one (a clustered index orders the information on the hard disk, for fast reading of information when it is required).

## Database optimization - Indexes

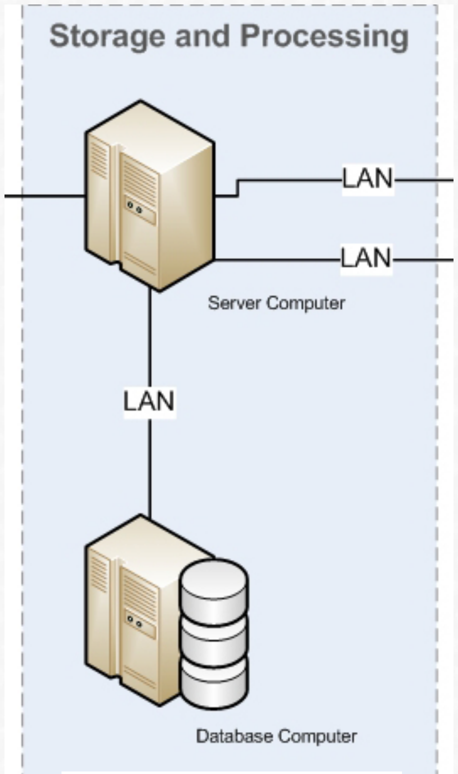


e.g. index after a *date & time* column (most used at selecting a set of records for a specific period of time).

For a table where are modified many of the records the index could become fragmentized (the performances of that table decrease). It is necessary to update the index:

- first method: delete the index and create it again;
- second method: defragmentation of the index.
- the first method is recommended, even it requires more time to be done and block the access to the tables.

# Database optimization - Statistics



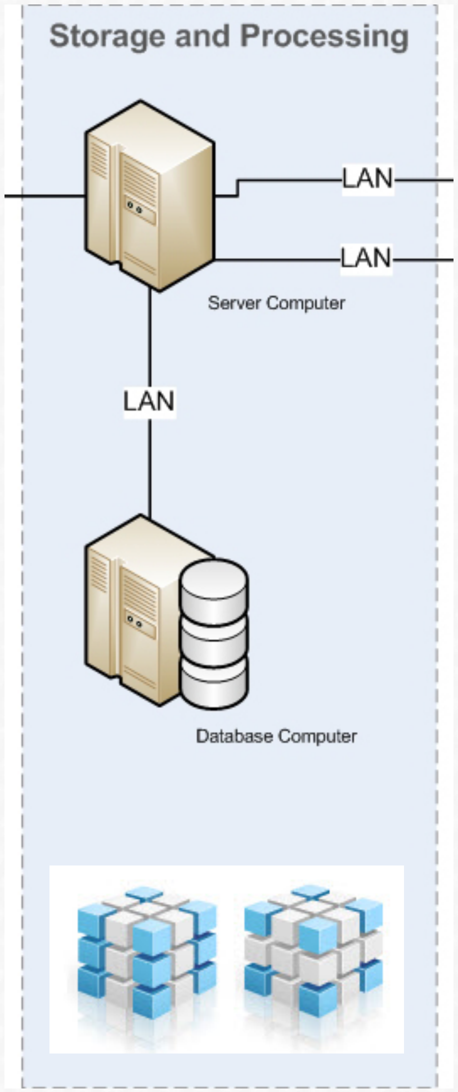
The statistics represent metadata (information about data):

- store information about index and columns from table;
- it is able to determine if an index improve the processing time or not;
- together with the indexes are one of the most important components which helps at improving the database performances;
- the database use statistics when it generates execution plans for accessing the necessary data.
- if they are wrong the database will have to search a long way until found the necessary data (or never found it);

A database should be able:

- to create automatically the statistics;
- to update them also automatically.

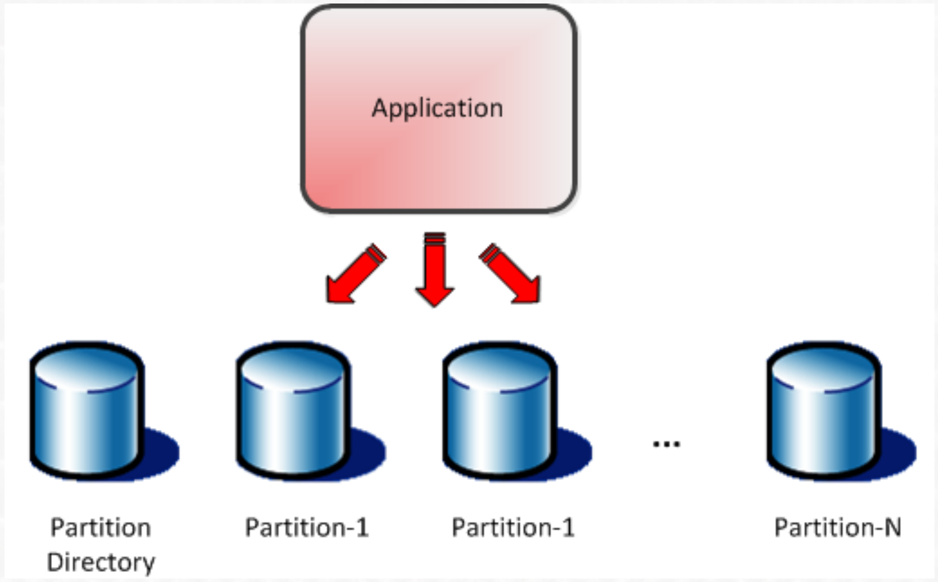
# Database optimization - partitioning



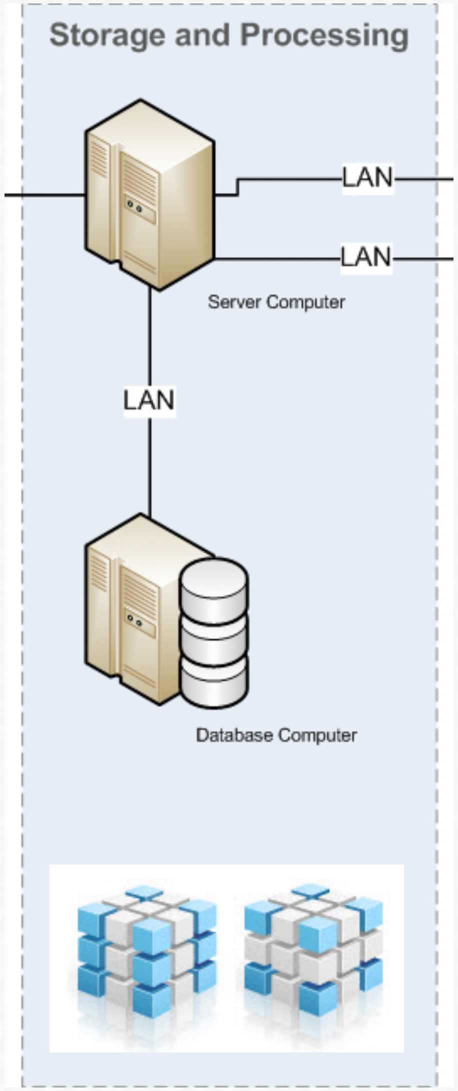
When a database contains a lot of information it is necessary to partition them according to some rules.

Partitioning = decompose a table into more tables.

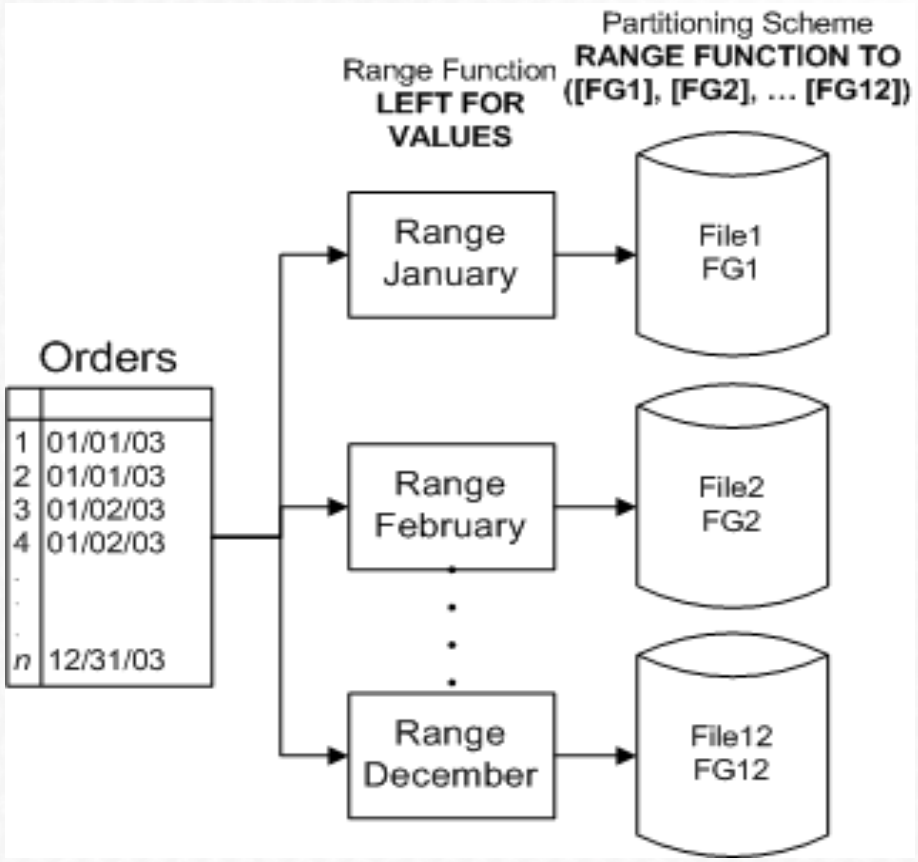
Partitioning
- horizontal (records from a table are put in separate tables);
- vertical (columns from a table are put in separate tables – rarely used).



# Database optimization – horizontal partitioning



e.g. information from a table is divided into other tables, according to the month when the information was recorded.


Partitioning Scheme
RANGE FUNCTION TO
([FG1], [FG2], … [FG12])
Range Function
LEFT FOR
VALUES
Orders
1 01/01/03
2 01/01/03
3 01/02/03
4 01/02/03
n 12/31/03
Range January
Range February
Range December
File1 FG1
File2 FG2
File12 FG12