
# 3rd Class

# Database

# قواعد بيانات

## إعداد : م. ياسر منذر

## أستاذة المادة : م.م. ندى نجيل

## 1.1 Introduction

Today, people use computers to perform many tasks formerly done with other tools. Computers have replaced typewriters for creating and modifying documents.

They've surpassed electromechanical calculators as the best way to do math.

They've also replaced millions of pieces of paper, file folders, and file cabinets as the principal storage medium for important information. Compared to those old tools, of course, computers do much more, much faster — and with greater accuracy. These increased benefits do come at a cost, however. Computer users no longer have direct physical access to their data. When computers occasionally fail, office workers may wonder whether computerization really improved anything at all. In the old days, a manila file folder only "crashed" if you dropped it — then you merely knelt down, picked up the papers, and put them back in the folder. Barring earthquakes or other major disasters, file cabinets never "went down," and they never gave you an error message.
A hard drive crash is another matter entirely: You can't "pick up" lost bits and bytes. Mechanical, electrical, and human failures can make your data go away and never to return.

If you are storing important data, you have four main concerns:

1- Storing data needs to be quick and easy, because you're likely to do it often.
2- The storage medium must be reliable. You don't want to come back later and find some (or all) of your data missing.
3- Data retrieval needs to be quick and easy, regardless of how many items you store.
4- You need an easy way to separate the exact information that you want today from the tons of data that you don't want right now.

*What is data?*
*** Data can be defined in many ways. Information science defines data an unprocessed information.*

*What is information?*
*** Information is data that have been organized and communicated in a coherent and meaningful manner.*
*- Data is converted into information, and information is converted into knowledge.*

*** Knowledge is information evaluated and organized so that it can be used purposefully as shown in figure (1.1)*

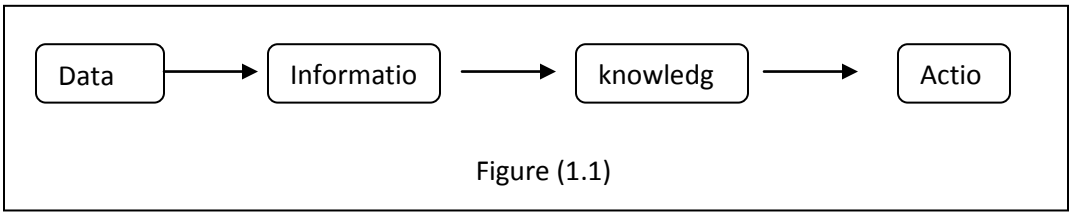



Figure (1.1)

**1.2 What is a Data Base ?**

A **database** is an organized collection of data for one or more uses, typically in digital form. The data can be textual, like order or inventory data, or it can be pictures, programs or anything else that can be stored on a computer in binary form.

One way of classifying databases involves the type of their contents, for example: bibliographic, document-text, statistical.

The purpose of a database is to store and retrieve related information, so databases are designed to offer an organized mechanism for :

- Storing
- managing
- and retrieving information.

**1.3 Files System:**

The File is a block of arbitrary information, it is a place that application programs stores there data in it. These application programs either database application or non-database application. Each file has a format. The information stored in the file can be organized in a record, which is a collection of fields.

The file system is typically described as various files and a number of different application programs are written to read from and add to the appropriate files.

File System Disadvantage:

- Program dependence : Each file has a format, the non-database application must know exactly the format of the file to deal with it. Any other application cannot access the file unless knowing the format of the file.
- When file format updated, then the application program must be updated, it is complicated to update all programs when data format is update.
- Security problems existed. Any one can write a program to read the data in the file.
- Data redundancy , if there are application A deals with file A and application B deals with file B, if application A store an information in file A, and if application B need this information , application B can not access file A , so application B must record the same information in file B.

*Some basic Definitions:*

Field: one category of information( one data value), i.e., Name, Address, Semester Grade, Academic topic.

Record: Collection of fields i.e., one student's information, a recipe, a test question.

A File: A group or collection of similar records, like student File.

Digital databases are managed using database management systems (DBMS) , which store database contents, allowing data creation and maintenance, and search and other access to the database.

**1.4 What is DBMS ?**

A **Database Management System** (**DBMS**) is a set of computer programs that controls the –

- Creation of the database
- The storing and organization of the data in the database
- Maintenance the database
- Searching ,data retrieval and the use of a database.

The DBMS accepts requests for data from an application program and instructs the operating system to transfer the appropriate data as shown in figure (1.2)

*ADVANTAGES OF A DBMS*

1- Database Development: It allows organizations to place control of database development in the hands of database administrators (DBAs) and other specialists.

2-Data independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

3-Efficient data access: A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. It allows different user application programs to easily access the same database. Instead of having to write computer programs to extract information, user can ask simple questions in a query language.

4-Data integrity and security: If data is always accessed through the DBMS, the DBMS can enforce :

- integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded.
- Also, the DBMS can enforce access controls that govern what data is visible to different classes of users.

5-Crash recovery: the DBMS protects users from the effects of system failures.

6- Data administration and Concurrent access: When several users share the data( more than one user access the database at the same time), DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time.

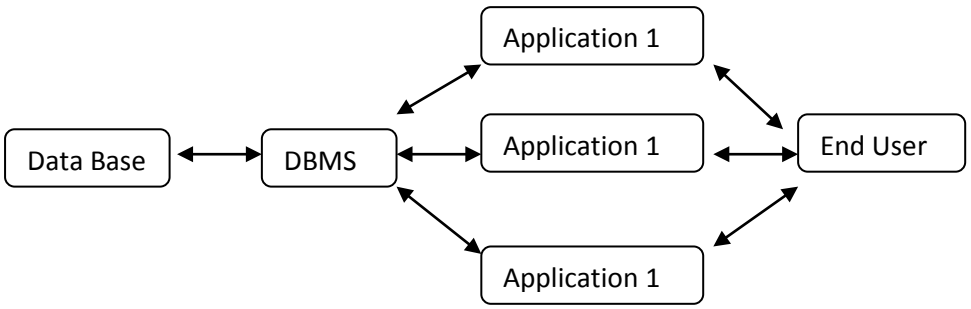



Figure (1.2)

DBMS

**1.5 Data Abstraction**

The major purpose of a database system is to provide users with an abstract view of the data. That is , the system hides certain details of how the data are stored and maintained.

For the system to be usable, it must retrieve data efficiently. This had led to the design of complex data structure to represent the data in the data base. Since many database-user are not computer trained , data base developers hide the complexity from users through several levels of abstraction as shown in figure (1.3).

** **Physical level** : the lowest level of abstraction, describe ***how*** the data are actually stored. At the Physical level , complex low level data structure are described in detail. In this level storage locations used to specify where the data are and the data been described by words and bytes.

** **Logical level** : This level describe ***what*** data are stored in the database and what relationship exist among those data. The entire database is describe in terms of small number of relatively simple structure. The logical level is used by the database administrator, who must decide what information is to be kept in the database.

** **View level** : The highest level of abstraction describes only part of the entire database. Many users of the database system will not be concern with all the information. Instead the user need to access only part of the database , so a view level of abstraction is defined .

The system may provide many views for the same database.

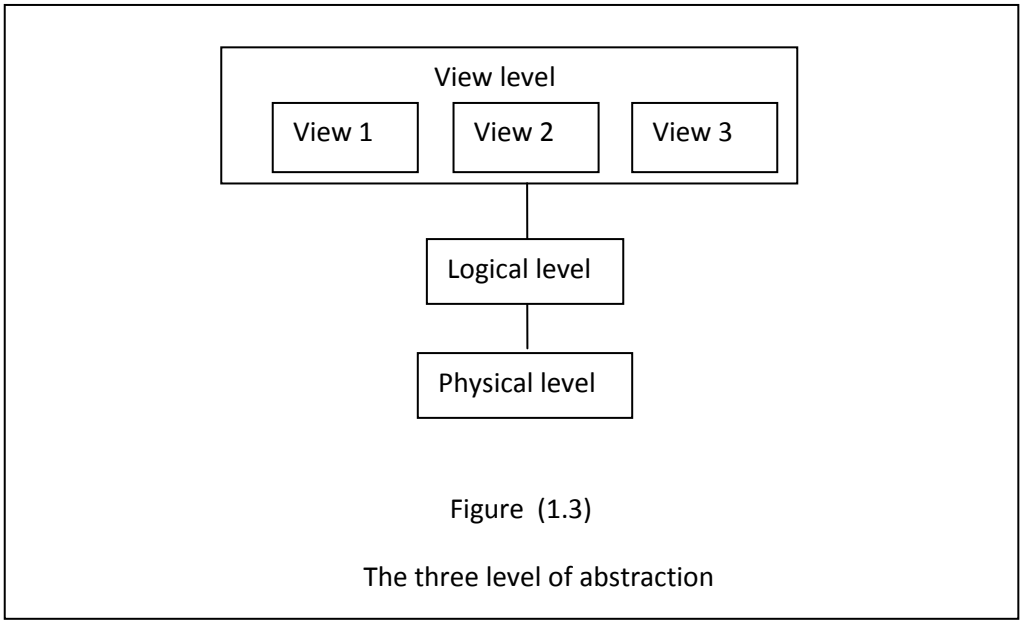



Figure (1.3)

The three level of abstraction

## 2.1 Database Models

A **data model** is a collection of tools that describes how data are represented and accessed. Data models formally define data elements and relationships among data elements for a domain of interest ***(data element is an unit of data that has precise meaning or precise semantics)***

A data model explicitly determines the structure of data . The main aim of data models is to support the development of information systems by providing the definition and format of data.

If the same data structures are used to store and access data then different applications can share data .

Communication and precision are the two key benefits that make a data model important to applications that use and exchange data. A data model is the medium which project team members from different backgrounds and with different levels of experience can communicate with one another. ***(Precision means that the terms and rules on a data model can be interpreted only one way and are not ambiguous)***

There are many deferent models:

1- The Entity Relationship model
2- The Relational data model
3- Object Based data model
4- The Network data model
5- The Hierarchical data model

### 2.1.1 : The Entity Relationship model

The entity-relationship model (or ER model) is a way of graphically representing the logical relationships of entities (or objects) in order to create a database.
In ER modeling, the structure for a database is portrayed as a diagram, called an entity-relationship diagram (or ER diagram),

The ER model consists of the following :

- **Entity**: a thing that exists and which can be uniquely identified
  e.g. person, automobile, department, employee
- **Entity Set**: a group of similar entities
  e.g. all persons, all automobiles, all employees
- **Relationship**: association between entities
  e.g. a person is assigned to a department
- **Relationship Set** : set of similar relationships
- **Attribute**: property of an entity or relationship
  e.g. person - name, address
- **Domain**: set of values allowed for an attribute

An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order

A relationship captures how two or more entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns.
Examples: an ***owns*** relationship between a company and a computer.
: a ***supervises*** relationship between an employee and a department
: a ***performs*** relationship between an artist and a song,.
: a ***proved*** relationship between a mathematician and a theorem

Entity sets are drawn as rectangles, attributes are drawn as oval. Entity is connected with an attribute with lines. Diamonds represent relationship among entity set.

Figure 2.1 shows an ER diagram notation for an attribute (*Grade*) of an entity (*student*)

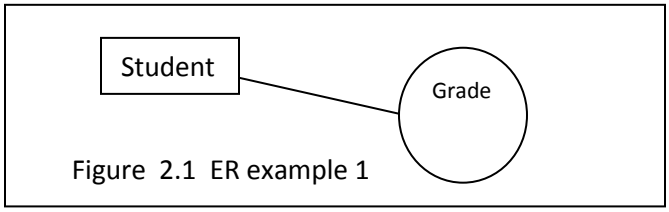


Figure 2.1 ER example 1

Figure 2.2 shows the following example :

There are three entity:

1- Employees : E#,ENAME, ADDRESS
2- Departments : D#,DNAME
3-Projects : PNAME

There are two relations that connect the three entities

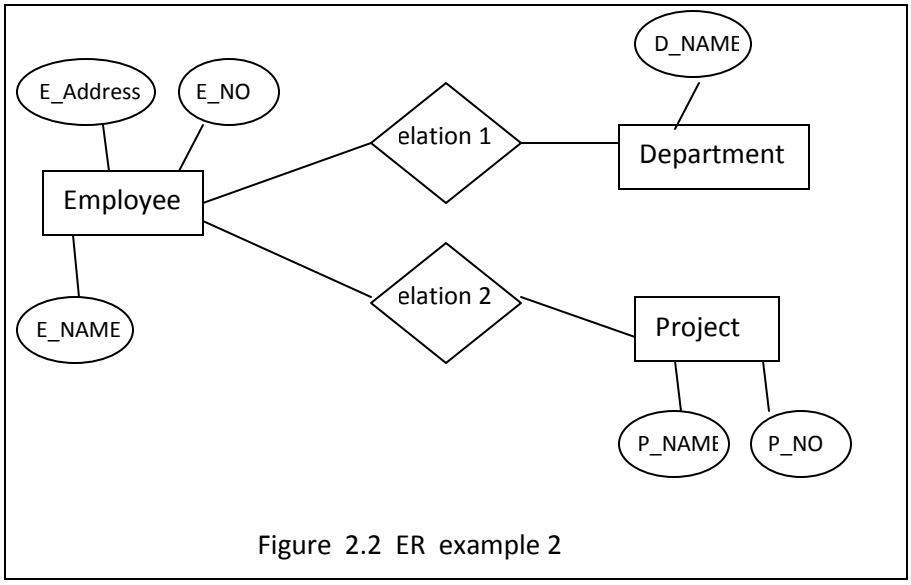



Figure 2.2 ER example 2

### 2.1.2 The Relational model

The Relational Model is a clean and simple model that uses the concept of a relation using a table rather than a graph or shapes. The information is put into a grid like structure that consists of columns running up and down and rows that run from left to right, this is where information can be categorized and sorted.

The relational model used the basic concept of a relation or table. The columns or fields in the table identify the attributes such as name, age, and so. A tuple or row contains all the data of a single instance of the table such as a person named Doug. In the relational model, every tuple must have a unique identification or key based on the data as shown in figure 2.3 , a social security account number (SSAN) is the key that uniquely identifies each tuple in the relation. Often, keys are used to join data from two or more relations based on matching identification.

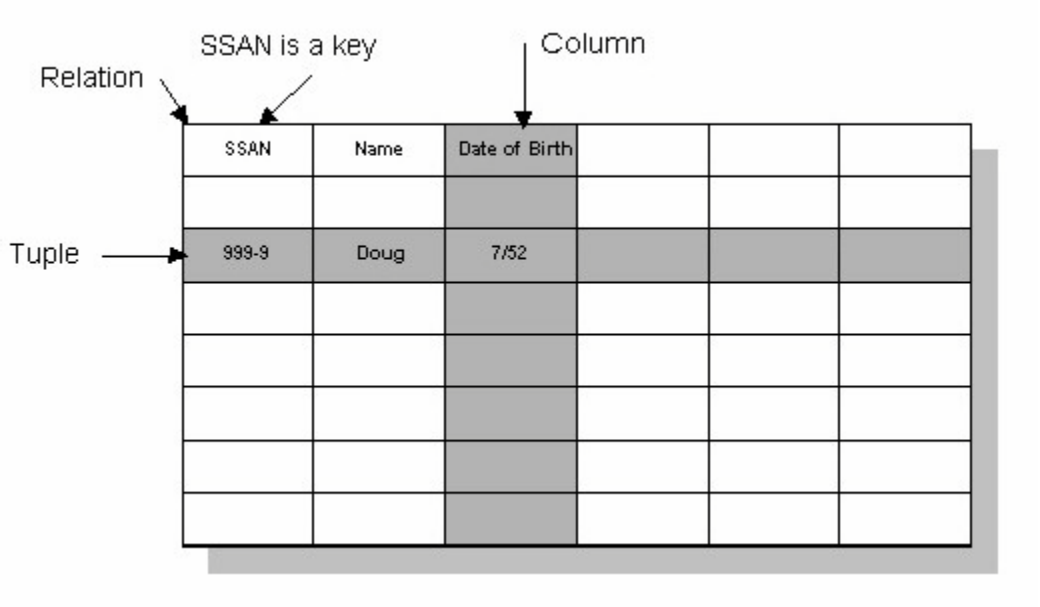



Figure 2.3 Relation (Table)

### 2.1.3 Object Based data model

An object database (also object-oriented database) is a database model in which information is represented in the form of objects as used in object-oriented programming

Today's trend in programming languages is to utilize objects, thereby making OODBMS ideal for OO programmers because they can develop the product, store them as objects, and can replicate or modify existing objects to make new objects within the OODBMS

### 2.1.4 Hierarchical and Network data model

A ***hierarchical database model*** is a data model in which the data is organized into a tree-like structure. The structure allows representing information using parent/child relationships: each parent can have many children, but each child has only one parent (also known as a 1-to-many relationship). All attributes of a specific record are listed under an entity type. Figure 2.4 shows as example.
This model is recognized as the first database model created by IBM in the 1960s.

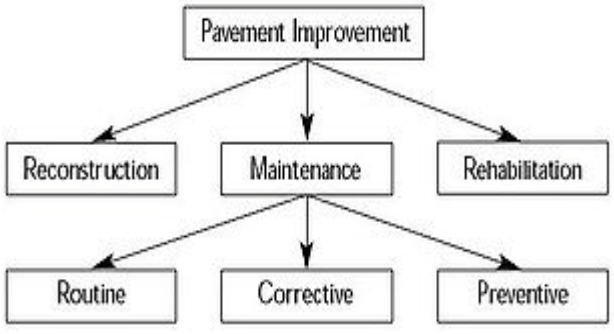
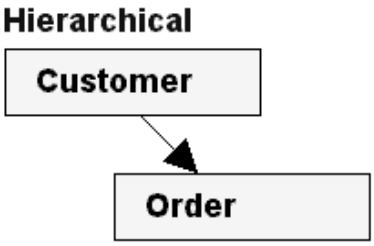
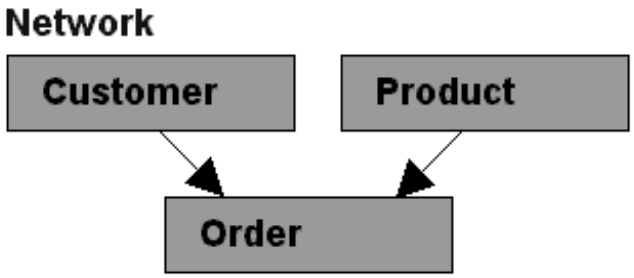



Figure 2.4 Hierarchical model

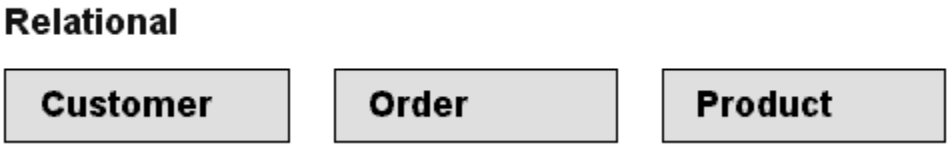
In the following example, orders are owned by only one customer. This model often restrictive in linking real-world structures.



In ***network databases***, data is organized as a graph ,a record type can have multiple owners. In the hierarchical model, each record having one parent record and many children, the network model allows each record to have multiple parent and child records, forming a generalized graph structure.
In the example below, orders are owned by both customers and products, reflecting their natural relationship in business.




Relational databases do not link records together physically, but the design of the records must provide a common field, such as account number, to allow for matching. Often, the fields used for matching are indexed in order to speed up the process.
In the following example, customers, orders and products are linked by comparing data fields and/or indexes when information from more than one record type is needed. This method is more flexible for ad hoc inquiries.



## 3 Relational Database

A relational database consists of a collection of tables, each of which assigned a unique name. The table consists of a number of rows . A row in a table represents a relation among a set of values.

A relational database is a collection of data items organized as a set of tables from which data can be accessed in many different ways without having to reorganize the database tables. The relational database was invented by E. F. Codd at IBM in 1970.

A relational database is a set of tables containing data fitted into predefined categories. Each table (which is sometimes called a *relation*) contains one or more data categories in columns. Each row contains a unique instance of data for the categories defined by the columns.

Consider the account table as shown in figure 3.1 . It has three column header (three fields) :

- Branch-name
- Account-number
- Balance

| Branch-name | Account-umber | Balance |
|---|---|---|
| A | A-101 | 500 |
| B | A-215 | 234 |
| C | A-234 | 800 |

Figure 3.1

Account table

## 3.1 Instance and schemas

Database changes over time as information is inserted and deleted.

*Instance* : is the collection of information stored in the database at a particular moment is called an *instance* of the database.

*Schema* : the overall design of the database is called the *database schema*. Schemas are changed infrequently.

The database schema is the logical design of the database, database instance is a snapshot of the data in the database at a given instance of time.

Schemas, like tables and fields, have names. For example the schema that describe the table in figure 3.1 could be :

Account table-Schema= (Branch-name , Account-number , Balance)

## 3.2 Keys

As you may already know, databases use tables to organize information. Each table consists of a number of rows, each of which corresponds to a single database record. So, how do databases keep all of these records straight? It's through the use of keys.

### Primary Keys

The first type of key we'll discuss is the primary key. ***Every database table should have one or more columns designated as the primary key***. The value this key holds should be unique for each record in the database. For example, assume we have a table called Employees that contains personnel information for every employee in our firm. We'd need to select an appropriate primary key that would uniquely identify each employee. Your first thought might be to use the employee's name.

This wouldn't work out very well because it's conceivable that you'd hire two employees with the same name. A better choice might be to use a unique employee ID number that you assign to each employee when they're hired

Once you decide upon a primary key and set it up in the database, ***the database management system(DBMS)*** will enforce the uniqueness of the key. If you try to insert a record into a table with a primary key that duplicates an existing record, the insert will fail.

### Foreign Keys

The other type of key that we'll discuss in this course is the foreign key. These keys are used to create relationships between tables. Natural relationships exist between tables in most database structures

The foreign key link is set up by matching columns in one table (the child) to the primary key columns in another table (the parent).

Example 1 :In the example of figure 3.2, there is a link between the Company and Contact tables. The Company table is the parent table in the link. The Contact table is the child: the Company ID field in the Contact table indicates which Company a Contact belongs to.

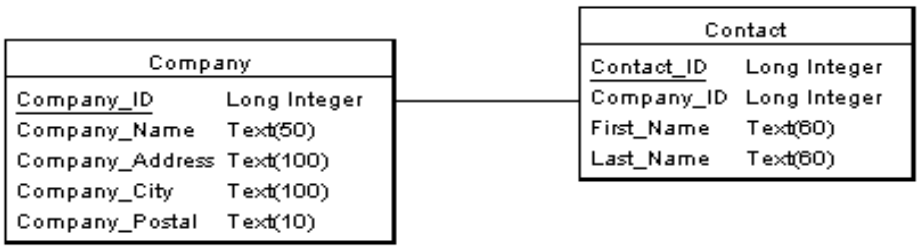



Figure 3.2 Example 1 of a relation between two tables

Example 2 : suppose there is a need to design a database to hold the history of each student of a school. The information of each student should have :

- Student name
- Student phone
- Information about the student in each year in the school.

This database should have at least two tables , the first one like the following schema:

St Table-Schema=(st-name, st-phone)

The second table contain the information about the history of the student :

History-schema=(year , level , state)

To link those two tables a primary key in the first table and a foreign key in the second table must be added to the tables as shown in figure 3.3 .

The new schemas will be :

St Table-Schema=(S-ID , st-name, st-phone)

History-schema =(S-ID , year , level , state)

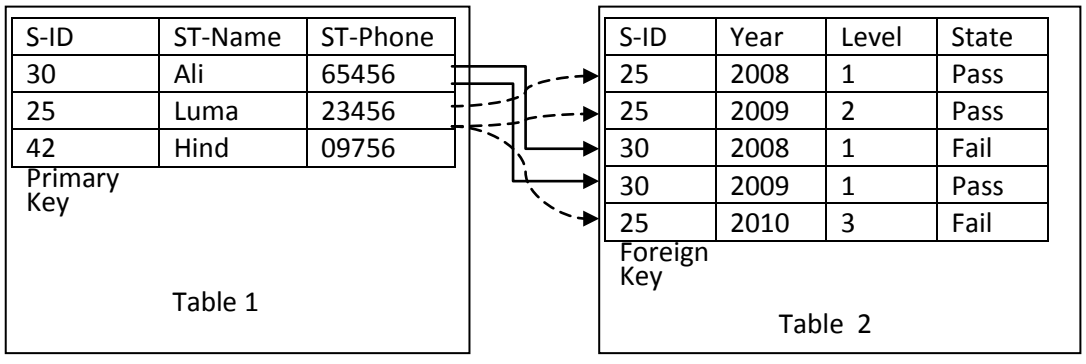



Figure 3.3 Example (2) of a relation between two tables

## 3.3 Structured Query Language (SQL)

SQL (Structured Query Language) is a database computer language designed for managing data in relational database management systems (RDBMS), and originally based upon Relational Algebra.

SQL designed to organize and simplify the process of getting information out of a database in a usable form, and also used to reorganize data within databases.
It is a set of statements to manage databases, tables, and data.
Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server and Microsoft Access.

SQL falls into two classes

1. Data Definition Language (DDL) - SQL for creating, altering and dropping tables
2. Data Manipulation Language (DML) - SQL for retrieving and storing data.

Data Definition Language (DDL) : a database schema is specified by a set of definitions expressed by the a language called (DDL).
It is the subset of SQL used for defining and examining the structure of a database. It used to define databases and their components.

Data Manipulation Language (DML): The data in the database are manipulated by the (DML) . With (DML) we can

- Retrieve information from the database
- Inserting new information into the database
- Delete information from the database
- Updating the data in the database

**3.4 Table Joining**

A JOIN is a means for combining fields from two tables by using values common to each.

Table Joining is a formal specification of which column(s) in a row in one table should be matched to a column(s) in a row in another table.

To combine tables, i.e. to perform a join, we have to select data from the tables, and relate the tables to each other with conditions on some attributes (often keys attributes).

Joining tables is used when retrieving information from the database is required .

***In SQL the word ( SELECT ) is used to display information from the database followed by the name of the tables***.

to explain the types of joins in the database, the following three tables are used as an example:

```
teachers table

 id |  name
----+--------
  1 | Volker
  2 | Elke
(2 rows)


Projects table

 id |   name   | duration | teacher
----+----------+----------+---------
  1 | compiler |      180 |       1
  2 | xpaint   |      120 |       1
  3 | game     |      250 |       2
  4 | Perl     |       80 |       4
(4 rows)


Assign table

 project | stud | percentage
---------+------+------------
       1 |    2 |         10
       1 |    4 |         60
       1 |    1 |         30
       2 |    1 |         50
       2 |    4 |         50
       3 |    2 |         70
       3 |    4 |         30
(7 rows)
```

**3.4.1 : Cross Join:**

Each row from table one is arbitrarily combine with each row from table two.

This is known as a Cartesian product. In practical terms a cross join is a join without a join condition
Each row in teachers is arbitrarily combined with each row in projects.:

Example: Cross join of tables *teachers* and *project* will be

```
> SELECT * FROM teachers, projects;

 id |  name  | id |   name   | duration | teacher
----+--------+----+----------+----------+---------
  1 | Volker |  1 | compiler |      180 |       1
  1 | Volker |  2 | xpaint   |      120 |       1
  1 | Volker |  3 | game     |      250 |       2
  1 | Volker |  4 | Perl     |      80  |       4
  2 | Elke   |  1 | compiler |      180 |       1
  2 | Elke   |  2 | xpaint   |      120 |       1
  2 | Elke   |  3 | game     |      250 |       2
  2 | Elke   |  4 | Perl     |      80  |       4

(8 rows)
```

**3.4.2 : Inner Join :**
Inner joins are the most common type of joins, they combine two or more tables using where clause as main condition with equality "=" sign or inequality "!=" .

A field from the first table is compared to another field from the second table and if they are equal ( or not equal ) the two records from the two tables are combined.

Example : inner join of tables *teachers* and *project* if the condition is
teachers.id = projects.id will be

SELECT * FROM teachers, projects ***where*** teachers.id = projects.id;

```
id |  name  | id |   name   | duration | teacher
----+--------+----+----------+----------+---------
  1 | Volker |  1 | compiler |      180 |       1
  2 | Elke   |  2 | xpaint   |      120 |       1
```

Example : inner join of tables *teachers* and *project* if the condition is
teachers.id ! = projects.id will be

SELECT * FROM teachers, projects where teachers.id != projects.id;

```
id |  name  | id |   name   | duration | teacher
---+--------+----+----------+----------+---------
  1 | Volker |  2 | xpaint   |      180 |       1
  1 | Volker |  3 | game     |      180 |       1
  1 | Volker |  4 | Perl     |      180 |       1
  2 | Elke   |  1 | compiler |      120 |       1
  2 | Elke   |  3 | game     |      120 |       1
  2 | Elke   |  4 | Perl   |      120 |       1
```

There is another format for inner join

***select T1.F1, F2,F3,…,FN from T1 inner join T2 <u>on</u> (T1.F1 = T2.F1) inner join T3 <u>on</u> (T2.F1=T3.F1) inner join ……***

**3.4.3 LEFT OUTER JOIN:**

this type of join will first select all shared records from those tables, and then will bring all records from left table which are not exist in the second table.

**3.4.4 RIGHT OUTER JOIN**

this type of join will first select all shared records from those tables, and then will bring all records from right table which are not exist in the first table

**3.4.5 FULL OUTER JOIN**

this type of join will first select all shared records from those tables, and then will bring all records exist in first table which are not exist in the second table and then bring all records exist in the second table which are not exist in first table.

select teachers.id , teachers.name , projects.id , projects.name from teachers full OUTER join projects on (teachers.id = projects.id);

```
id |  name  | id |   name
---+--------+----+---------
  1 | Volker |  1 | compiler
  2 | Elke   |  2 | xpaint
               4 | Perl
               3 | games
```

**4 : Database Administrator**

A **database administrator** (**DBA**) is a person responsible for :

- Design and Implementation of the database
- maintenance and repair of the database
- monitoring and improving database performance and capacity
- planning for future expansion requirements.

A basic responsibility for just about every database administrator involves the installation of new databases. As part of the database installation, the database administrator will set up login credentials to authorized persons, define the privileges associated with each authorized user, and ensure that every work station attached to the network is set up to access the new database.

**5: Database Design**

Database design is the process of producing a detailed data model of a database , this data model which can then be used to create a database.
The term database design can be used to describe many different parts of the design , 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.

**5.1 : The Design Process**

The design process consists of the following steps:

1. Determine the purpose of your database - This helps prepare you for the remaining steps.
2. Find and organize the information required - Gather all of the types of information you might want to record in the database, such as product name and order number.
3. Divide the information into tables - Divide your information items into major entities or subjects, such as Products or Orders. Each subject then becomes a table.
4. Turn information items into columns - Decide what information you want to store in each table. Each item becomes a field, and is displayed as a column in the table. For example, an Employees table might include fields such as Last Name and Hire Date.
5. Specify primary keys - Choose each table's primary key. The primary key is a column that is used to uniquely identify each row. An example might be Product ID or Order ID.
6. Set up the table relationships - Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. Refine your design - Analyze your design for errors. Create the tables and add a few records of sample data. See if you can get the results you want from your tables. Make adjustments to the design, as needed.
8. Apply the normalization rules - Apply the data normalization rules to see if your tables are structured correctly. Make adjustments to the tables, as needed.

**5.2:Database Cardinality**

In data modeling, the cardinality of one data table with respect to another data table is a critical aspect of database design. Relationships between data tables define cardinality when explaining how each table links to another.

In the relational model, tables can be related as any of:

- **many-to-many**
- **many-to-one** (rev. **one-to-many**)
- **one-to-one**

This is said to be the **cardinality** of a given table in relation to another.

For example, considering a database designed to keep track of hospital records. Such a database could have many tables like:

- a *Doctor* table full of doctor information
- a *Patient* table with patient information
- and a *Department* table with an entry for each department of the hospital.

In that model:

- There is a **many-to-many** relationship between the records in the doctor table and records in the patient table (Doctors have many patients, and a patient could have several doctors);
- a **one-to-many** relation between the department table and the doctor table (each doctor works for one department, but one department could have many doctors).
- **one-to-one** relationship is mostly used to split a table in two in order to optimize access or limit the visibility of some information. In the hospital example, such a relationship could be used to keep apart doctor's personal or administrative information.

Example 1 : A chemical factory producing chemical materials , each material identified by a name and a formula .
The supplier , identified by his name and his ID , purchase from the factory by an order . The order has date , amount and total.

To draw the ER model
Each supplier can have any material , and any material can go to any supplier . The relation is of type **many-to-many ,** as shown in figure 5.1.

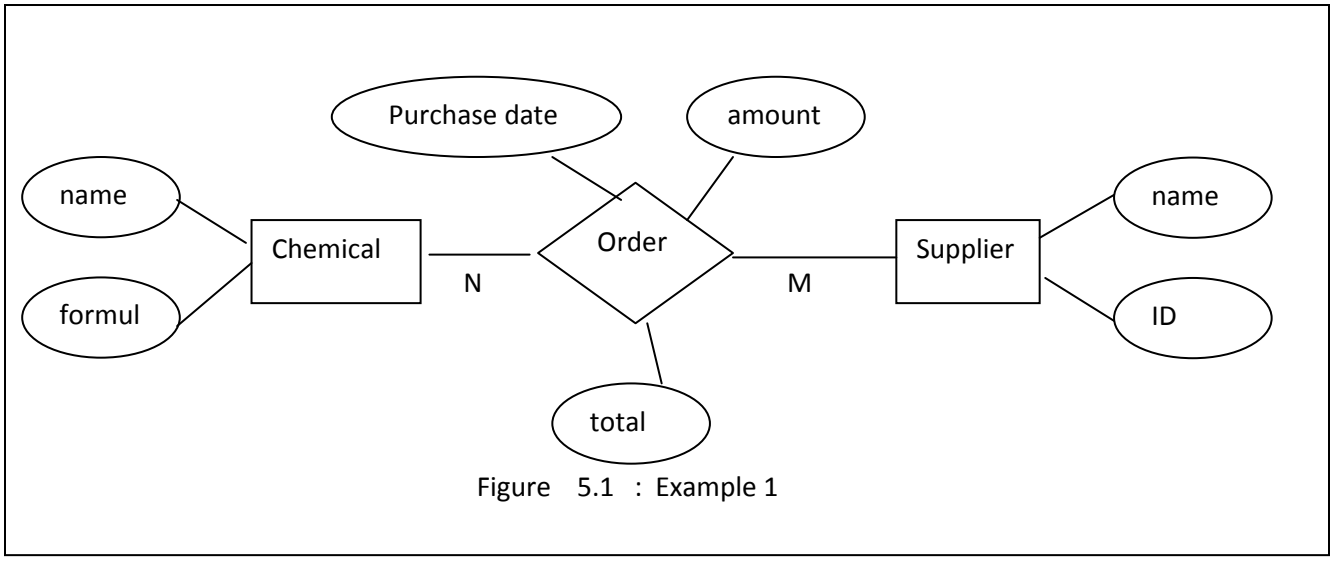
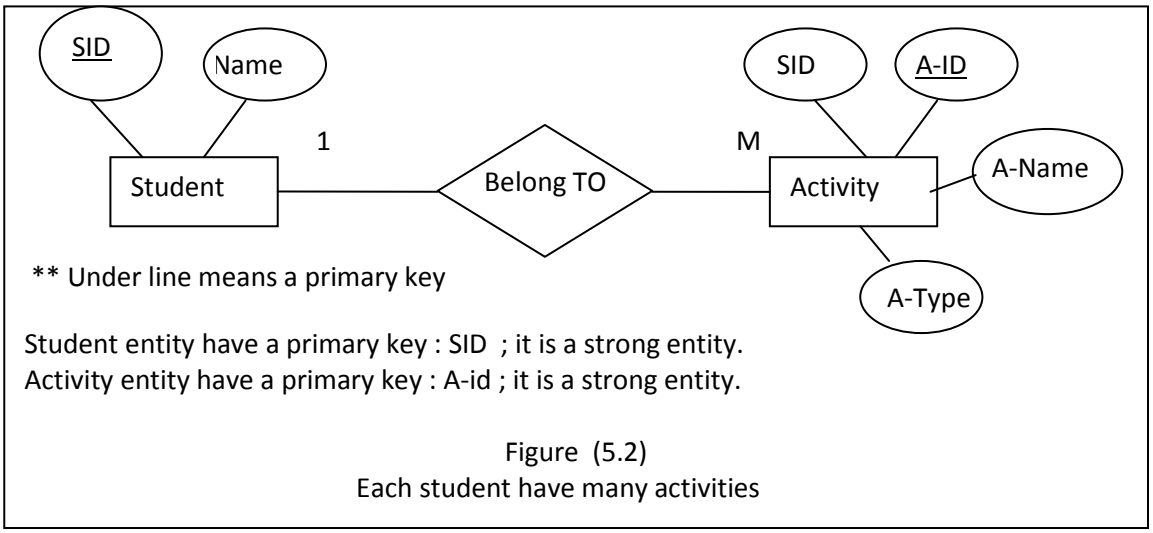



Figure 5.1 : Example 1

## 5.3 : Weak Entity

An entity may not have sufficient attributes to form a primary key. Such an entity is termed a ***weak entity .*** An entity that has a primary key is termed a ***strong entity.***

Consider the relationship ***Have*** which links a student with his activates as shown in figure (5.2).




** Under line means a primary key

Student entity have a primary key : SID ; it is a strong entity.
Activity entity have a primary key : A-id ; it is a strong entity.

Figure (5.2)
Each student have many activities

If more details is needed about all the activities the student have (history) , another entity will be added as shown if figure (5.3).

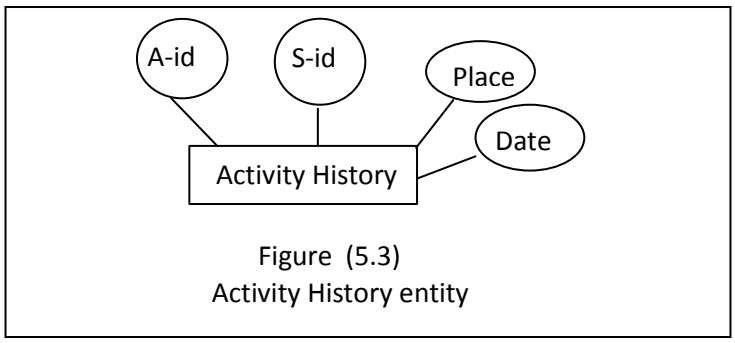



Figure (5.3)
Activity History entity

This new entity have S-ID and A-ID as a foreign keys to recognize each record belong to what student and what activity. This mean for each activity a student have , he/she can carry it many times.

Figure (5.4) shows how the new entity , Activity History , is linked with activity entity

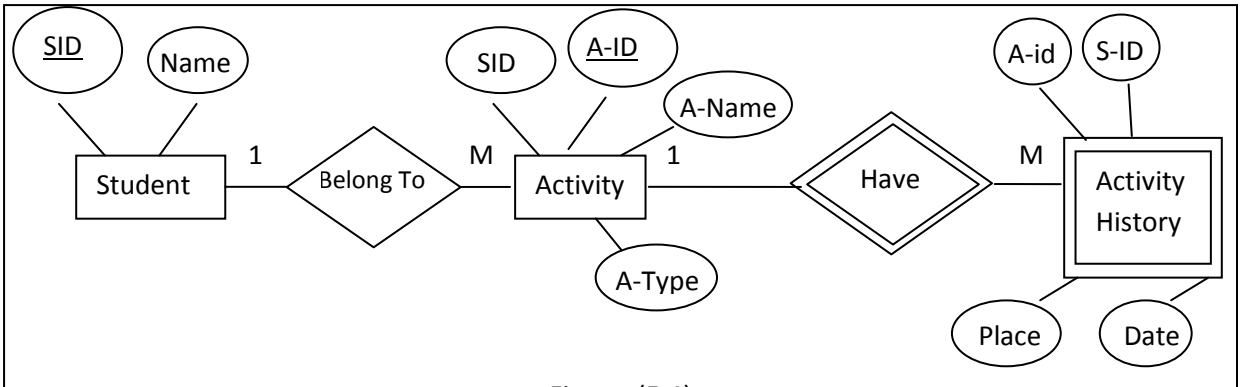



Figure (5.4)
Linking the three entities

In the Activity History, there is no primary key, so it is a weak entity.
The weak entity to be meaningful , it must be part of a one-to-many relationship.
The weak entity is depends on the strong entity , that means if the strong entity is delete then the weak entity must be deleted.

In the example of figure (5.4) if the activity is deleted then all the history for that activity must be deleted.
But if the history is deleted , the activity no need to be deleted.

The DOMINANT entity is the strong entity.
The SUBORDINATE entity is the weak entity.

The weak entity in the ER diagram is indicated by a double outline box , and the corresponding relationship by a double outline diamond.

## 5.4 More Designs Considerations

In this section , we consider how a database designer may select from the wide range of alternatives . Among the decision to be made are the following:

- weather to use an attribute or an entity to represent an object :
  consider the entity ***employee*** with attribute **employee-name** .

  if we want to add telephone number to the employee:

  case 1 : another attribute ,**telephone-number** , is added to the entity.
  Case2 : the ***telephone*** can be consider as an entity in its own with an attribute : **telephone-number** and **location**, ant the two entities are connected with some relation.

  In the first case every employee has one telephone number.
  In the second case , each employee has several telephone numbers

- weather a real world concept is expressed by an entity or by a relationship .
  For example a ***playing*** can be modeled as an entity, or it can be a relation between **player** and ***game,*** with playing-time attribute.

  If every game is played by one player then playing is better be a relationship as in figure (5.5)

  If several players playing the same game then we must replicate the playing information (time) for each player , and the problems of repetition and updating are arise.

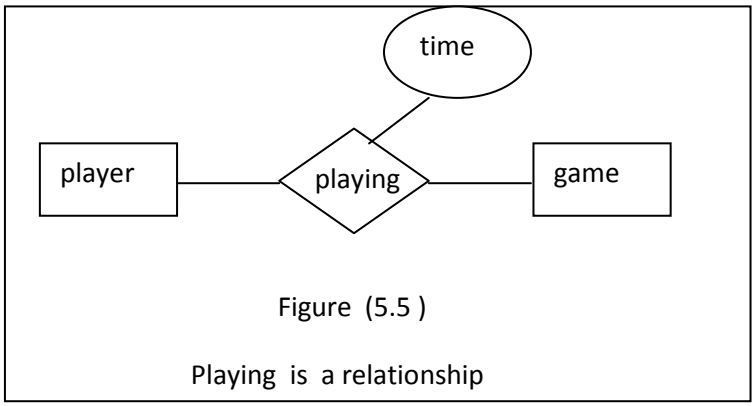



Figure (5.5 )

Playing is a relationship

If the information of the playing is updated then this update must be done to each player.

Playing can be an entity rather than an relation. In this case there will be only one existence of the playing information , and the player and the games is connected with the playing by keys as shown in figure (5.6)

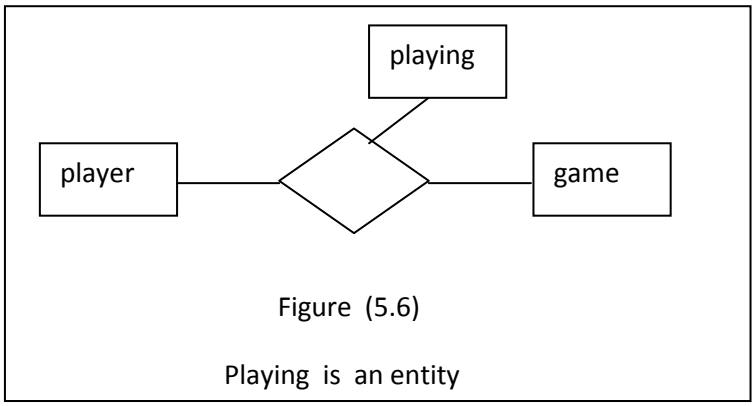



Figure (5.6)

Playing is an entity

## 6 Indexing and Hashing

Many queries reference only a small proportion of the records in a file. For example , the queries " Find all accounts at the Tech branch" reference only a fraction of the account records. It is inefficient for the system to have to read every record and to check the branch names. The system should be able to locate these records directly. To allow that, we design additional structures with the files.

An index for a file in the system works like a catalog for a book in a library, if we are looking for a book , the catalog of the name of the books tells us where to find the book.

To assist us searching the catalog, the names in the catalog listed in an alphabetic order.

There are two basic kinds of indices :

- Ordered indices: such indices are based on a sorted ordering of the values.
- Hash index : such indices are based on some values , these values calculated by a function called hash function.

We often want to have more than one index for the file. Return to the example of the library, there can be a catalog for the names of the books and another catalog for the others of the books and third one for the subjects of the books.

### *6.1 Ordered Indices :*

These are used to gain fast random access to records in a file. Each index structure is associated with a particular ***search key***. The index stores the values of the search keys in sorted order.

The record in the indexed file may themselves be sorted in some way. The file may have several indices of different search key.

**Primary index :** if the file containing the record is sequentially ordered , the index whose search key specifies the sequential order of the file , this index is a primary index for that file.

**Secondary index :** is the index of the file whose search key specifies an order different from the order of the file.

### 6.1.1 : Ordered Primary Index

Figure 6.1 shows an ordered file for ***account*** records.

| | | | |
|---|---|---|---|
| Brighton | A-217 | 750 | |
| Downtown | A-101 | 500 | |
| Downtown | A-110 | 600 | |
| Mianus | A-215 | 700 | |
| Perryridge | A-102 | 400 | |
| Perryridge | A-201 | 900 | |
| Perryridge | A-218 | 700 | |
| Redwood | A-222 | 700 | |
| Round Hill | A-305 | 350 | |

Figure ( 6.1) Sequential file for account records

The file of figure 6.1 is sorted on a search key order, with branch name is used as the search key.

**6.1.1.1 : Indices Types :** There are two types of ordered indices : ***Dense*** and ***Sparse*** indices.

***Dense index*** : an index entry appears for every search key value in the file. The index record contains the search key value and a pointer to the first data record with that search key value, as shown in figure (6.2) for the account file.

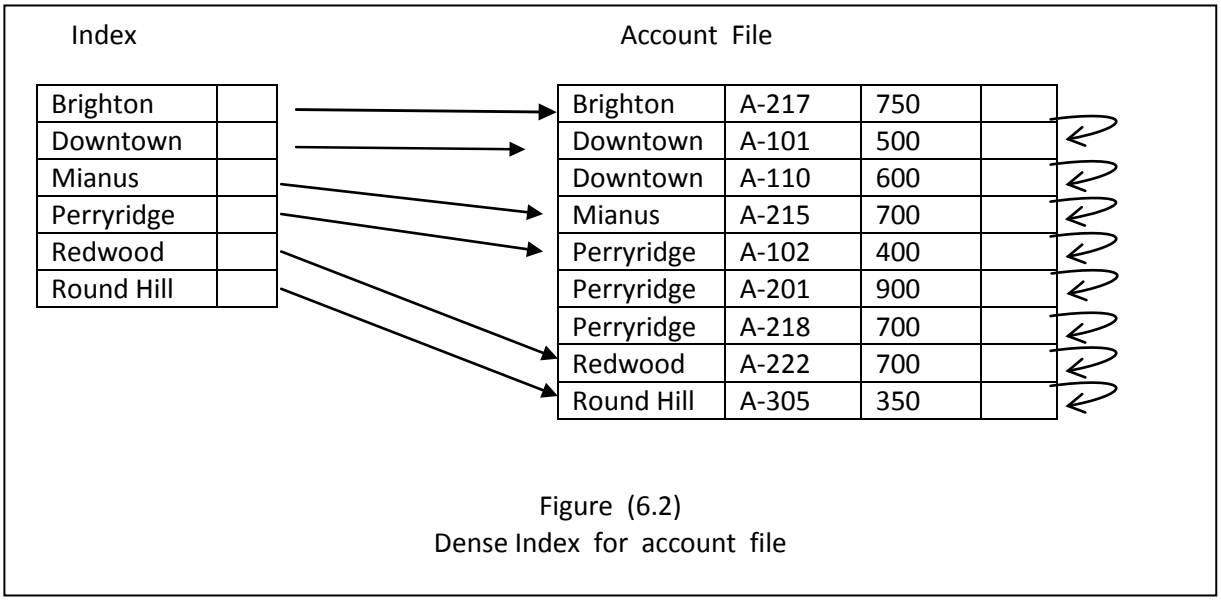



Figure (6.2)
Dense Index for account file

Suppose that we are locking up records for the PEERYIDGE branch using the dense index of figure (6.2), we follow the pointer directly to the first PEERYIDGE record . We process this record and **follow the pointer in that record** to locate the next record in search key (branch name) order. We continue processing records until we encounter a record for a branch other than PEERYIDGE.

***Spars index*** : An index record is created for only some of the values. To locate a record we find the index entry with the largest search key value that is less than or equal to the search key value for which we are locking . We start at the record pointed to by that index entry, and follow the pointer in the file until we find the desired record. As shown in figure (6.3) for the account file.

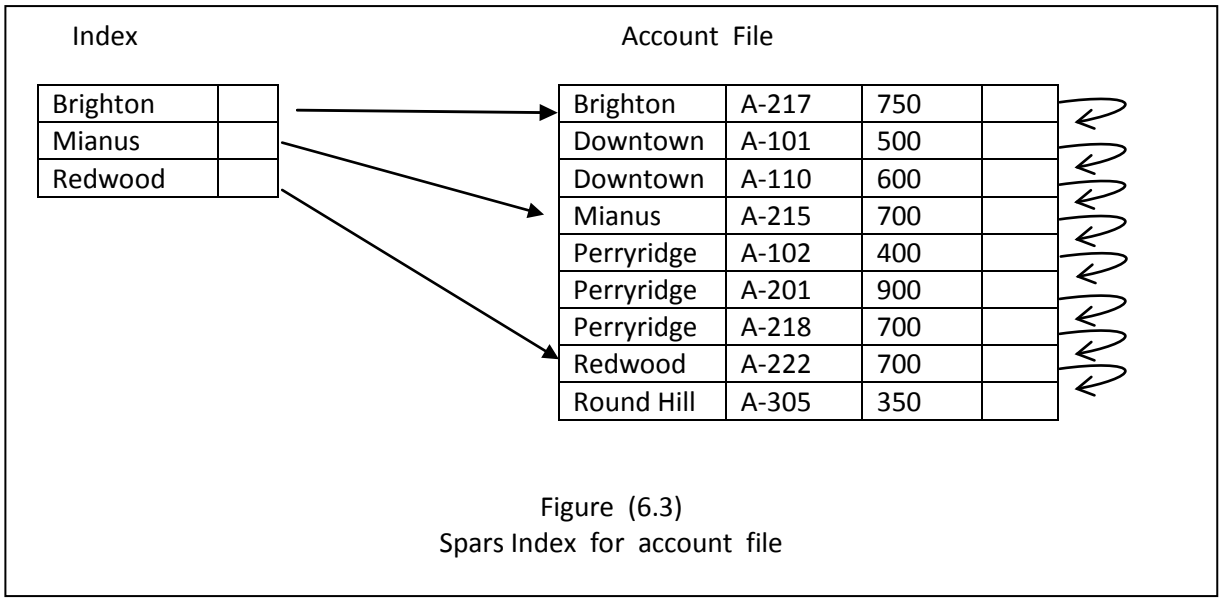



Figure (6.3)
Spars Index for account file

If we are using the spars index of figure (6.3) to find PEERYIDGE , we do not find an index entry for PEERYIDGE in the index. Since the last entry (in alphabetic order) before BEERYIDGE is MIANUS , we follow that pointer.
We then read the account file in sequential order until we find the first PEERYIDEG record and begin processing at that point.

As we have seen , it is faster to locate a record by using a dense index rather than a sparse index., but sparse indices require less space.

**6.1.1.2 Index Update**

Every index must be updated whenever a record is inserted into the file or deleted from the file .

Deletion : To delete a record we first look up the record to be deleted .

In dense indices ,if the deleted record was the only record with its particular search key value, then the search key value is deleted from the index.

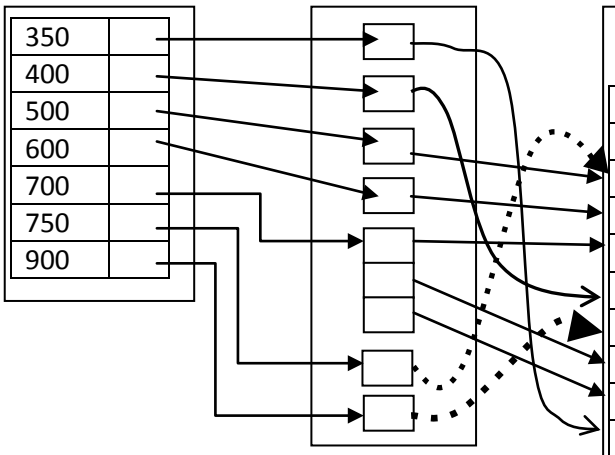
For sparse indices , we delete a key value by replacing its entry (if one exist) in the index with the next search key value . if the next search key value already has an index entry , the entry is deleted instead of being replace.

Insertion : First we perform a lookup using the search key value that appears in the record to be inserted .

In dense index and the value does not appear in the index, the value is inserted in the index.

6.1.2 : Ordered Secondary Index

A secondary index looks like dense primary index except that the records pointed to by successive values in the index are not stored sequentially.
It is not enough to point to just the first record with each search key value because the remaining records with the same search key value could be anywhere in the file, since the records are ordered by a search key of the primary index rather than by the search key of the secondary index. Therefore a secondary index must contain pointers to all the records.
The pointers in the secondary index do not point directly to the file , instead each pointer points to a bucket that contains pointers to the file . Figure (6.4) shows the account file sorted by the balance field, which is not a primary key.




Account File

| | | Balance | |
|---|---|---|---|
| Brighton | A-217 | 750 | |
| Downtown | A-101 | 500 | |
| Downtown | A-110 | 600 | |
| Mianus | A-215 | 700 | |
| Perryridge | A-102 | 400 | |
| Perryridge | A-201 | 900 | |
| Perryridge | A-218 | 700 | |
| Redwood | A-222 | 700 | |
| Round Hill | A-305 | 350 | |

Figure (6.4)
Secondary index for account file on balance as the secondary key

## ***6.2 : Hash Index***

One disadvantage of sequential file is that we must access an index structure to locate data or must use binary search.
File organization based on technique of hashing allow us to avoid accessing an index structure. Hashing also provide a way of constructing indices.

### *6.2.1 Hash File Organization*

In a hash file organization, we obtain the address of the disk block containing the desired record directly by computing a function on the search key value of the record.

Let ( K ) denote the set of all search key values. Let ( B ) denote the set of all blocks addresses. A ***hash function ( H )*** is a function from ( K ) to ( B ) .

To insert a record with search key ( Ki ) , we compute [ H(Ki) ] which gives the address of the block for that record and the record stores in that block.

To perform a lockup on the search key value ( Ki ) , we compute [ H(Ki) ] , then search the block with that address. Suppose that to search keys , K5 and K7 , have the same hash value; that is H(K5)=H(K7), thus we have to check the search key value of every record in the block to find the desired record. Figure (6.5) shows Hash organization for the account file.

Block 0

| | | |
|---|---|---|
| | | |

Block 1

| | | |
|---|---|---|
| | | |

Block 2

| | | |
|---|---|---|
| | | |

Block 3

| | | |
|---|---|---|
| Brightton | A-217 | 750 |
| Round Hill | A-305 | 350 |

Block 4

| | | |
|---|---|---|
| Redwood | A-222 | 700 |

Block 5

| | | |
|---|---|---|
| Perryridge | A-102 | 400 |
| Perryridge | A-201 | 900 |
| Perryridge | A-218 | 700 |
| | | |

Block 6

| | | |
|---|---|---|
| | | |

Block 7

| | | |
|---|---|---|
| Mianus | A-215 | 700 |

Block 8

| | | |
|---|---|---|
| Downtown | A-101 | 500 |
| Downtown | A-110 | 600 |

Block 9

| | | |
|---|---|---|
| | | |

Figure (6.5)
Hash organization of account file using branch name as the key

### *6.2.2 Hash Index*

Hashing can be used for index structure creation. A hash index organize the search keys with their associated pointers into a hash file structure.

We construct a hash index as follow , we apply a hash function on a search key value to identify the block, and store the key and its pointer in that block as shown in figure (6.6).

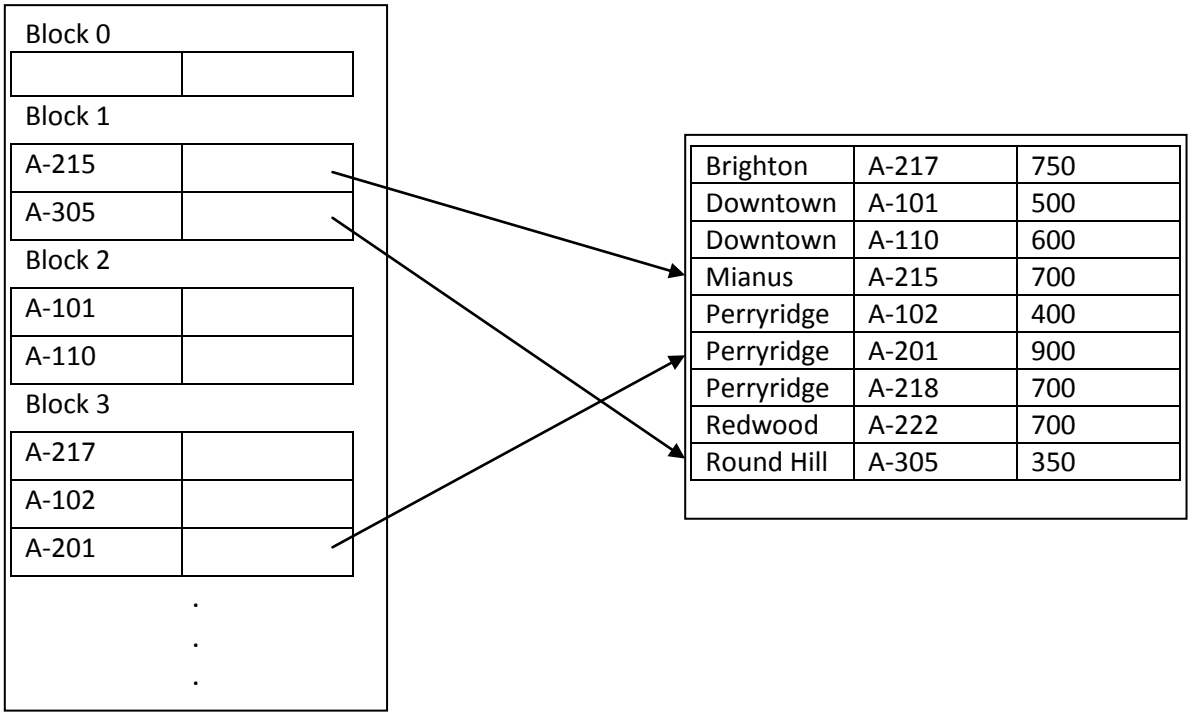



Figure (6.6)
Hash index on search key account-number of account file

## 7 : Normalization

Normalization In relational database , is the process of organizing data to minimize redundancy . The goal of database normalization is to decompose complex relations in order to produce smaller, well-structured relations.

Normalization usually involves dividing large, badly-formed tables into smaller, well-formed tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships

### 7.1 Benefit of Normalization

Free the database of modification anomalies

When an attempt is made to modify (update, insert into, or delete from) a table, undesired side-effects may follow. Not all tables can suffer from these side-effects; rather, ***the side-effects can only arise in tables that have not been sufficiently normalized***. An insufficiently normalized table might have one or more of the following characteristics:

- ***Update anomaly :*** The same information can be expressed on multiple rows; therefore updates to the table may result in logical inconsistencies. For example, each record in an "Employees' Skills" table might contain an Employee ID, Employee Address, and Skill; as shown in figure (7.1). Thus a change of address for a particular employee will potentially need to be applied to multiple records (one for each of his skills). If the update is not carried through successfully, the employee's address is updated on some records but not others—then the table is left in an inconsistent state. Specifically, the table provides conflicting answers to the question of what this particular employee's address is.
- ***Insertion anomaly :*** There are circumstances in which certain facts cannot be recorded at all. For example, each record in a "Faculty and Their Courses" table might contain a Faculty ID, Faculty Name, Faculty Hire Date, and Course Code—thus we can record the details of any faculty member who teaches at least one course, but we cannot record the details of a newly-hired faculty member who has not yet been assigned to teach any courses as shown in figure (7.2).
- ***Deletion anomaly :*** There are circumstances in which the deletion of data representing certain facts necessitates the deletion of data representing completely different facts. The "Faculty and Their Courses" table described in the previous example suffers from this type of anomaly, for if a faculty member temporarily ceases to be assigned to any courses, we must delete the last of the records on which that faculty member appears, effectively also deleting the faculty member as shown in figure (7.3).

**Employees' Skills**

| Employee ID | Employee Address | Skill |
|---|---|---|
| 426 | 87 Sycamore Grove | Typing |
| 426 | 87 Sycamore Grove | Shorthand |
| 519 | 94 Chestnut Street | Public Speaking |
| 519 | 96 Walnut Avenue | Carpentry |

Figure (7.1) An **update anomaly**. Employee 519 is shown as having different addresses on different records

| Faculty ID | Faculty Name | Faculty Hire Date | Course Code |
|---|---|---|---|
| 389 | Dr. Giddens | 10-Feb-1985 | ENG-206 |
| 407 | Dr. Saperstein | 19-Apr-1999 | CMP-101 |
| 407 | Dr. Saperstein | 19-Apr-1999 | CMP-201 |
| 424 | Dr. Newsome | 29-Mar-2007 | ? |

Figure (7.2) An **insertion anomaly**. Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his details cannot be recorded.

| Faculty ID | Faculty Name | Faculty Hire Date | Course Code |
|---|---|---|---|
| 389 | Dr. Giddens | 10-Feb-1985 | ENG-206 |
| 407 | Dr. Saperstein | 19-Apr-1999 | CMP-101 |
| 407 | Dr. Saperstein | 19-Apr-1999 | CMP-201 |

Delete

Figure (7.3) A **deletion anomaly**. All information about Dr. Giddens is lost when he temporarily ceases to be assigned to any courses.

**7.2 : Levels of normalization**

To normalize a database, there are three levels :

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

7.2.1 First Normal Form (1NF)

The role of (1NF) is: No repeating elements or groups of elements, this mean:

- Eliminate duplicative columns from the same table.
- Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

The first rule dictates that we must not duplicate data within the same row of a table. For example consider the table of figure (7.4) of a bank.

| Person ID | Name | City Name | City Number | Account Type | Balance | Account Notes |
|---|---|---|---|---|---|---|
| 123 | Nader | Baghdad | 1 | A | 4556 $ | ****** |
| | | | | B | 7654 $ | ###### |
| | | | | C | 1287 $ | &&&&& |
| 150 | Muna | Basra | 2 | A | 654 $ | ****** |
| | | | | B | 66743 $ | ###### |

Figure (7.4) Accounts table of a bank

The table of figure (7.4) consists of repeated information.

***The meaning of repeated information is all the information belong for a single person. For person "NADER" there are three records repeated for him.***

***For "Muna" there are two record repeated for her.***

The table must be **separated** from the repeated information into **two tables** as shown in figure (7.5). The two tables are joined by the ***Person ID*** as a key. For table 2 the primary key is the combination of the first two fields 'Person Id + Account Type'

Primary Key

| Person ID | Name | City Name | City Number |
|---|---|---|---|
| 123 | Nader | Baghdad | 1 |
| 150 | Muna | Basra | 2 |

Table 1

The Primary Key

| Person ID | Account Type | Balance | Account Notes |
|---|---|---|---|
| 123 | A | 4556 $ | ****** |
| 123 | B | 7654 $ | ###### |
| 123 | C | 1287 $ | &&&&& |
| 150 | A | 654 $ | ****** |
| 150 | B | 66743 $ | ###### |

Table 2

Figure (7.5) the new two tables of (1NF)

7.2.2 Second Normal Form (2NF)

A table that has a concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column only depends upon one part of the concatenated key, then we say that the entire table has failed Second Normal Form and we must create another table to rectify the failure

We looks at the tables with a primary key made from many fields, for each non key fields that do not depends of the primary key (with all the fields of the primary key), this non key fields must be separated in another table.
Table 1 of figure (7.5) have a primary key with single fields, this mean it is in (2NF).

Table 2 of figure (7.5) , field BALANCE depends on both the fields of the primary key, because each balance we must know the person and the account for it.

But "ACCOUNT NOTS" depends only on the second field (Account Type) because the not of the account is the same for all the persons .

Figure (7.6) shows the (2NF)

| Person ID | Name | City Name | City Number |
|---|---|---|---|
| 123 | Nader | Baghdad | 1 |
| 150 | Muna | Basra | 2 |

Table 1 with no change

| Person ID | Account Type | Balance |
|---|---|---|
| 123 | A | 4556 $ |
| 123 | B | 7654 $ |
| 123 | C | 1287 $ |
| 150 | A | 654 $ |
| 150 | B | 66743 $ |

Table 2

| Account Type | Account Notes |
|---|---|
| A | ****** |
| B | ###### |
| C | &&&&& |

Table 3

Table 2 of figure (7.5) is separated into two tables

Figure (7.6) the Second Normal Form (2NF)

7.2.3 : Third Normal Form (3NF)

Third normal form (3NF) is to remove columns that are not dependent upon the primary key. This means we look at the non key fields and try to find a relation between them.

Table 2 and table 3 of figure (7.6) are in the (3NF). But table 1 is not.

In table 1 we find that City Number depends on City Name , not on the key. We separate these two fields on a table and link this new table with the exist one as shown in figure (7.7)

| Person ID | Name | City Number |
|---|---|---|
| 123 | Nader | 1 |
| 150 | Muna | 2 |

| City Number | City Name |
|---|---|
| 1 | Baghdad |
| 2 | Basra |

The new two tables in (3NF) of table1 of figure (7.6)

Figure (7.7)

Example: Find the appropriate tables in 3NF for the following inventory form

Store Inventory form

Store No :
Store Address :
Date of inventory :

| Part No. | Description | Location | Quantity |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

Solution:

First we must find the unnormalized table :

| Store no | Store address | Date of inventory | Location | Part No | Desc | Qty |
|---|---|---|---|---|---|---|
| | | | | | | |

## 1- The 1NF will be:

Table 1

| Store no | Store address | Date of inventory |
|---|---|---|

Primary Key

Table 2

| Store no | Location | Part No | Desc | Qty |
|---|---|---|---|---|

Primary Key

## 2- The 2NF : (non-key fields depends on Key-field)

Table 1 of 1Nf is the 2NF with no change.

Table 2 has the following problem: when we look at the non-key fields, we find that two fields **(Part No) and (Qty)** depends on the entire primary key, but **(Desc) don't depend on any part of the primary key**. instead (Desc) depends on non-key field (Part No).

The 2Nf will be

Table 1

| Store no | Store address | Date of inventory |
|---|---|---|

Primary Key

(table 2 of 1NF will be)

Table 2 on 2NF

| Store no | Location | Part No | Qty |
|---|---|---|---|

Primary Key

Table 3 of 2NF

| Part No | Desc |
|---|---|

Primary Key

When we splits (Desc) on a new table 3 , we put with it (Part No) to connect it with table 2

## 3- The 3NF : (Non-key field depends on non-key field)

3NF is the same as 2NF.

**Problem** : look at table 1 of 3NF (or 2NF) , if any store have many inventory bill, each bill with a deferent date. In table 1 , we must sore a record for each date and repeat the store no ***and the address*** . This is a problem.
It is better to split table 1 into two tables as bellow :

Table 1-1

| Store no | Store address |
|---|---|

Table 1-2

| Store no | Date of inventory |
|---|---|

Another problem arise : in table 2 how we will know each record belong to which date, i.e. how we shall link table 2 with table 1-2 ?

**8 : System Architecture**

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Aspects of computer architecture such as networking, parallelism , and distribution are reflected in the architecture of the database system.

- Networking of computers allows some tasks to be executed on server system and some tasks to be executed on client systems. This division of work has led to the development of *client-server* database systems.

- Parallel processing within a computer system allows database system architecture to be speed up , allowing faster response to transaction, as well as more transaction per second. Queries can be processed in a manner that exploits the parallelism offered by the underling computer system. The need for parallel query processing has led to the development of *parallel* database systems.

- Distributing data across sites or departments in an organization allows those data to reside where they are generated or most needed, but still to be accessible from sites and from other departments. Keeping multiple copies of the database across different sites also allows large organizations to continue their database operations even when one site is affected by a natural disaster, such as flood, fire, of earthquake. Distributed database systems have been developed to handle geographically or administratively distributed data spread across multiple database systems.

There are four types of database system architecture :

- Centralized database system
- Client – Server database system
- Parallel database system
- Distributed database system

**8.1 : Centralized system**

Centralized database systems are those that run on a single computer system and do not interact with other computer systems. Such systems span a rang from single user database systems running on personal computers to high performance database systems running on mainframe systems.

A modern general purpose computer system consists of one to a few CPU's and a number of devices controllers that are connected through a common bus that provides access to shared memory as shown in figure ( 8.1 )

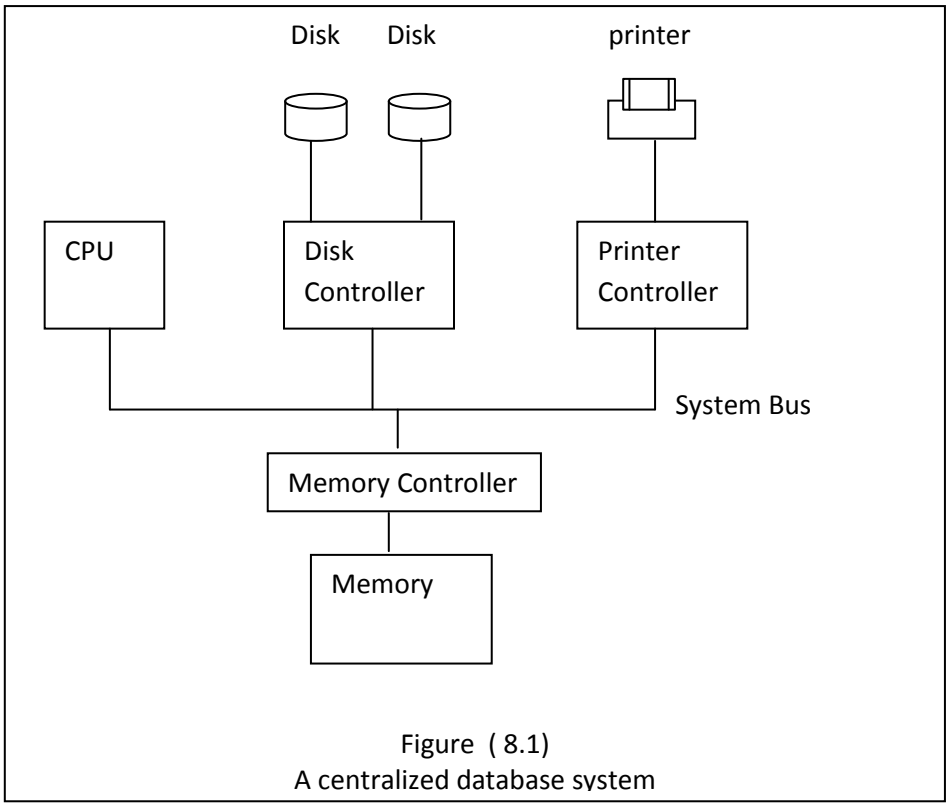



Figure ( 8.1)
A centralized database system

We distinguish two ways in which computers are used : as single user systems and as multiuser systems . Personal computers and workstations fall into the first category. A typical single user system is a desktop unit used by single person, has only one CPU and one or two hard disks, and has an operating system that may support only one user. A typical multiuser system, on the other hand, has more disks and more memory, may have multiple CPU's and has a multiuser operating system. It serves a large number of users who are connected to the system via terminals.

**8.2 : Client – Server database system**

As personal computers have become faster, more powerful and cheaper, there has been a shift away from the centralized system architecture. User interface functionality that used to be handled directly by the centralized systems is increasingly being handled by the personal computers. As a result, centralized systems today act as server system that satisfies requests generated by client systems. The general structure of a client-server system is shown in figure (8.2) . Database functionality can be divided into two parts , the front-end and the back-end as shown in figure (8.3). The back end manage access structure, query evaluation and optimization, concurrency control, and recovery.

The front end of a database system consists of tools such as forms, report writers.

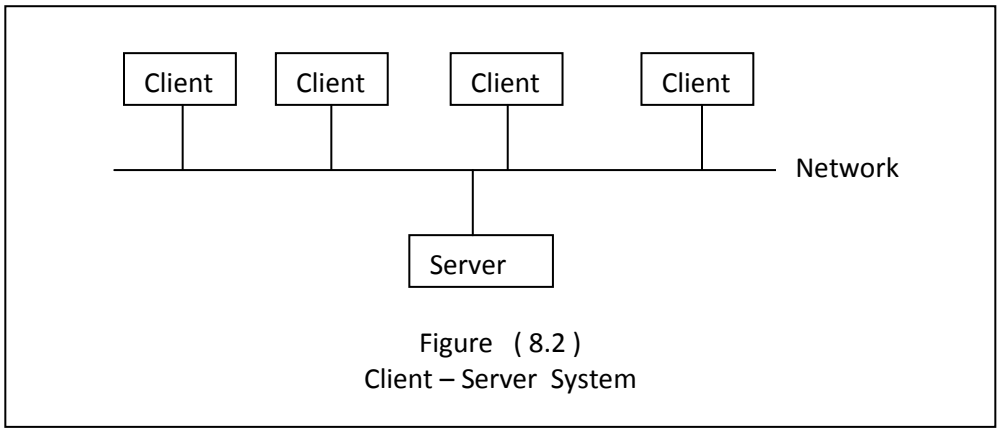



Figure ( 8.2 )
Client – Server System

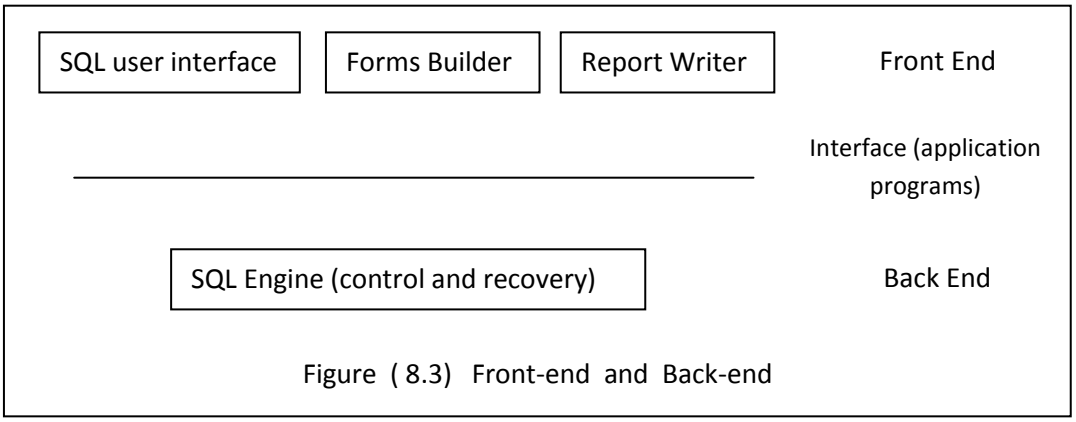



Figure ( 8.3) Front-end and Back-end

The interface between front and back end is through SQL or application programs.

**8.3 : Parallel database system**

Parallel systems improve processing and I/O speeds by using multiple CPUs and disks in parallel. Parallel machines are becoming increasingly common, making the study of parallel database systems correspondingly more important. The driving force behind parallel database systems has been the demands of applications that have to query extremely large databases ( of the order of terabytes – that is, $10^{12}$ bytes) or that have to process an extremely large number of transactions per second ( of the order of thousands of transactions per second) .

Centralized and client-server database systems are not powerful enough to handle such applications.

In parallel processing, many operations are performed simultaneously, as opposed to in serial processing, in which the computational steps are performed sequentially.

There are two main measures of performance of a database system. The first is ***throughput : the number of tasks that can be completed in a given time interval.***

***The second is response time : the amount of time it takes to complete a single task from the time it is submitted.***

A system that processes a large number of small transactions can improve throughput by processing many transactions in parallel.

A system that processes large transactions can improve response time as well as throughput by performing subtasks of each transaction in parallel.

**8.4 : Distributed database system**

In a distributed database system, the database is stored on several computers. The computers in a distributed system communicate with one another through various communication media, such as high-speed networks or telephone lines. They do not share main memory or disks. The computers in a distributed system may vary in size and function, ranging from workstations up to mainframe systems.

The computers in a distributed system are referred to by a number of different names, such as *sites* or *nodes*, depending on the context in which they are mentioned. We mainly use the term *site*, to emphasize the physical distribution of these systems. The general structure of a distributed system is shown in Figure 8.4 .

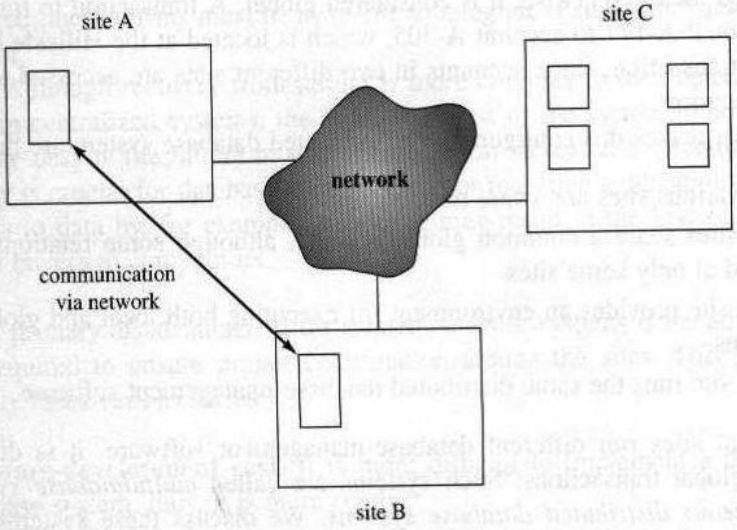



Figure 8.4 : A distributed system.

In distributed systems, the database is geographically separated, and the system hav two types of transaction:

- Local transaction : is the one that access data in the single site at which the transaction was initiated.
- Global transaction : is the one that access data in a site (or many sites) different from the one at which the transaction was initiated.

## 9 : Transactions

9.1 : Definition :A transaction is unit of program execution that accesses and possibly updates various data items.

Often a collection of several operations on the database is considered to be a single unit from the point of view of the user. For example a transfer of funds from a checking account to a saving account is a single operation for the user , but for the database it comprise several operations.

A database system must ensure proper execution of transaction, either the entire transaction is executed or none of it does.

To ensure integrity of the data, the database system maintains the following properties of the transaction:

1- Atomicity : either all operations of the transaction are executed of none are.
2- Consistency : execution of the transaction in isolation to preserve the consistency of the database
3- Isolation : if there are two transaction Ta and Tb ; it appear for Ta that Tb either finished before Ta started , or Tb start execution after Ta finished.
   Thus each transaction unaware of other transactions executing concurrently in the system.
4- Durability : after a transaction completes successfully , the change it has made to the database persist even if there are a system failures.

For example , access to the database is accomplished by the two operation:

- Read (X) : is to read X from the database to a local buffer belonging to the transaction.
- Write (X) : is to write X to the database from the local buffer.

Let Ta be a transaction that transfer 50$ from account A to account B as follow:

```
Ta : Read(A)
     A=A – 50;
     Write (A)
     Read(B)
     B=B + 50;
     Write(B)
```

*Let us consider* ***Atomicity*** : the database system keeps track (on disk) of the old values of any data on which a transaction performs a write, and if the transaction does not complete its execution, the old value is restored to make it appear as though the transaction never executed.

*Let us consider* ***Consistency*** : the consistency required here is the sum of A and B be unchanged.

Without this consistency, money could be created or destroyed by the transaction. If the database is consistent before the execution of the transaction then the database must remain consistent after the execution of the transaction.

*Let us consider* ***Durability*** : we assume that system failure may result of losing data in main memory but data written to disk are never lost. We can guarantee durability by:

1- The update carried by the transaction have been written to disk before the transaction completes.
2- There are sufficient Information about the update to enable the database to reconstruct the update when the system is restarted after the failure.

*Let us consider* ***Isolation*** : while executing the Ta transaction , the Ta transaction execute the third statement (Write (A)) and before executing the last statement (Write (B)), another transaction Tb read the values of A and B , and calculate the sum ( A+B ), then the consistency of transaction Tb will discover an error of the values of A and B.

The solution of the problem of concurrently execution is to execute the transaction in sequence, one after another.

**9.2 : Implementation of atomicity and durability**

The recovery-management system is the one responsible of atomicity and durability. One way to implement them is to use a copy of the database, and a pointer that point at the current copy as shown in figure (9.1).

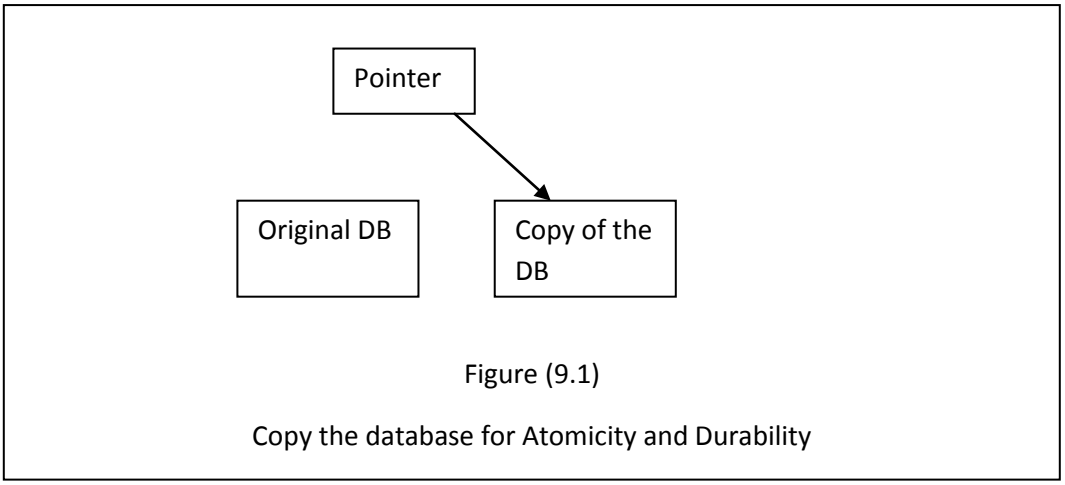



Figure (9.1)

Copy the database for Atomicity and Durability

If a transaction want to update the database , all updates are done on the copy database. If an error occur during the execution of the transaction then the copy is deleted and the system return to the original database.

**9.3 : Concurrent Execution**

Transaction processing system usually allow multiple transaction to run concurrently. When several transaction run concurrently, database consistency can be destroyed.

The database system must control the interaction among the concurrent transaction to prevent them from destroying the consistency of the database.

As an example , consider a banking system which has several accounts, and a set of transactions that access and updates these accounts.

Let T1 and T2 be two transactions that transfer funds from one account to another, they can be executed one after another T1 then T2 as shown in figure (9.2) or T2 then T1 as shown in figure (9.3).

| T1 | T2 |
|---|---|
| Read(A)<br>A=A-50<br>Write(A)<br>Read (B)<br>B=B+50<br>Write (B) | |
| | Read (A)<br>temp=A * 0.1<br>A=A-temp<br>Write (A)<br>Read(B)<br>B=B+temp<br>Write(B) |

Figure (9.2)
Execution of T1 then T2

| T1 | T2 |
|---|---|
| | Read (A)<br>temp=A * 0.1<br>A=A-temp<br>Write (A)<br>Read(B)<br>B=B+temp<br>Write(B) |
| Read(A)<br>A=A-50<br>Write(A)<br>Read (B)<br>B=B+50<br>Write (B) | |

Figure (9.3)
Execution of T2 then T1

When several transactions are executed concurrently , the system may execute one transaction for a little while , then switch to another transaction and execute it for a while then switch back to the first one, and so on as shown in figure (9.4)

| T1 | T2 |
|---|---|
| Read(A)<br>A=A-50<br>Write(A) | |
| | Read (A)<br>temp=A * 0.1<br>A=A-temp<br>Write (A) |
| Read (B)<br>B=B+50<br>Write (B) | |
| | Read(B)<br>B=B+temp<br>Write(B) |

Figure (9.4)
Execution of two transaction

## 10 : Database Security

### 10.1 : Introduction

There is a need to secure computer systems ,and securing data must be part of an overall computer security plan. Growing amounts of sensitive data are being retained in databases and more of these databases are being made accessible via the Internet. As more data is made available electronically, it can be assumed that threats and vulnerabilities to the integrity of that data will increase as well.

### 10.2 : Security objective

The primary objectives of database security are:

- Confidentiality : access control
- Integrity : data corruption
- Availability :

*To preserve the data confidentiality*, enforcing access control policies on the data, these policies are defined on the database management system (DBMS). Access control is to insure that only authorized users perform authorized activities at authorized time.

There are two points of concern in access control:

- Authentication
- Authorization

*To preserve data integrity*, we must guaranty that the data cannot be corrupted in an invisible way.

*Availability* property is to ensure timely and reliable access to the database.

### 10.3 Access control

Access control is the process by which rights and privileges are assigned to users and database objects. Database objects include tables, views, rows and columns. To ensure proper access to the data, authentication and authorization are applied.

Authentication is the process by which you verify that someone is who they claim they are. This usually involves a user name and a password, but can include any other method of demonstrating identity, such as a smart card, voice recognition or fingerprints.

Authentication is equivalent to showing your driver license or your ID.

Authorization is finding out that the person, once identified , is permitted to have the resource. This is usually determined by finding out if that person has paid admission or has a particular level of security clearness.

Authorization is equivalent to checking your name in a list of names, or checking your ticket for something.

For Example, student A may be given login rights to the university database with authorization includes read only for the course listing data table.

## 10.4 Database encryption

Application data security has to deal with several security threats and issues beyond hose handle by SQL authorization. For example , data must be protected while they are being transmitted, data may need be protected from intruders.

Encryption is the process of transferring a clear text (plain text) into disguised text (cipher text) by using a key. Decryption is the process to transfer the cipher text back to plain text as shown in figure 10.1

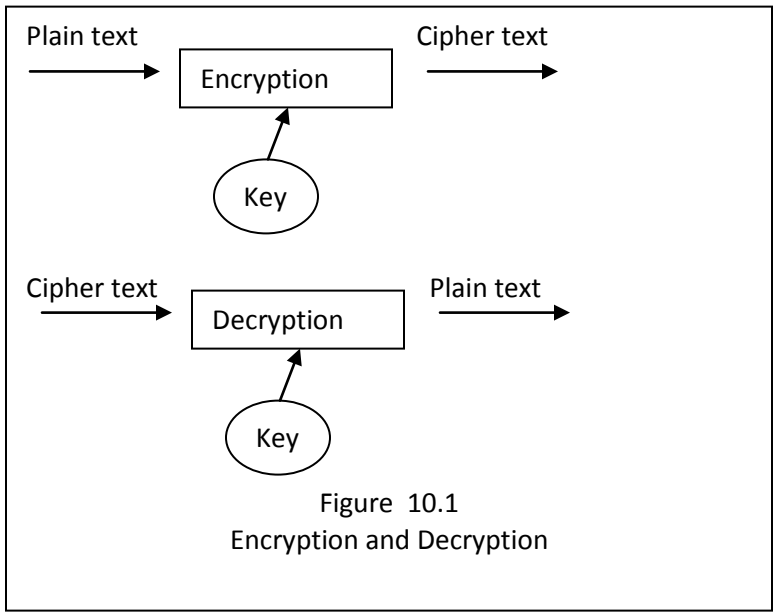



Figure 10.1
Encryption and Decryption

Database encryption refers to the use of encryption techniques to transform a plain text into encrypted database, thus making it unreadable to anyone except those who possess the knowledge of the encryption.

The purpose of database encryption is to ensure the database opacity by keeping the information hidden to any unauthorized persons. Even if someone get through and bypass the access control policies , he/she still unable to read the encrypted database.

There are a vast number of techniques for the encryption of data. An example for simple encryption techniques is to substitute of each character with the next character in the alphabet. ***For example 'Perryridge' becomes 'Qfsszsjehf'.***

A good encryption techniques has the following properties:

1- It is relatively simple for authorized users to encrypt and decrypt data.
2- It depends not on the security of the algorithm, but rather on a parameter of the algorithm called the encryption key.
3- Its encryption key is extremely difficult for an intruder to determine.

## 10.5 Database encryption level

### 10.5.1 Application-Level Encryption:

When you encrypt information at the application level, you can protect sensitive data. In many ways the application is the obvious place to encrypt and decrypt data because the application knows exactly which data is sensitive and can apply protection selectively.
You can task a given application with encrypting its own data. This encryption capability is designed into the application itself. By the time the database receives the data, it has already been encrypted and then stored in the database in this encrypted state. As the traffic travels from the application to the database, the data can also be encrypted across the network.

### 10.5.2 Database Encryption Level:

In this case the information is encrypted in the database. As an example, we'll discuss ***Oracle Advanced Security's transparent data encryption (TDE)***, which automatically encrypts and decrypts the data stored in the database and provides this capability without having to write additional code.
With TDE, the encryption process and associated encryption keys are created and managed by the database. This is transparent to database users who have authenticated to the database. At the operating system, however, attempts to access database files return data in an encrypted state. Therefore, for any operating system level users, the data remains inaccessible. Additionally, because the database is doing the encryption, there is no need to change the application(s), and there is a minimal performance overhead when changes occur in the database. TDE is designed into the database itself: Oracle has integrated the TDE syntax with its data definition language (DDL).
If you encrypt on the database, that means the data is sent to and from the database in unencrypted form. This potentially allows for snooping/tampering between the application and the encryption routines on the database.

**11 : Fundamental of relational algebra:**

Relational algebra consists of a set of operations that takes one or two relations as input and produce a new relation as their result. In the relational algebra, symbols are used to denote an operation.

- For SELECT we use the sigma letter σ . The relation(table name) is written in parentheses:
  Select * from Loan where B_name="Perryridge:
  will be : $\sigma_{B\_name="Perryridge}$ (Loan)

- For projection we use the pi letter π . projection mean select some fields from the table , not all the fields.
  If we have : Student (S-Id,S-Name,S-Address)
  To display only the names → Select S-Name from student
  will be : $\pi_{S\text{-}Name}$ (Student)

- Selection (σ) and Projection (π) can be used together to select some of the fields with a condition.
  To display only the names whose address in Baghdad from student table:
  ➔ Select S-Name from student where S-Address='Baghdad'

  Will be : $\pi_{S\text{-}Name}$ ($\sigma_{s\text{-}address='Baghdad'}$ (Student) )

- Cartesian product (Cross Join) between two tables denoted by X.
  It is used to combine information from any two relations. It will produce a tuple from each possible pair of tuples: one from the first table and one from the second.

  Ex 1 : To Cross Join between Student and Class ; with the condition only for level 2
  ➔ $\sigma_{LVL=2}$ (Student X Class)

  Example 2: we have two tables as shown in figure (11.1)

tables1 :
Loan (loan-no ,branch-name, amount)

| loan-no | Branch-name | amount |
|---|---|---|
| L-11 | Round hill | 900 |
| L-14 | Downtown | 1500 |
| L-15 | Perryridge | 1500 |
| L-16 | Perryridge | 1300 |
| L-17 | Downtown | 1000 |
| L-23 | Redwood | 2000 |
| L-93 | Minus | 500 |
| | | |

tables2 :
Borrower (customer-name, loan-number)

| Customer-name | Loan-number |
|---|---|
| Adams | L-16 |
| Curry | L-93 |
| Hayes | L-15 |
| Jackson | L-14 |
| Jones | L-17 |
| Smith | L-11 |
| Smith | L-23 |
| Williams | L-17 |

Figure (11.1)

If we want to know the names of all customers who have a lone at the Perryridge branch. So if we write : $\sigma_{\text{branch-name=' Perry ridge'}}$ (Borrower X Loan) ; the result of this cross join shown in figure (11.2).

| customer-name | Borrower. loan-number | Loan. loan-number | branch-name | amount |
|---|---|---|---|---|
| Adams | L-16 | L-15 | Perry ridge | 1500 |
| Adams | L-16 | L-16 | Perry ridge | 1300 |
| Curry | L-93 | L-15 | Perry ridge | 1500 |
| Curry | L-93 | L-16 | Perry ridge | 1300 |
| Hayes | L-15 | L-15 | Perry ridge | 1500 |
| Hayes | L-15 | L-16 | Perry ridge | 1300 |
| Jackson | L-14 | L-15 | Perry ridge | 1500 |
| Jackson | L-14 | L-16 | Perry ridge | 1300 |
| Jones | L-17 | L-15 | Perry ridge | 1500 |
| Jones | L-17 | L-16 | Perry ridge | 1300 |
| Smith | L-11 | L-15 | Perry ridge | 1500 |
| Smith | L-11 | L-16 | Perry ridge | 1300 |
| Smith | L-23 | L-15 | Perry ridge | 1500 |
| Smith | L-23 | L-16 | Perry ridge | 1300 |
| Williams | L-17 | L-15 | Perry ridge | 1500 |
| Williams | L-17 | L-16 | Perry ridge | 1300 |

Figure (11.2)
the result of cross join $\sigma_{\text{branch-name=' Perry ridge'}}$ (Borrower X Loan)

THE RESULT IS NOT RIGHT!!.

The Cross Join links every record from Borrower with all the records of Loan who have Perry ridge in branch-name.
The correct answer will be :

$\sigma_{\text{borrower. loan-number=loan.loan-number}}$ ( $\sigma_{\text{branch-name=' Perry ridge'}}$ (Borrower X Loan) )

| | | | | |
|---|---|---|---|---|
| Adams | L-16 | L-16 | Perry ridge | 1300 |
| Hayes | L-15 | L-15 | Perry ridge | 1500 |

- The natural join operation
  It allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the join symbol (⋈). The natural join operation do the following:
  - Cartesian product of its arguments ( ex: two tables)
  - Perform selection forcing equality on those attributes that appear in both tables.
  - Remove duplicate attributes.

  ***Example 11.1***: Consider the borrower and loan tables in figure (11. 1 ), to find the names of all customers who have a loan at the bank, and find the amount of the loan:

  $\Pi_{\text{customer-name,loan-number,amount}}(\text{borrower} \bowtie \text{loan})$

  Because borrower and loan tables both have the attribute loan-number, the natural join operation considers only pairs of tuples from the two tables that have the same value on loan-number. The result will be as shown in figure (11.3) :

| Customer-name | Loan-number | amount |
|---|---|---|
| Adams | L-16 | 1300 |
| Curry | L-93 | 500 |
| Hayes | L-15 | 1500 |
| Jackson | L-14 | 1500 |
| Jones | L-17 | 1000 |
| Smith | L-11 | 900 |
| Smith | L-23 | 2000 |
| Williams | L-17 | 1000 |

Figure (11.3)
The result of natural join $\Pi_{\text{customer-name,loan-number,amount}}(\text{borrower} \bowtie \text{loan})$

***Example 11.2*** : Notes: the natural join usually required that the two relations must have at least one common attribute, but if this constraint is omitted, and the two relations have no common attributes, then the natural join becomes exactly the Cartesian product as shown in figure (11.4)

***Car***

| CarModel | CarPrice |
|---|---|
| CarA | 20,000 |
| CarB | 30,000 |

***Boat***

| BoatModel | BoatPrice |
|---|---|
| Boat1 | 10,000 |
| Boat2 | 40,000 |

Car ⋈ ***Boat***

| CarModel | CarPrice | BoatModel | BoatPrice |
|---|---|---|---|
| CarA | 20,000 | Boat1 | 10,000 |
| CarA | 20,000 | Boat2 | 40,000 |
| CarB | 30,000 | Boat1 | 10,000 |
| CarB | 30,000 | Boat2 | 40,000 |

Figure (11.4)
Natural join becomes a cross join because there are no common attribute

***Example 11.3*** : find the names of all branches with customers who have an account in the bank and who live in Harrison for the relations shown in figure (11.5).

Customer relation

| Customer-name | Customer-street | Customer-city |
|---|---|---|
| Adams | Spring | Pittsfield |
| Brooks | Senator | Brooklyn |
| Curry | North | Rye |
| Glenn | Sand hill | Woodside |
| Green | Walnut | Stamford |
| Hayes | Main | Harrison |
| Johnson | Alma | Palo alto |
| Jones | Main | Harrison |
| Lindsay | Park | Pittsfield |
| Smith | North | Rye |
| Turner | Putnam | Stamford |
| Williams | Nassau | Princeton |

Account relation

| Account-number | Branch-name | Balance |
|---|---|---|
| A-101 | Downtown | 500 |
| A-215 | Minus | 700 |
| A-102 | Perryridge | 400 |
| A-305 | Round hill | 350 |
| A-201 | Brighton | 900 |
| A-222 | Redwood | 700 |
| A-217 | Brighton | 750 |

Depositor relation

| Customer-name | Account-number |
|---|---|
| Hayes | A-102 |
| Johnson | A-101 |
| Johnson | A-201 |
| Jones | A-217 |
| Lindsay | A-222 |
| Smith | A-215 |
| Turner | A-305 |

Figure ( 11.5 ) . Three Tebles

The result will be : $\Pi_{\text{branch-name}}(\sigma_{\text{customer-city}}=\text{'Harrison'}(\text{customer} \bowtie \text{account} \bowtie \text{depositor}))$

| Branch-name |
|---|
| Perryridge |
| Brighton |

## 12 : Query processing

Query processing refers to the number of activities involved in extracting data from a database. The steps involved in query processing are :

1- Parsing and translation
2- Optimization
3- Evaluation

### 12.1- Parsing and translation

Before query processing can begin, the system must translate the query into a usable form. A language such as SQL is suitable for human use , but it is not suitable for the internal representation of the query in the system, thus the query must translated into its internal form, and this is the work of the ***PARSER***.

The ***PARSER*** check for :

- The syntax of the query
- The relation names appearing in the query are exist in the database
- Generate the relational-algebra expression.

### 12.2- Optimization

The **query optimizer** is the component of a database management system that attempts to determine the most efficient way to execute a query.

The optimizer considers the possible query plans for a given input query, and attempts to determine which of those plans will be the most efficient. *[A **query plan** (or **query execution plan**) is an ordered set of steps used to access or modify information in a database.]*

Cost-based query optimizers assign an estimated "cost" to each possible query plan, and choose the plan with the smallest cost.

[Costs are used to estimate the runtime cost of evaluating the query, in terms of the number of I/O operations required, the CPU requirements(CPU time to execute a query), the cost of memory used for the query and the cost of communication (in distributed or client-server DB)]

#### 12.2-1 : Equivalent expression

To find the least-costly query evaluation plan, the optimizer generates alternative plan (by generating alternative algebra expression) that produce the same result but with deferent costs.

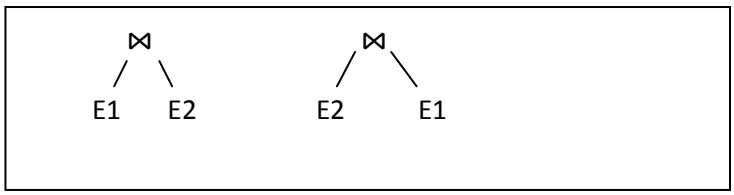
The rules of equivalence are :

- Rule (1) : Selection operations are commutative

  $\sigma_a(\sigma_b(E)) = \sigma_b(\sigma_a(E))$

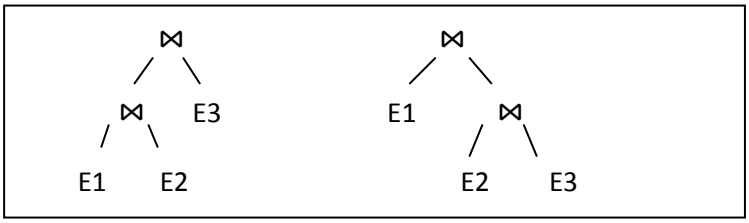
- Rule (2): Natural Join operations are commutative

  $E1 \bowtie E2 = E2 \bowtie E1$




- Rule(3): Natural join operations are associative

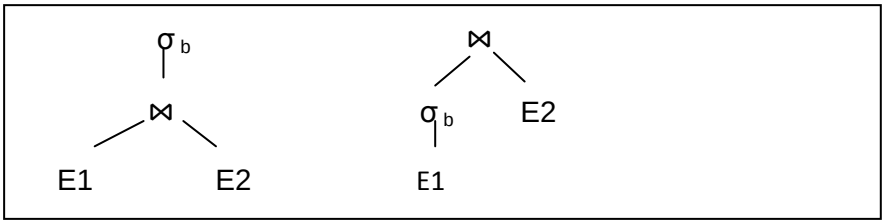
  $(E1 \bowtie E2) \bowtie E3 = E1 \bowtie (E2 \bowtie E3)$




- Rule(4) :The selection operation distributes over the join operation under the following two condition:

  a- It distribute when all the attributes in selection condition ($_b$) involves only the attributes of one of the expressions (say E1) being joined.

  $\sigma_b (E1 \bowtie E2) = (\sigma_b(E1)) \bowtie E2$




  b- It distributes when selection condition ($_{b1}$) involves only the attributes of E1 And ($_{b2}$) involves only the attributes of E2

  $\sigma_{b1 \wedge b2} (E1 \bowtie E2) = (\sigma_{b1} (E1)) \bowtie (\sigma_{b2} (E2))$

***Example 12.1*** : consider the relational algebra

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}city}\text{='Brooklyn'}(branch \bowtie (account \bowtie depositor)))$$

This expression construct a large intermediate relation , branch ⋈ account ⋈ depositor. However we are interesting in only a few tuples of this relation (branch in Brooklyn) and one attribute of the relation (customer-name).

Since we are concerned with only tuples those in Brooklyn in the ***branch*** relation, we do not need to consider those tuples that do not have branch-city="Brooklyn" from the ***branch*** relation.

By reducing the number of tuples of the branch relation that we need to access, we reduce the size of the intermediate result. By using rule (4.a) our query is now represented by the relational expression: (figure 12.1)

$$\Pi_{\text{customer-name}}((\sigma_{\text{branch-city}}=\text{'Brooklyn'}(\text{branch})) \bowtie (\text{account} \bowtie \text{depositor}))$$

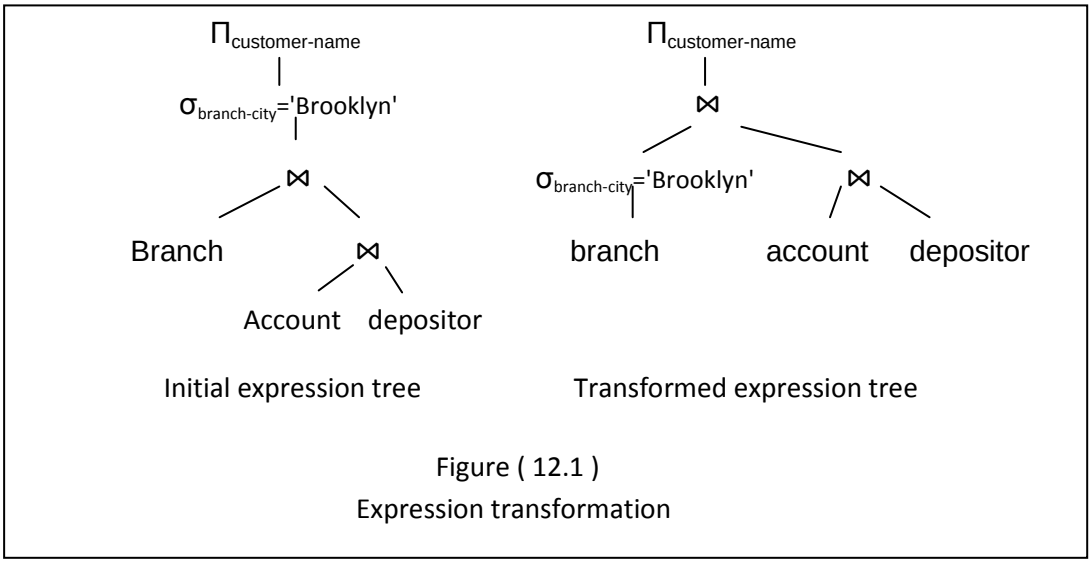



Figure ( 12.1 )
Expression transformation

***Example12. 2***: We have three relations

Branch (branch-name,branch-city,assets)
Account (account-number,branch-name,balance)
Depositor (customer-name,account-number)

To find customer names in Brooklyn and have balance over 1000$. The relational algebra is:

$$\Pi_{\text{customer-name}}(\sigma_{\text{branch-city}}=\text{'Brooklyn'} \wedge \text{balance}>1000 (\text{branch} \bowtie (\text{account} \bowtie \text{depositor})))$$

We cannot apply rule (4) directly because the condition involves attributes of both the ***branch and account*** relations. We can apply rule (3) to transform the join (branch ⋈ ( account ⋈ depositor))

Into (branch ⋈ account ) ⋈ depositor) :

$$\Pi_{\text{customer-name}}(\sigma_{\text{branch-city='Brooklyn'} \wedge \text{balance}>1000} ((\text{branch} \bowtie \text{account}) \bowtie \text{depositor}))$$

Then using rule (4.a) to take ***depositor relation*** out from the condition:

$$\Pi_{\text{customer-name}}((\sigma_{\text{branch-city='Brooklyn'} \wedge \text{balance}>1000} (\text{branch} \bowtie \text{account})) \bowtie \text{depositor})$$

Then using rule (4.b) :

$$\Pi_{\text{customer-name}}( (\sigma_{\text{branch-city='Brooklyn'}} \text{ branch} \bowtie \sigma_{\text{balance}>1000} \text{ account}) \bowtie \text{depositor})$$

12.2-2 : Disk I/O cost

In the database, the cost to access data from disk is important, since disk accesses are slow compared to in memory operation. Disk access measured by taking into account:

* Number of disk seeks ( average-seek-cost )
* Number of blocks transfers from disk ( average-block-read-cost )

If the disk subsystem takes an average of

*tT* – seconds to transfer one block of data
*tS* – seconds for one seek (block access time)

then the operation of transfers N blocks and performs S seeks would take :

$$N * tT + S * tS \quad seconds$$

When calculating disk I/O cost, some system need one seek to transfer many block. For example if there are 10 blocks need to be transfer from disk to memory with one seek , then the time will be tS + 10 * tT.

12.2-3 :Projection Example

Projections produce a result tuple for every argument tuple. Change in the output size is the change in the length of tuples .

Let's take a relation 'R' : R(a, b, c), the number of tuples in this relation are (20,000 tuples).
Each Tuple (190 bytes size) : header = 24 bytes, a = 8 bytes, b = 8 bytes, c = 150 bytes.
Each Block (1024): header = 24 bytes
We can fit 5 tuples into 1 block

- 5 tuples * 190 bytes(size of the tuple) = 950 bytes can fit into 1 block
- For 20,000 tuples, we would require ***4,000*** blocks (20,000 / 5 tuples per block)

With a projection resulting in elimination of column c (150 bytes), we could estimate that each tuple would decrease to 40 bytes (190 – 150 bytes)
Now, the new estimate will be 25 tuples in 1 block. (25 tuples * 40 byte= 1000)

- 25 tuples * 40 bytes/tuple = 1000 bytes will be able to fit into 1 block
- With 20,000 tuples, the new estimate is 800 blocks (20,000 tuples / 25 tuples per block = ***800*** blocks)

Result is reduction by a factor of 5

## 12.3- Evaluation

Query evaluation is the process of executing the plan for that query and return the result to query.
The ***query-execution engine*** is the subsystem of the DBMS that execute the query plan.