Database System

Database systems are designed to manage large data set in an organization. The data management involves both definition and the manipulation of the data which ranges from simple representation of the data to considerations of structures for the storage of information. The data management also consider the provision of mechanisms for the manipulation of information.

Today, Databases are essential to every business. They are used to maintain internal records, to present data to customers and clients on the World-WideWeb, and to support many other commercial processes. Databases are likewise found at the core of many modern organizations.

The power of databases comes from a body of knowledge and technology that has developed over several decades and is embodied in specialized software called a database management system, or DBMS. A DBMS is a powerful tool for creating and managing large amounts of data efficiently and allowing it to persist over long periods of time, safely. These systems are among the most complex types of software available.

Thus, for our question: What is a database? In essence a database is nothing more than a collection of shared information that exists over a long period of time, often many years. In common dialect, the term database refers to a collection of data that is managed by a DBMS.

Thus, the DB course is about:

- ✓ How to organize data
- ✓ Supporting multiple users
- ✓ Efficient and effective data retrieval ④ Secured and reliable storage of data
- ✓ Maintaining consistent data
- ✓ Making information useful for decision making

Data management passes through the different levels of development along with the development in technology and services. These levels could best be described by categorizing the levels into three levels of development. Even though there is an advantage and a problem overcome at each new level, all methods of data handling are in use to some extent. The major three levels are;

1. **Manual Approach**
2. **Traditional File Based Approach**
3. **Database Approach**

1. Manual Approach

In the manual approach, data storage and retrieval follow the primitive and traditional way of information handling where cards and paper are used for the purpose.

- ✓ Files for as many event and objects as the organization are used to store information.
- ✓ Each of the files containing various kinds of information is labelled and stored in one or more cabinets.
- ✓ The cabinets could be kept in safe places for security purpose based on the sensitivity of the information contained in it.
- ✓ Insertion and retrieval are done by searching first for the right cabinet then for the right the file then the information.
- ✓ One could have an indexing system to facilitate access to the data

Limitations of the Manual approach

- ✓ Prone to error
- ✓ Difficult to update, retrieve, integrate
- ✓ You have the data but it is difficult to compile the information
- ✓ Limited to small size information
- ✓ Cross referencing is difficult

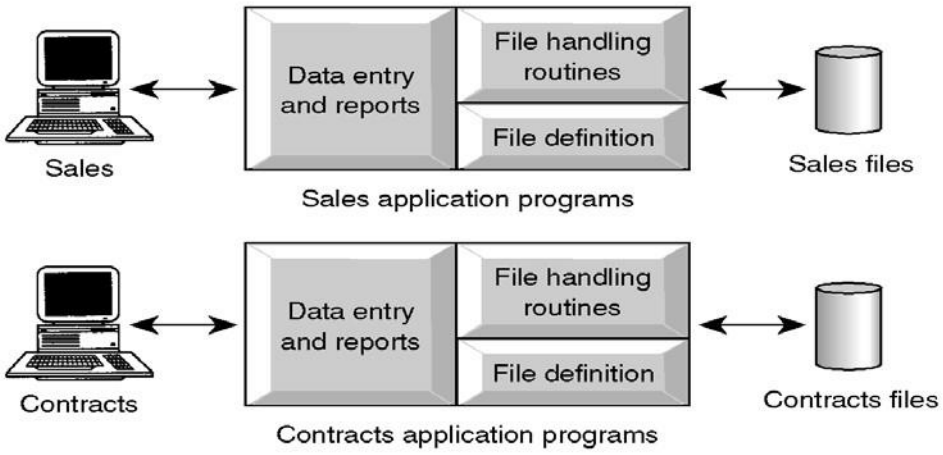
An alternative approach of data handling is a computerized way of dealing with the information. The computerized approach could also be either decentralized or centralized base on where the data resides in the system.

2. Traditional File Based Approach

After the introduction of Computer for data processing to the business community, the need to use the device for data storage and processing increase. There were, and still are, several computer applications with file-based processing used for the purpose of data handling. Even though the approach evolved over time, the basic structure is still similar if not identical.

- ✓ File based systems were an early attempt to computerize the manual filing system.

- ✓ This approach is the decentralized computerized data handling method.
- ✓ A collection of application programs perform services for the end-users. In such systems, every application program that provides service to end users define and manage its own data
- ✓ Such systems have number of programs for each of the different applications in the organization.
- ✓ Since every application defines and manages its own data, the system is subjected to serious data duplication problem.
- ✓ File, in traditional file based approach, is a collection of records which contains logically related data.



Sales Files

Property_for_Rent(Property Number, Street, Area, City, Post Code, Property Type, Number of Rooms, Monthly Rent, Owner Number)

Owner(Owner Number, First Name, Last Name, Address, Telephone Number)

Renter(Renter Number, First Name, Last Name, Address, Telephone Number, Preferred Type, Maximum Rent)

Contracts Files

Lease(Lease Number, Property Number, Renter Number, Monthly Rent, Payment Method, Deposit, Paid, Rent Start Date, Rent Finish Date, Duration)

Property_for_Rent(Property Number, Street, Area, City, Post Code, Monthly Rent)

Renter(Renter Number, First Name, Last Name, Address, Telephone Number)

Limitations of the Traditional File Based approach

As business application become more complex demanding more flexible and reliable data handling methods, the shortcomings of the file-based system became evident. These shortcomings include, but not limited to:

- ✓ Separation or Isolation of Data: Available information in one application may not be known.

- ✓ Limited data sharing
- ✓ Lengthy development and maintenance time
- ✓ Duplication or redundancy of data
- ✓ Data dependency on the application
- ✓ Incompatible file formats between different applications and programs creating inconsistency.
- ✓ Fixed query processing which is defined during application development The limitations for the traditional file based data handling approach arise from two basic reasons.
    1. Definition of the data is embedded in the application program which makes it difficult to modify the database definition easily.
    2. No control over the access and manipulation of the data beyond that imposed by the application programs.

The most significant problem experienced by the traditional file based approach of data handling is the **"update anomalies"**. We have three types of update anomalies;

1. **Modification Anomalies**: a problem experienced when one ore more data value is modified on one application program but not on others containing the same data set.
2. **Deletion Anomalies**: a problem encountered where one record set is deleted from one application but remain untouched in other application programs.
3. **Insertion Anomalies**: a problem encountered where one can not decide whether the data to be inserted is valid and consistent with other similar data set.

3. Database Approach

Following a famous paper written by Ted Codd in 1970, database systems changed significantly. Codd proposed that database systems should present the user with a view of data organized as tables called relations. Behind the scenes, there might be a complex data structure that allowed rapid response to a variety of queries. But, unlike the user of earlier database systems, the user of a relational system would not be concerned with the storage structure. Queries could be expressed in a very high-level language, which greatly increased the efficiency of database programmers. The database approach emphasizes the integration and sharing of data throughout the organization.

Thus, in Database Approach:

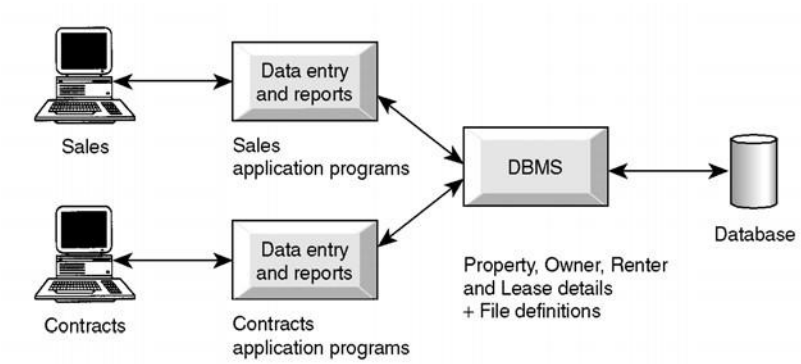
- ✓ Database is just a computerized record keeping system or a kind of electronic filing cabinet.
- ✓ Database is a repository for collection of computerized data files.
- ✓ Database is a shared collection of logically related data designed to meet the information needs of an organization. Since it is a shared corporate resource, the database is integrated with minimum amount of or no duplication.
- ✓ Database is a collection of logically related data where these logically related data comprises entities, attributes, relationships, and business rules of an organization's information.
- ✓ In addition to containing data required by an organization, database also contains a description of the data which called as "**Metadata**" or "**Data Dictionary**" or "**Systems Catalogue**" or "**Data about Data**".
- ✓ Since a database contains information about the data (metadata), it is called a self *descriptive collection* on integrated records.
- ✓ The purpose of a database is to store information and to allow users to retrieve and update that information on demand.
- ✓ Database is deigned once and used simultaneously by many users.
- ✓ Unlike the traditional file based approach in database approach there is program data independence. That is the separation of the data definition from the application. Thus application is not affected by changes made in the data structure and file organization.

  ✓ Each database application will perform the combination of: Creating database, Reading, Updating and Deleting data.

Benefits of the database approach

- ✓ *Data can be shared*: two or more users can access and use same data instead of storing data in redundant manner for each user.
- ✓ *Improved accessibility of data*: by using structured query languages, the users can easily access data without programming experience.
- ✓ *Redundancy can be reduced*: isolated data is integrated in database to decrease the redundant data stored at different applications.
- ✓ *Quality data can be maintained*: the different integrity constraints in the database approach will maintain the quality leading to better decision making
- ✓ *Inconsistency can be avoided*: controlled data redundancy will avoid inconsistency of the data in the database to some extent.

- ✓ *Transaction support can be provided*: basic demands of any transaction support systems are implanted in a full scale DBMS.
- ✓ *Integrity can be maintained*: data at different applications will be integrated together with additional constraints to facilitate shared data resource.
- ✓ *Security majors can be enforced*: the shared data can be secured by having different levels of clearance and other data security mechanisms.
- ✓ *Improved decision support*: the database will provide information useful for decision making.
- ✓ *Standards can be enforced*: the different ways of using and dealing with data by different unite of an organization can be balanced and standardized by using database approach.
- ✓ *Compactness*: since it is an electronic data handling method, the data is stored compactly (no voluminous papers).
- ✓ *Speed*: data storage and retrieval is fast as it will be using the modern fast computer systems.
- ✓ *Less labour*: unlike the other data handling methods, data maintenance will not demand much resource.
- ✓ *Centralized information control:* since relevant data in the organization will be stored at one repository, it can be controlled and managed at the central level.



Property_for_Rent(Property Number, Street, Area, City, Post Code, Property Type, Number of Rooms, Monthly Rent, Owner Number)

Owner(Owner Number, First Name, Last Name, Address, Telephone Number)

Renter(Renter Number, First Name, Last Name, Address, Telephone Number), Preferred Type, Maximum Rent)

Lease(Lease Number, Property Number, Renter Number, Payment Method, Deposit, Paid, Rent Start Date, Rent Finish Date)

Limitations and risk of Database Approach

- ✓ Introduction of new professional and specialized personnel.
- ✓ Complexity in designing and managing data
- ✓ The cost and risk during conversion from the old to the new system
- ✓ High cost incurred to develop and maintain
- ✓ Complex backup and recovery services from the users perspective
- ✓ Reduced performance due to centralization
- ✓ High impact on the system when failure occur

Database Management System (DBMS)

Database Management System (DBMS) is a Software package used for providing EFFICIENT, CONVENIENT and SAFE MULTI-USER (many people/programs accessing same database, or even same data, simultaneously) storage of and access to MASSIVE amounts of PERSISTENT (data outlives programs that operate on it) data. A DBMS also provides a systematic method for creating, updating, storing, retrieving data in a database. DBMS also provides the service of controlling data access, enforcing data integrity, managing concurrency control, and recovery. Having this in mind, a full-scale DBMS should at least have the following services to provide to the user.

1. Data ***storage***, ***retrieval*** and ***update*** in the database
2. A user accessible ***catalogue***
3. ***Transaction support service***: ALL or NONE transaction, which minimize data inconsistency.
4. ***Concurrency Control Services***: access and update on the database by different users simultaneously should be implemented correctly.
5. ***Recovery Services***: a mechanism for recovering the database after a failure must be available.
6. ***Authorization Services*** (Security): must support the implementation of access and authorization service to database administrator and users.
7. ***Support for Data Communication***: should provide the facility to integrate with data transfer software or data communication managers.
8. ***Integrity Services***: rules about data and the change that took place on the data, correctness and consistency of stored data, and quality of data based on business constraints.
9. Services to promote ***data independency*** between the data and the application
10. ***Utility services***: sets of utility service facilities like

- Importing data
- Statistical analysis support
- Index reorganization
- Garbage collection

DBMS and Components of DBMS Environment

A DBMS is software package used to design, manage, and maintain databases. It provides the following facilities:

Data Definition Language (DDL):

- Language used to define each data element required by the organization.
- Commands for setting up schema of database

Data Manipulation Language (DML):

- Language used by end-users and programmers to store, retrieve, and access the data.
- **Data Dictionary:** tool used to store and organize information about the data

The DBMS is software that helps to design, handle, and use data using the database approach. Taking a DBMS as a system, one can describe it with respect to it environment or other systems interacting with the DBMS. The DBMS environment has five components.

1. **Hardware**: Components that are comprised of personal computers, mainframe or any server computers, network infrastructure, etc.
2. **Software**: those components like the DBMS software, application programs, operating systems, network software, and other relevant software.
3. **Data**: This is the most important component to the user of the database.
   There are two types of data in a database approach that is *Operational* and *Metadata*. The structure of the data in the database is called the ***schema***, which is composed of the *Entities*, *Properties of entities*, and *relationship between entities*.
4. **Procedure**: this is the rules and regulations on *how to design and use* a database. It includes procedures like how to log on to the DBMS, how to use facilities, how to start and stop transaction, how to make backup, how to treat hardware and software failure, how to change the structure of the database.
5. **People**: people in the organization responsible to designing, implement, manage, administer and use of the database.

Database Development Life Cycle

As it is one component in most information system development tasks, there are several steps in designing a database system. Here more emphasis is given to the design phases of the system development life cycle. The major steps in database design are;

1. **Planning**: that is identifying information gap in an organization and propose a database solution to solve the problem.
2. **Analysis**: that concentrates more on fact finding about the problem or the opportunity. Feasibility analysis, requirement determination and structuring, and selection of best design method are also performed at this phase.
3. **Design**: in database designing more emphasis is given to this phase. The phase is further divided into three sub-phases.

a. **Conceptual Design**: concise description of the data, data type, relationship between data and constraints on the data.
   - There is no implementation or physical detail consideration.
   - Used to elicit and structure all information requirements

b. **Logical Design**: a higher-level conceptual abstraction with selected *specific data model* to implement the data structure.
   - It is particular DBMS **independent** and with no other physical considerations.

c. **Physical Design**: physical implementation of the upper level design of the database with respect to internal storage and file structure of the database for the selected DBMS.
   - To develop all technology and organizational specification.

4. **Implementation**: the testing and deployment of the designed database for use.
5. **Operation and Support**: administering and maintaining the operation of the database system and providing support to users.

Roles in Database Design and Use

As people are one of the components in DBMS environment, there are group of roles played by different stakeholders of the designing and operation of a database system.

1. Database Administrator (DBA)

- ✓ Responsible to oversee, control and manage the database resources (the database itself, the DBMS and other related software)
- ✓ Authorizing access to the database
- ✓ Coordinating and monitoring the use of the database
- ✓ Responsible for determining and acquiring hardware and software resources
- ✓ Accountable for problems like poor security, poor performance of the system

- ✓ Involves in all steps of database development

We can have further classifications of this role in big organizations having huge amount of data and user requirement.

1. **Data Administrator (DA)**: is responsible on management of data resources. Involves in database planning, development, maintenance of standards policies and procedures at the conceptual and logical design phases.

2. **DataBase Administrator (DBA)**: is more technically oriented role. Responsible for the physical realization of the database. Involves in physical design, implementation, security and integrity control of the database.

2. DataBase Designer (DBD)

- ✓ Identifies the data to be stored and choose the appropriate structures to represent and store the data.
- ✓ Should understand the user requirement and should choose how the user views the database.
- ✓ Involve on the design phase before the implementation of the database system.

We have two distinctions of database designers, one involving in the logical and conceptual design and another involving in physical design.

1. Logical and Conceptual DBD

- ✓ Identifies data (entity, attributes and relationship) relevant to the organization
- ✓ Identifies constraints on each data
- ✓ Understand data and business rules in the organization
- ✓ Sees the database independent of any data model at conceptual level and consider one specific data model at logical design phase.

2. Physical DBD

- ✓ Take logical design specification as input and decide how it should be physically realized.
- ✓ Map the logical data model on the specified DBMS with respect to tables and integrity constraints. (DBMS dependent designing)
- ✓ Select specific storage structure and access path to the database 🕘 Design security measures required on the database

3. Application Programmer and Systems Analyst

- ✓ System analyst determines the user requirement and how the user wants to view the database.
- ✓ The application programmer implements these specifications as programs; code, test, debug, document and maintain the application program.
- ✓ Determines the interface on how to retrieve, insert, update and delete data in the database.
- ✓ The application could use any high-level programming language according to the availability, the facility and the required service.

4. End Users

Workers, whose job requires accessing the database frequently for various purpose. There are different group of users in this category.

1. Naïve Users:

- ✓ Sizable proportion of users
- ✓ Unaware of the DBMS
- ✓ Only access the database based on their access level and demand
- ✓ Use standard and pre-specified types of queries.

2. Sophisticated Users

- ✓ Are users familiar with the structure of the Database and facilities of the DBMS.
- ✓ Have complex requirements
- ✓ Have higher level queries
- ✓ Are most of the time engineers, scientists, business analysts, etc

3. Casual Users

- ✓ Users who access the database occasionally.
- ✓ Need different information from the database each time.
- ✓ Use sophisticated database queries to satisfy their needs.
- ✓ Are most of the time middle to high level managers.

These users can be again classified as "Actors on the Scene" and "Workers Behind the Scene".

Actors On the Scene:

- ✓ Data Administrator
- ✓ Database Administrator
- ✓ Database Designer
- ✓ End Users

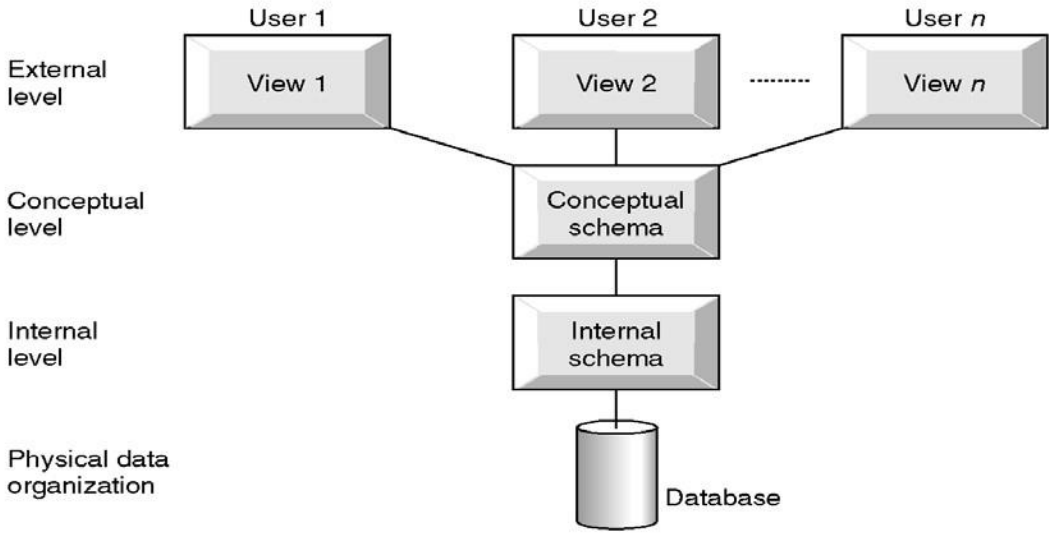
Workers Behind the Scene

- ✓ **DBMS** designers and implementers: who design and implement different DBMS software.
- ✓ **Tool Developers**: experts who develop software packages that facilitates database system designing and use. Prototype, simulation, code generator developers could be an example. Independent software vendors could also be categorized in this group.
- ✓ **Operators and Maintenance Personnel**: system administrators who are responsible for actually running and maintaining the hardware and software of the database system and the information technology facilities.

History of Database Systems

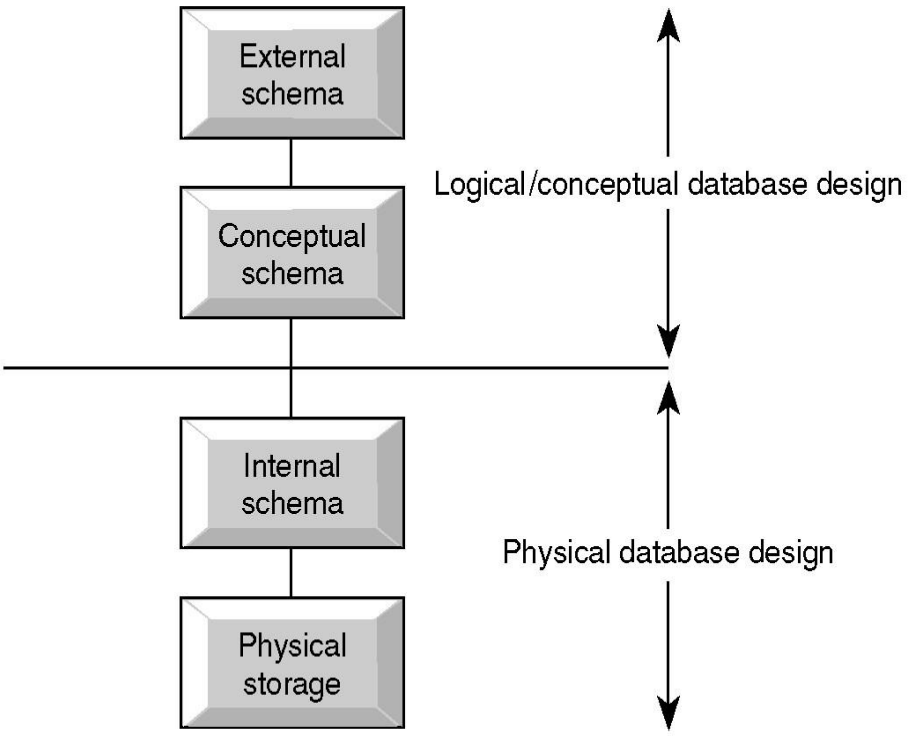
The purpose and origin of the Three-Level database architecture

- ✓ All users should be able to access same data
- ✓ A user's view is unaffected or immune to changes made in other views
- ✓ Users should not need to know physical database storage details
- ✓ DBA should be able to change database storage structures without affecting the users' views.
- ✓ Internal structure of database should be unaffected by changes to physical aspects of storage.
- ✓ DBA should be able to change conceptual structure of database without affecting all users.



ANSI-SPARC Three-level Architecture

ANSI-SPARC Architecture and Database Design Phases

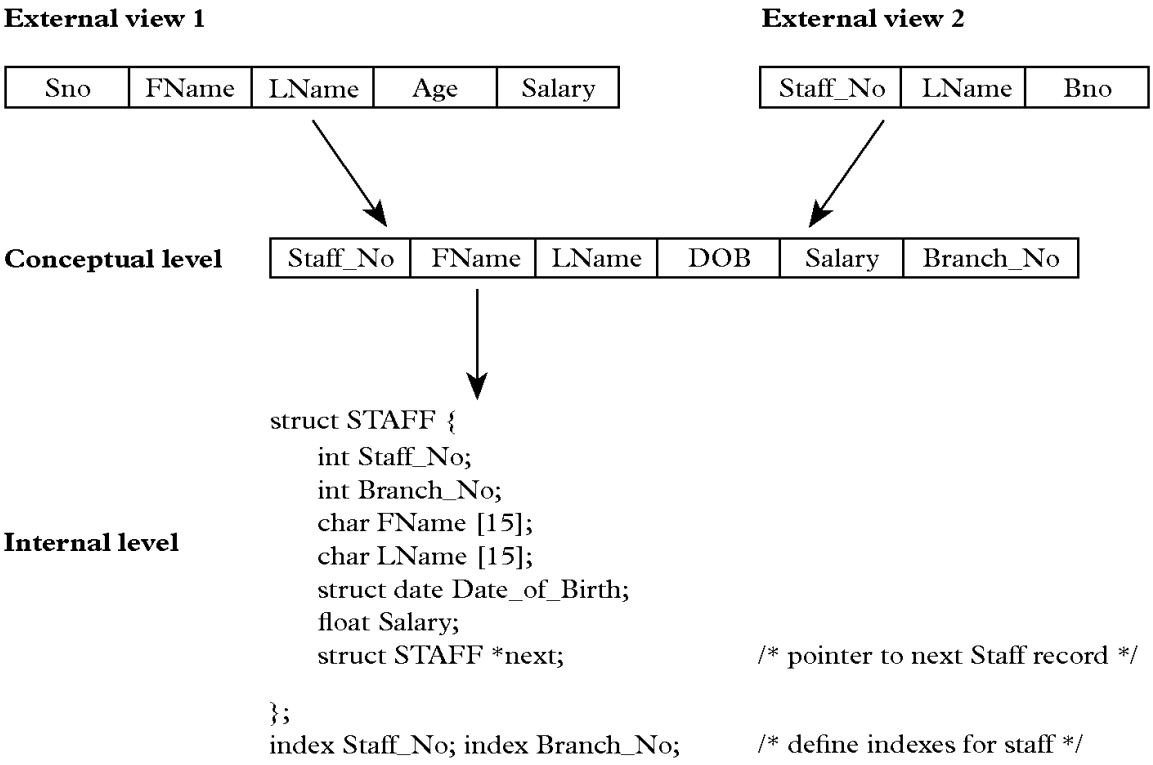


The contents of the external, conceptual and internal levels The purpose of the external/conceptual and the conceptual/internal mappings

External Level: Users' view of the database. Describes that part of database that is relevant to a particular user. Different users have their own customized view of the database independent of other users.

Conceptual Level: Community view of the database. Describes what data is stored in database and relationships among the data.

Internal Level: Physical representation of the database on the computer. Describes how the data is stored in the database.



Differences between Three Levels of ANSI-SPARC Architecture

Defines DBMS schemas at *three levels*:

Internal schema at the internal level to describe physical storage structures and access paths. Typically uses a *physical* data model.

Conceptual schema at the conceptual level to describe the structure and constraints for the *whole* database for a community of users. Uses a *conceptual* or an *implementation* data model.

External schemas at the external level to describe the various user views. Usually uses the same data model as the conceptual level.

The meaning of logical and physical data independence

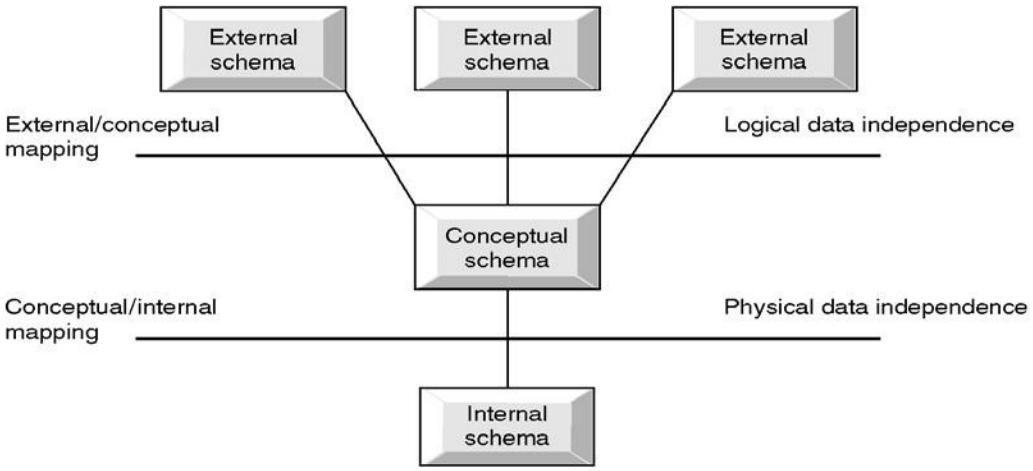
Data Independence

Logical Data Independence:

- ✓ Refers to immunity of external schemas to changes in conceptual schema.
- ✓ Conceptual schema changes e.g. addition/removal of entities should not require changes to external schema or rewrites of application programs.
- ✓ The capacity to change the conceptual schema without having to change the external schemas and their application programs.

Physical Data Independence

- ✓ The ability to modify the physical schema without changing the logical schema
- ✓ Applications depend on the logical schema
- ✓ In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.
  - ✓ The capacity to change the internal schema without having to change the conceptual schema
- ✓ Refers to immunity of conceptual schema to changes in the internal schema
- ✓ Internal schema changes e.g. using different file organizations, storage structures/devices should not require change to conceptual or external schemas.



Data Independence and the ANSI-SPARC Three-level Architecture

The distinction between a Data Definition Language (DDL) and a Data Manipulation Language (DML)

Database Languages

Data Definition Language (DDL)

- ✓ Allows DBA or user to describe and name entitles, attributes and relationships required for the application.
- ✓ Specification notation for defining the database schema

Data Manipulation Language (DML)

- ✓ Provides basic data manipulation operations on data held in the database.
- ✓ Language for accessing and manipulating the data organized by the appropriate data model
- ✓ DML also known as query language

Procedural DML: user specifies what data is required and how to get the data.

Non-Procedural DML: user specifies what data is required but not how it is to be retrieved

SQL is the most widely used non-procedural language query language

A Classification of data models

Data Model

A specific DBMS has its own specific Data Definition Language, but this type of language is too low level to describe the data requirements of an organization in a way that is readily understandable by a variety of users. We need a higher-level language. Such a higher-level is called data-model.

Data Model: a set of concepts to describe the *structure* of a database, and certain *constraints* that the database should obey.

A **data model** is a description of the way that data is stored in a database. Data model helps to understand the relationship between entities and to create the most effective structure to hold data.

Data Model is a collection of tools or concepts for describing

- ✓ Data
- ✓ Data relationships
- ✓ Data semantics
- ✓ Data constraints

The main ***purpose*** of Data Model is to represent the data in an understandable way.

Categories of data models include:

- ✓ Object-based
- ✓ Record-based
- ✓ Physical

Object-based Data Models

- ✓ Entity-Relationship
- ✓ Semantic
- ✓ Functional
- ✓ Object-Oriented

Record-based Data Models

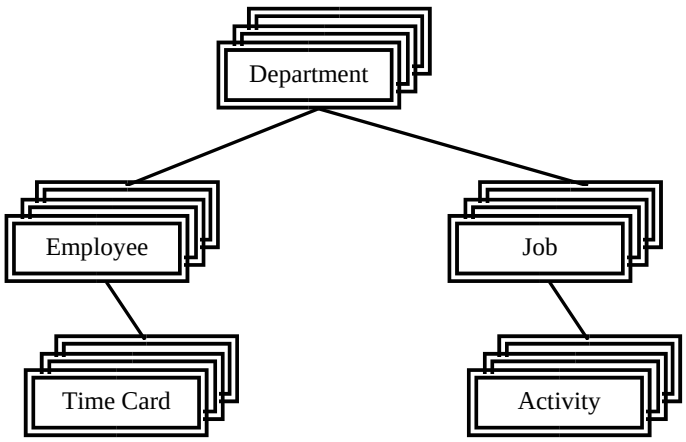
Consist of a number of fixed format records. Each record type defines a fixed number of fields, each field is typically of a fixed length.

We have three major types of data models

1. Hierarchical Model
2. Network Model
3. Relational Data Model

1. Hierarchical Model

- ✓ The simplest data model
- ✓ Record type is referred to as node or segment
- ✓ The top node is the root node
- ✓ Nodes are arranged in a hierarchical structure as sort of upside-down tree
- ✓ A parent node can have more than one child node
- ✓ A child node can only have one parent node
- ✓ The relationship between parent and child is one-to-many
- ✓ Relation is established by creating physical link between stored records (each is stored with a predefined access path to other records)
- ✓ To add new record type or relationship, the database must be redefined and then stored in a new form.



ADVANTAGES of Hierarchical Data Model:

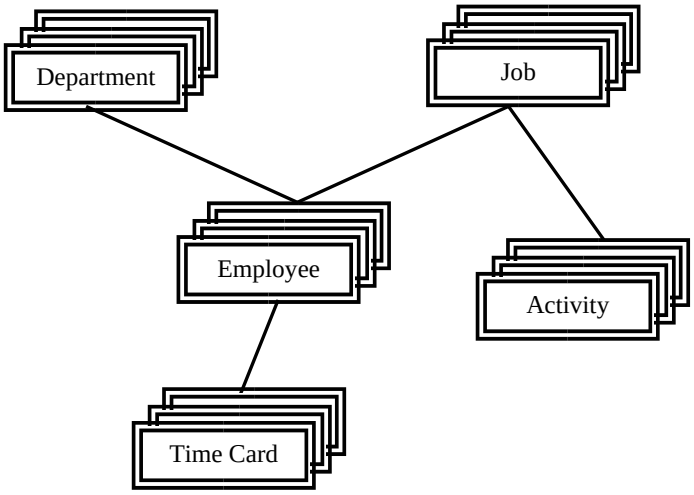
- ✓ Hierarchical Model is simple to construct and operate on
- ✓ Corresponds to a number of natural hierarchically organized domains - e.g., assemblies in manufacturing, personnel organization in companies
- ✓ Language is simple; uses constructs like GET, GET UNIQUE, GET NEXT, GET NEXT WITHIN PARENT etc.

DISADVANTAGES of Hierarchical Data Model:

- ✓ Navigational and procedural nature of processing
- ✓ Database is visualized as a linear arrangement of records
- ✓ Little scope for "query optimization"

2. Network Model

- ✓ Allows record types to have more that one parent unlike hierarchical model
- ✓ A network data models sees records as set members
- ✓ Each set has an owner and one or more members
- ✓ Allow no many to many relationship between entities
- ✓ Like hierarchical model network model is a collection of physically linked records.
- ✓ Allow member records to have more than one owner



ADVANTAGES of Network Data Model:

- ✓ Network Model is able to model complex relationships and represents semantics of add/delete on the relationships.
- ✓ Can handle most situations for modeling using record types and relationship types.
- ✓ Language is navigational; uses constructs like FIND, FIND member, FIND owner, FIND NEXT within set, GET etc. Programmers can do optimal navigation through the database.

DISADVANTAGES of Network Data Model:

- ✓ Navigational and procedural nature of processing
- ✓ Database contains a complex array of pointers that thread through a set of records.
- ✓ Little scope for automated "query optimization”

3. Relational Data Model

- ✓ Developed by **Dr. Edgar Frank Codd in 1970** (famous paper, 'A Relational Model for Large Shared Data Banks')
- ✓ Terminologies originates from the branch of mathematics called set theory and relation
- ✓ Can define more flexible and complex relationship
- ✓ Viewed as a collection of tables called "Relations" equivalent to collection of record types
- ✓ **Relation**: Two-dimensional table
- ✓ Stores information or data in the form of tables 🡺 **rows and columns**
- ✓ A **row** of the table is called tuple🡺 equivalent to **record**
- ✓ A **column** of a table is called attribute🡺 equivalent to **fields**
- ✓ Data value is the value of the Attribute
- ✓ Records are related by the data stored jointly in the fields of records in two tables or files. The related tables contain information that creates the relation
- ✓ The tables seem to be independent but are related somehow.
- ✓ No physical consideration of the storage is required by the user
- ✓ Many tables are merged together to come up with a new virtual view of the relationship

Alternative terminologies		
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

- ✓ The rows represent records (collections of information about separate items)
- ✓ The columns represent fields (particular attributes of a record)
- ✓ Conducts searches by using data in specified columns of one table to find additional data in another table
- ✓ In conducting searches, a relational database matches information from a field in one table with information in a corresponding field of another table to produce a third table that combines requested data from both tables

Relational Data Model

Properties of Relational Databases

- ✓ Each row of a table is uniquely identified by a **PRIMARY KEY** composed of one or more columns
- ✓ Each tuple in a relation must be unique
- ✓ Group of columns, that uniquely identifies a row in a table is called a **CANDIDATE KEY**
- ✓ **ENTITY INTEGRITY RULE** of the model states that no component of the primary key may contain a **NULL** value.
- ✓ A column or combination of columns that matches the primary key of another table is called a **FOREIGN KEY**. Used to cross-reference tables.
- ✓ The **REFERENTIAL INTEGRITY RULE** of the model states that, for every foreign key value in a table there must be a corresponding primary key value in another table in the database or it should be **NULL**.
- ✓ All tables are **LOGICAL ENTITIES**
- ✓ A table is either a BASE TABLES (Named Relations) or VIEWS (Unnamed Relations)
- ✓ Only Base Tables are physically stores
- ✓ VIEWS are derived from BASE TABLES with SQL instructions like:
    - [SELECT .. FROM .. WHERE .. ORDER BY]
- ✓ Is the collection of tables
    - Each entity in one table
    - Attributes are fields (columns) in table
- ✓ Order of rows and columns is immaterial
- ✓ Entries with repeating groups are said to be un-normalized
- ✓ Entries are single-valued
- ✓ Each column (field or attribute) has a distinct name

All values in a column represent the same attribute and have the same data format

Building Blocks of the Relational Data Model

The building blocks of the relational data model are:

- **Entities**: real world physical or logical object
- **Attributes**: properties used to describe each Entity or real-world object.
- **Relationship**: the association between Entities
- **Constraints**: rules that should be obeyed while manipulating the data.

1. The **ENTITIES** (persons, places, things etc.) which the organization has to deal with. Relations can also describe relationships

   The name given to an entity should always be a singular noun descriptive of each item to be stored in it. E.g.: student NOT students.

   Every relation has a schema, which describes the columns, or fields
   The relation itself corresponds to our familiar notion of a table: A relation is a collection of ***tuples***, each of which contains values for a fixed number of ***attributes***

2. The **ATTRIBUTES** - the items of information which characterize and describe these entities.

   Attributes are pieces of information ABOUT entities. The analysis must of course identify those which are actually relevant to the proposed application. Attributes will give rise to recorded items of data in the database

At this level we need to know such things as:

- ✓ **Attribute name** (be explanatory words or phrases)
- ✓ **The domain** from which attribute values are taken (A DOMAIN is a set of values from which attribute values may be taken.) Each attribute has values taken from a *domain*. For example, the domain of Name is string and that for salary is real
- ✓ Whether the attribute is part of the **entity identifier** (attributes which just describe an entity and those which help to identify it uniquely)
- ✓ Whether it is **permanent or time-varying** (which attributes may change their values over time)
- ✓ Whether it is **required or optional** for the entity (whose values will sometimes be unknown or irrelevant)

Types of Attributes

(1) Simple (atomic) Vs Composite attributes

- **Simple**: contains a single value (not divided into sub parts) E.g. Age, gender
- **Composite**: Divided into sub parts (composed of other attributes)
  E.g., Name, address

(2) Single-valued Vs multi-valued attributes

- **Single-valued:** have only single value (the value may change but has only one value at one time)
  E.g., Name, Sex, Id. No. color_of_eyes
- **Multi-Valued:** have more than one value **E.g.,** Address, dependent-name, Person may have several college degrees

(3) Stored vs. Derived Attribute

- **Stored**: not possible to derive or compute
  E.g., Name, Address
- **Derived**: The value may be derived (computed) from the values of other attributes.
  E.g. Age (current year – year of birth)
  Length of employment (current date- start date)
  Profit (earning-cost)
  G.P.A (grade point/credit hours)

(4) Null Values

- NULL applies to attributes which are not applicable or which do not have values.
- You may enter the value NA (meaning not applicable)
- Value of a key attribute cannot be null.

Default value - assumed value if no explicit value

3. The **RELATIONSHIPS** between entities which exist and must be taken into account when processing information.

- One external event or process may affect several related entities.
- Related entities require setting of LINKS from one part of the database to another.
- A relationship should be named by a word or phrase which explains its function
- Role names are different from the names of entities forming the relationship: one entity may take on many roles, the same role may be played by different entities
- An important point about a relationship is how many entities participate in it. The number of entities participating in a relationship is called the **DEGREE** of the relationship.

    - **UNARY/RECURSIVE RELATIONSHIP:** *Single entity*
    - **BINARY RELATIONSHIPS:** *Two entities associated*
    - **TERNARY RELATIONSHIP:** *Three entities associated*
    - **N-NARY RELATIONSHIP:** *arbitrary number of entities sets*

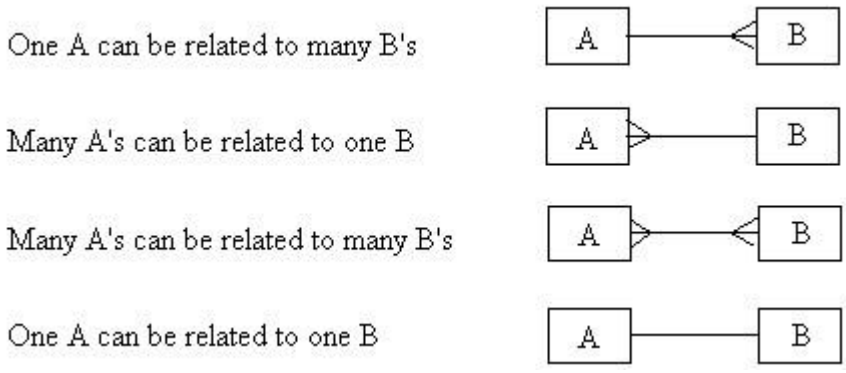
• Another important point about relationship is the range of instances that can be associated with a single instance from one entity in a single relationship. The number of instances participating or associated with a single instance from another entity in a relationship is called the **CARDINALITY** of the relationship.

ONE-TO-ONE, e.g., person - Passport,

ONE-TO-MANY, e.g., hospital - patient,

MANY-TO-ONE, e.g. Employee – Department

MANY-TO-MANY, e.g., Author - Book.



4. **Relational Constraints/Integrity Rules**

• Relational Integrity

- **Domain Integrity:** No value of the attribute should be beyond the allowable limits
- **Entity Integrity:** In a base relation, no attribute of a primary key can be null
- **Referential Integrity**: If a foreign key exists in a relation, either the foreign key value must match a candidate key in its home relation or the foreign key value must be null foreign key to primary key match-ups
- **Enterprise Integrity:** Additional rules specified by the users or database administrators of a database are incorporated

• Key constraints

If tuples are need to be unique in the database, and then we need to make each tuple distinct. To do this we need to have relational keys that uniquely identify each relation.

Super Key: an attribute or set of attributes that uniquely identifies a tuple within a relation.

Candidate Key: a super key such that no proper subset of that collection is a Super Key within the relation. A candidate key has two properties:

1. **Uniqueness**
2. **Irreducibility**

If a candidate key consists of more than one attribute it is called composite key.

Primary Key: the candidate key that is selected to identify tuples uniquely within the relation.

The entire set of attributes in a relation can be considered as a primary case in a worst case.

Foreign Key: an attribute, or set of attributes, within one relation that matches the candidate key of some relation.

A foreign key is a link between different relations to create the view or the unnamed relation

• Relational languages and views

The languages in relational database management systems are the DDL and the DML that are used to define or create the database and perform manipulation on the database.

We have the two kinds of relation in relational database. The difference is on how the relation is created, used and updated:

1. **Base Relation**
   A ***Named Relation*** corresponding to an entity in the conceptual schema, whose tuples are physically stored in the database.
2. **View**
   Is the dynamic result of one or more relational operations operating on the base relations to produce another virtual relation. So a view **virtually derived relation** that does not necessarily exist in the database but can be produced upon request by a particular user at the time of request.

Purpose of a view

- ✓ Hides unnecessary information from users
- ✓ Provide powerful flexibility and security 🕘 Provide customized view of the database for users 🕘 A view of one base relation can be updated.
- ✓ Update on views derived from various relations is not allowed
- ✓ Update on view with aggregation and summary is not allowed.

Schemas and Instances and Database State

- ✓ Relational databases are developed based on the relational data model

Schemas

- ✓ Schema describes how data is to be structured, defined at set-up time, rarely changes (also called "metadata")
- ✓ **Database Schema** (**intension**): specifies name of relation, plus name and type of each column.
  - refer to a description of database (or intention)
  - specified during database design
  - should not be changed unless during maintenance

- ✓ **Schema Diagrams**
  - convention to display some aspect of a schema visually
- ✓ **Schema Construct**
  - refers to each object in the schema (e.g. STUDENT)

    E.g.: STUNEDT (FName,LName,Id,Year,Dept,Sex)

Three-Schema Architecture

Internal schema (or internal level)

- ✓ Internal schema describes the physical storage, structure of the database (data storage, access paths)

Conceptual schema (or conceptual level)

- ✓ describes the structure of the entire database
- ✓ hides the details of physical storage structures
- ✓ concentrates on the describing
  - entities, data types, relationships, operations, and constraints
- ✓ high-level data models or an implementation data model may be used here

External schema (or view-level)

- ✓ includes a number of external schema or user view
- ✓ each view describes subset of database needed by a particular user
  - High Level data model or an implementation data model can be used here

Instances

- ✓ **Database state (snapshot or extension)**: is the collection of data in the database at a particular point of time (snap-shot).
  - Refers to the actual data in the database at a specific time
  - State of database is changed any time we add or delete a record
  - ***Valid state***: the state that satisfies the structure and constraints specified in the schema and is enforced by DBMS
- ✓ Instance is actual data of database at some point in time, changes rapidly
- ✓ To define a new database, we specify its database schema to the DBMS (database is empty)
- ✓ database is initialized when we first load it with data

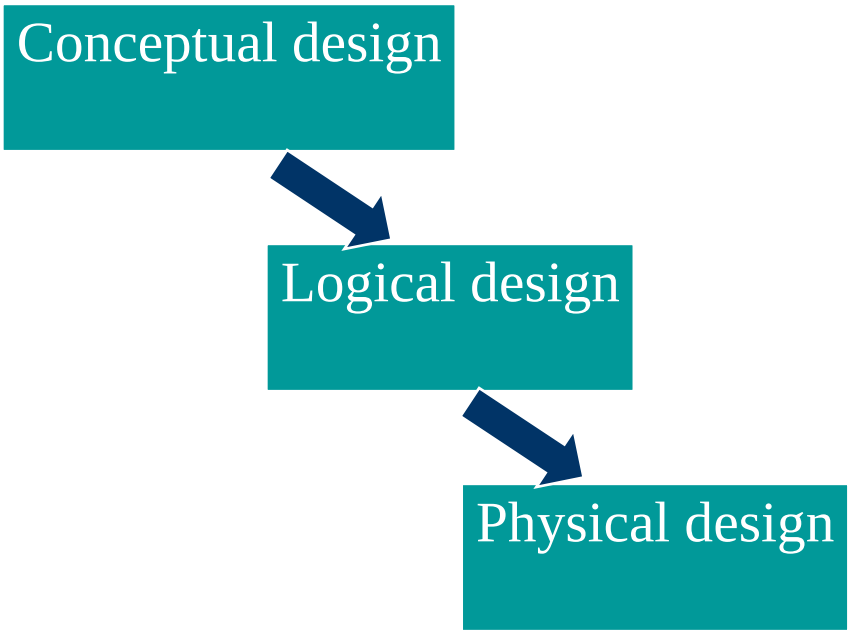
Database Design

Database design consists of several tasks:

- *Requirements Analysis,*
- *Conceptual Design, and Schema Refinement,*
- *Logical Design,*
- *Physical Design* and
- *Tuning*

- In general, one has to go back and forth between these tasks to refine a database design, and decisions in one task can influence the choices in another task.
- In developing a good design, one should ask:
- What are the important queries and updates? What attributes/relations are involved?

The Three levels of Database Design



Conceptual Database Design

- Conceptual design is the process of constructing a model of the information used in an enterprise, ***independent of any physical considerations.***
  - It is the source of information for the logical design phase.
  - Community User's view
- After the completion of Conceptual Design one has to go for refinement of the schema, which is verification of Entities, Attributes, and Relationships

Logical Database Design

- Logical design is the process of constructing a model of the information used in an enterprise based on a specific data model (e.g. relational, hierarchical or network or object), but independent of a particular DBMS and other physical considerations.
  - Normalization process
    - Discover new entities
    - Revise attributes

Physical Database Design

- Physical design is the process of producing a description of the implementation of the database on secondary storage. -- defines specific storage or access methods used by database
  - Describes the storage structures and access methods used to achieve efficient access to the data.
  - Tailored to a specific DBMS system -- Characteristics are function of DBMS and operating systems
  - Includes estimate of storage space

Conceptual Database Design

- Conceptual design revolves around discovering and analyzing organizational and user data requirements
- The important activities are to identify
  - Entities
  - Attributes
  - Relationships
  - Constraints
- And based on these components develop the ER model using
  - ER diagrams

The Entity Relationship (E-R) Model

- Entity-Relationship modeling is used to represent conceptual view of the database
- The main components of ER Modeling are:

Entities

- Corresponds to entire table, not row
- Represented by Rectangle

Attributes

- Represents the property used to describe an entity or a relationship
- Represented by Oval

Relationships

- Represents the association that exist between entities
- Represented by Diamond

Constraints

- Represent the constraint in the data

Before working on the conceptual design of the database, one has to know and answer the following basic questions.

- What are the ***entities*** and ***relationships*** in the enterprise?
- What information about these entities and relationships should we store in the database?
- What are the ***integrity constraints*** that hold? Constraints on each data with respect to update, retrieval and store.
- Represent this information pictorially in ***ER diagrams***, then map ER diagram into a relational schema.

Developing an E-R Diagram

- Designing conceptual model for the database is not a one linear process but an iterative activity where the design is refined again and again.
- To identify the entities, attributes, relationships, and constraints on the data, there are different set of methods used during the analysis phase.
- These include information gathered by…
  - Interviewing end users individually and in a group
  - Questionnaire survey
  - Direct observation

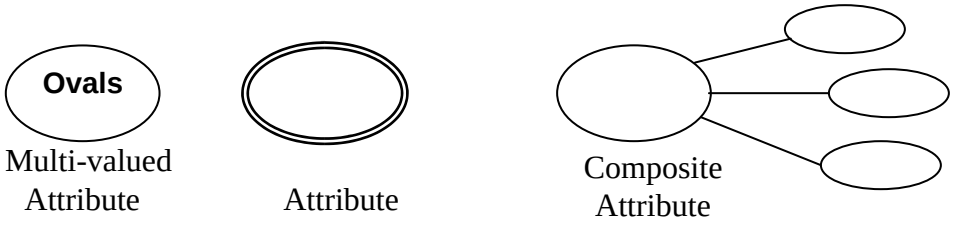
- Examining different documents

- The basic E-R model is graphically depicted and presented for review.
- The process is repeated until the end users and designers agree that the ER diagram is a fair representation of the organization's activities and functions.
- Checking for Redundant Relationships in the ER Diagram. Relationships between entities indicate access from one entity to another - it is therefore possible to access one entity occurrence from another entity occurrence even if there are other entities and relationships that separate them - this is often referred to as *Navigation*' of the ER diagram
- The last phase in ER modeling is validating an ER Model against requirement of the user.

Graphical Representations in ER Diagramming

- Entity is represented by a **RECTANGLE** containing the name of the entity.

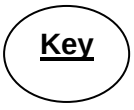
- Connected entities are called relationship participants
- Attributes are represented by **OVALS** and are



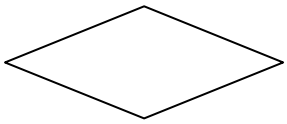
- A derived attribute is indicated by a **DOTTED LINE**.

(……..)

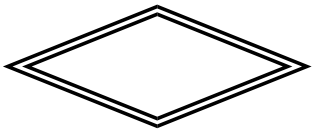
- **PRIMARY KEYS** are underlined.



- Relationships are represented by **DIAMOND** shaped symbols



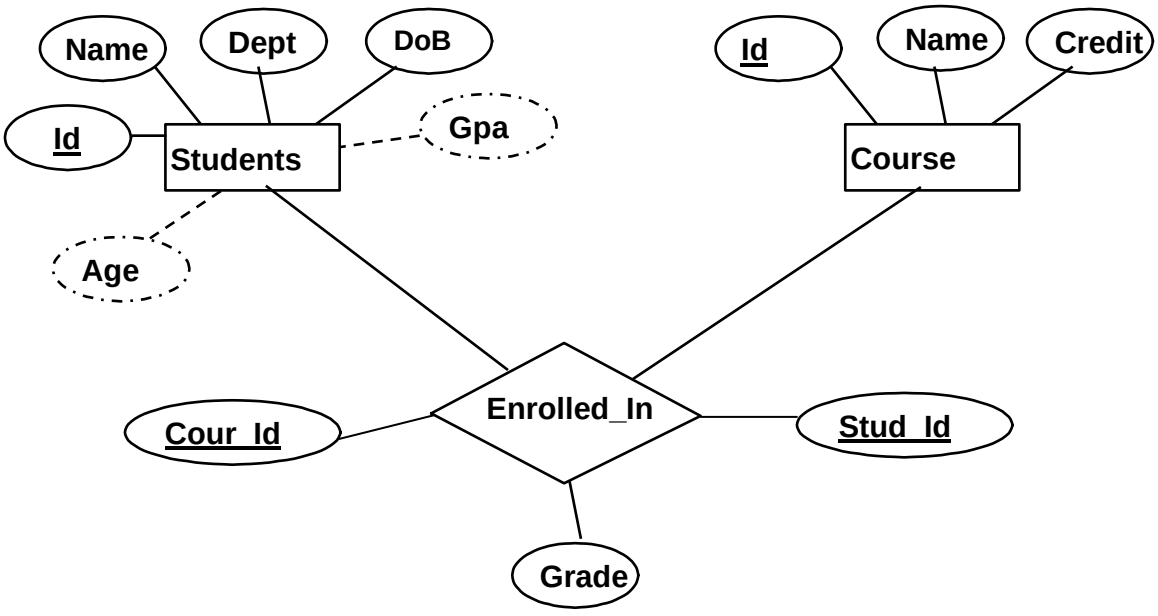
Relationship



Weak Relationship

Example 1

- **Build an ER Diagram for the following information:**
  - **Students**
    - **Have an Id, Name, Dept, Age, Gpa**
  - **Courses**
    - **Have an Id, Name, Credit Hours**
  - **Students enroll in courses and receive a grade**



Entity versus Attributes

- Consider designing a database of employees for an organization:
- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
    - If we have several addresses per employee, *address* must be an entity (*attributes cannot be set-valued/multi valued*)
- If the structure (city, Woreda, Kebele, etc) is important, e.g. want to retrieve employees in a given city, address must be modeled as an entity (*attribute values are atomic*)
- Cardinality on Relationship expresses the number of entity occurrences/tuples associated with one occurrence/tuple of related entity.
- Existence Dependency: the dependence of an entity on the existence of one or more entities.
- Weak entity : an entity that can not exist without the entity with which it has a relationship – it is indicated by a **double rectangle**
- Participating entity in a relationship is either optional or mandatory.

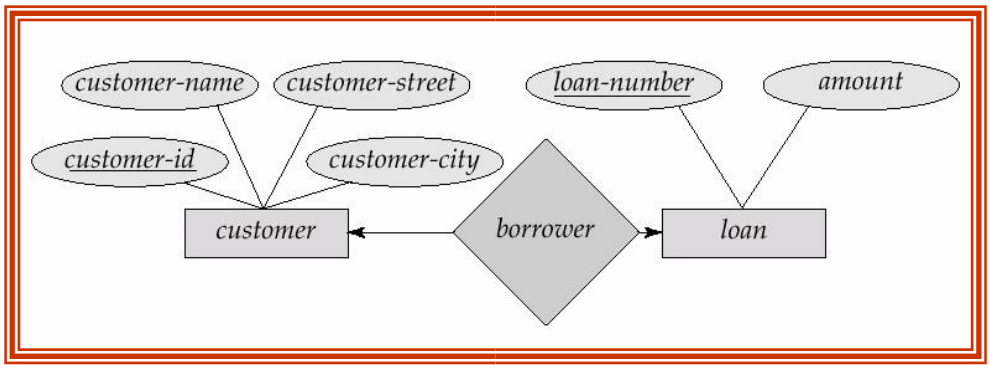
Structural Constraints on Relationship

1. Constraints on Relationship / Multiplicity/ Cardinality Constraints

- Multiplicity constraint is the number of or range of possible occurrence of an entity type/relation that may relate to a single occurrence/tuple of an entity type/relation through a particular relationship.
- Mostly used to insure appropriate enterprise constraints.

One-to-one relationship:

- A customer is associated with at most one loan via the relationship *borrower*
- A loan is associated with at most one customer via *borrower*

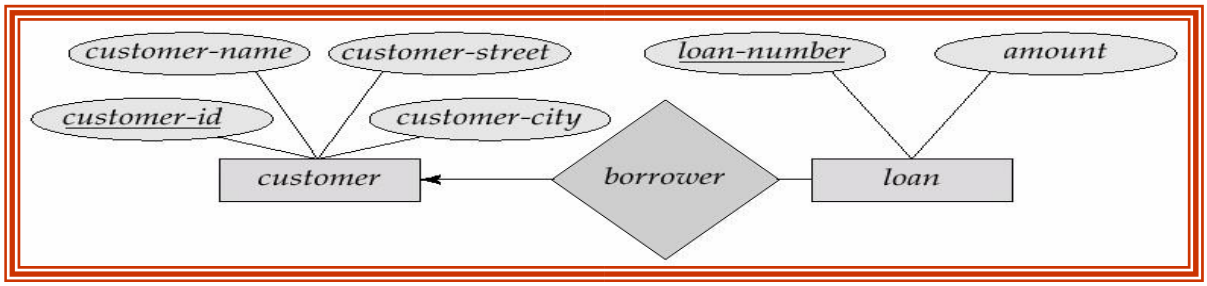


E.g.: Relationship ***Manages*** between ***STAFF*** and ***BRANCH The multiplicity of the relationship is:***

- *One branch can only have one manager*
- *One employee could manage either one or no branches*

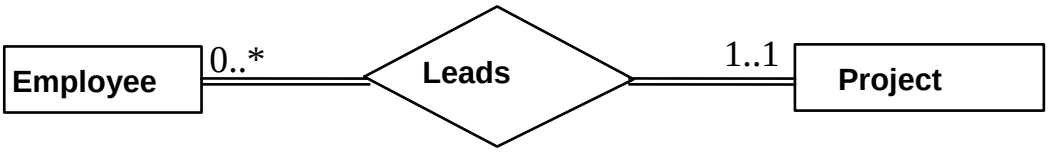
One-To-Many Relationships

- In the one-to-many relationship a loan is associated with at most one customer via *borrower*, a customer is associated with several (including 0) loans via *borrower*



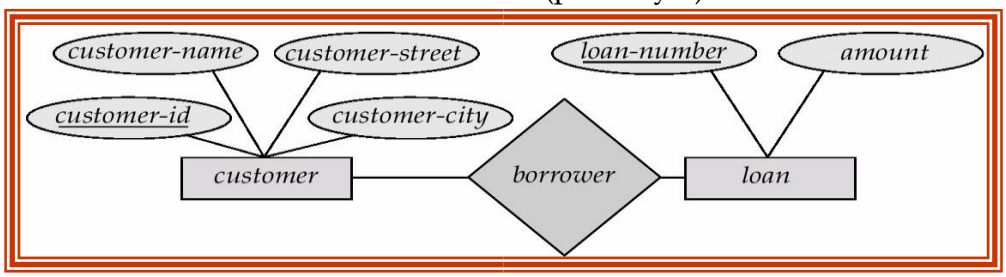
E.g.: Relationship ***Leads*** between ***STAFF*** and ***PROJECT***
The multiplicity of the relationship

- *One staff may Lead one or more project(s)*
- *One project is Lead by one staff*



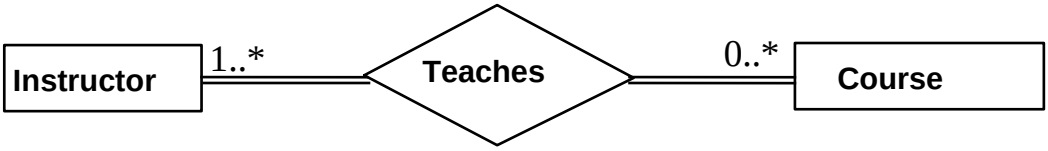
Many-To-Many Relationship

- A customer is associated with several (possibly 0) loans via borrower
- A loan is associated with several (possibly 0) customers via borrower



E.g.: Relationship ***Teaches*** between ***INSTRUCTOR*** and ***COURSE***
The multiplicity of the relationship

- *One Instructor Teaches one or more Course(s)*
- *One Course Thought by Zero or more Instructor(s)*



Participation of an Entity Set in a Relationship Set

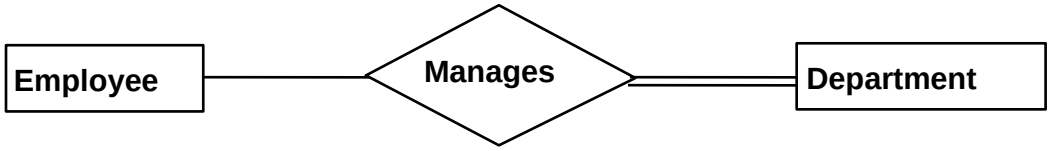
- **<u>Total participation</u>** (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set. The entity with total participation will be connected with the relationship using a double line.

  E.g. 1: Participation of EMPLOYEE in "belongs to" relationship with DEPARTMENT is total since every employee should belong to a department.

E.g. 2: Participation of EMPLOYEE in "manages" relationship with DEPARTMENT, DEPARTMENT will have total participation but not EMPLOYEE

- **<u>Partial participation</u>**: some entities may not participate in any relationship in the relationship set

  E.g. 1: Participation of EMPLOYEE in "manages" relationship with DEPARTMENT, EMPLOYEE will have partial participation since not all employees are managers.



Problem in ER Modeling

The Entity-Relationship Model is a conceptual data model that views the real world as consisting of entities and relationships. The model visually represents these concepts by the Entity-Relationship diagram. The basic constructs of the ER model are entities, relationships, and attributes. Entities are concepts, real or abstract, about which information is collected. Relationships are associations between the entities. Attributes are properties which describe the entities.

While designing the ER model one could face a problem on the design which is called a connection traps. **Connection traps** are problems arising from misinterpreting certain relationships

There are two types of connection traps;

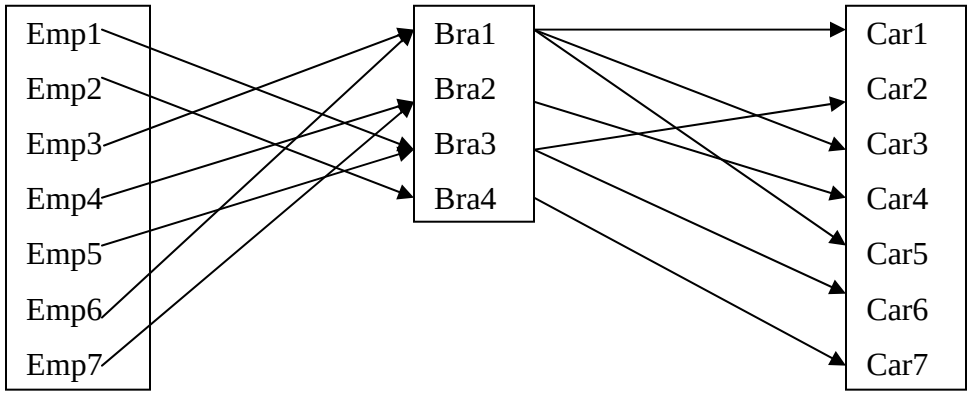
1. **Fan trap:**

   Occurs where a model represents a relationship between entity types, but the pathway between certain entity occurrences is ambiguous.

   May exist where two or more one-to-many (1:M) relationships fan out from an entity. The problem could be avoided by restructuring the model so that there would be no 1:M relationships fanning out from a single entity and all the semantics of the relationship is preserved.

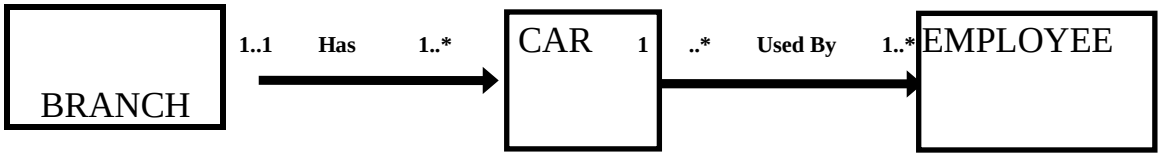
Example:

Semantics description of the problem;

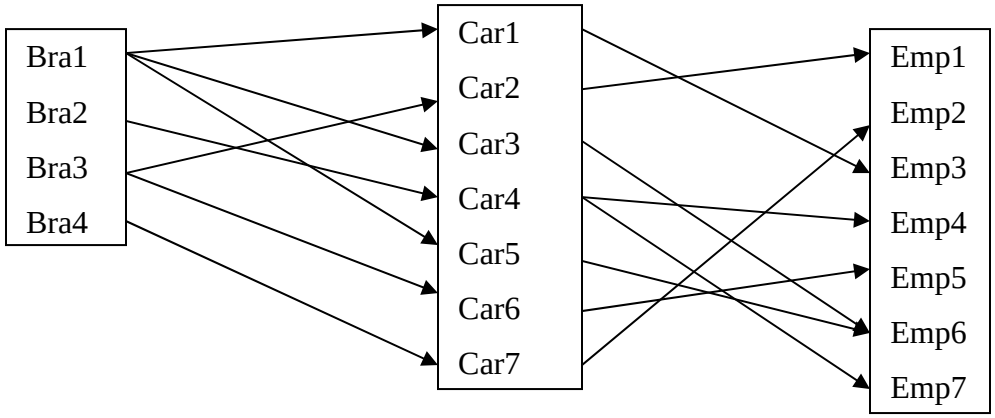


Problem: Which car (Car1 or Car3 or Car5) is used by Employee 6 Emp6 working in Branch 1 (Bra1)? Thus, from this ER Model one cannot tell which car is used by which staff since a branch can have more than one car and also a branch is populated by more than one employee. Thus, we need to restructure the model to avoid the connection trap.

To avoid the Fan Trap problem, we can go for restructuring of the E-R Model. This will result in the following E-R Model.



Semantics description of the problem;



2. Chasm Trap:

Occurs where a model suggests the existence of a relationship between entity types, but the path way does not exist between certain entity occurrences.

May exist when there are one or more relationships with a minimum multiplicity on cardinality of zero forming part of the pathway between related entities.

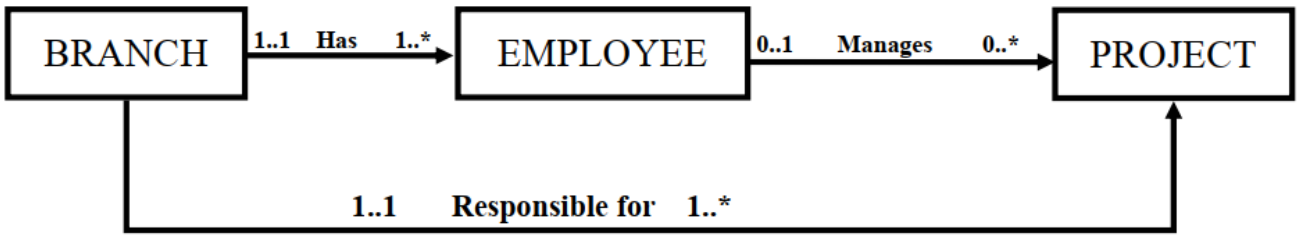
Example:

If we have a set of projects that are not active currently then we cannot assign a project manager for these projects. So, there are project with no project manager making the participation to have a minimum value of zero.

Problem:

How can we identify which BRANCH is responsible for which PROJECT? We know that whether the PROJECT is active or not there is a responsible BRANCH. But which branch is a question to be answered, and since we have a minimum participation of zero between employee and PROJECT, we can't identify the BRANCH responsible for each PROJECT.

The solution for this Chasm Trap problem is to add another relationship between the extreme entities (BRANCH and PROJECT)



Enhanced E-R (EER) Models

- Object-oriented extensions to E-R model
- EER is important when we have a relationship between two entities and the participation is partial between entity occurrences. In such cases EER is used to reduce the complexity in participation and relationship complexity.
- ER diagrams consider entity types to be primitive objects
- EER diagrams allow refinements within the structures of entity types

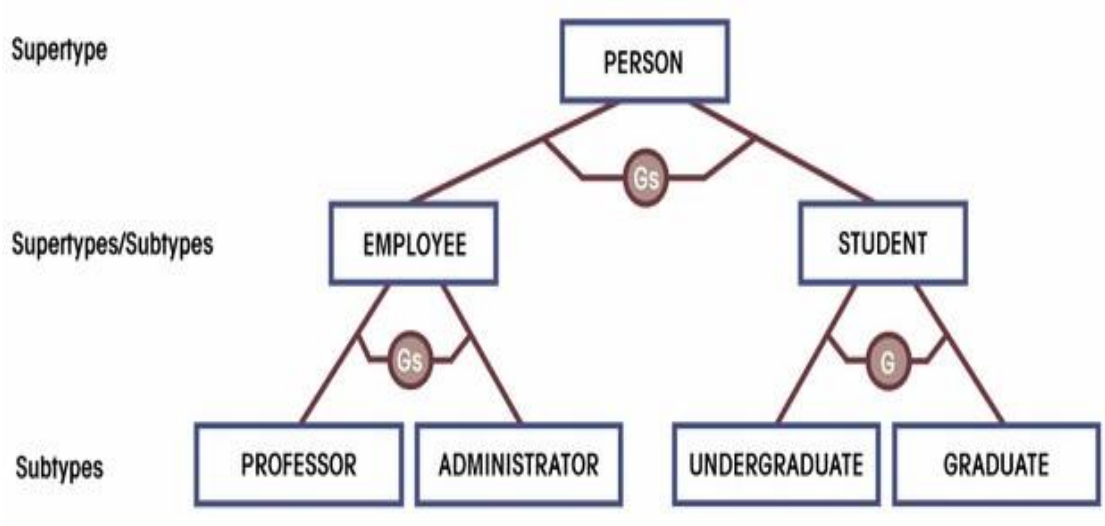
- EER Concepts
    - Generalization
    - Specialization
    - Sub classes
    - Super classes
    - Attribute Inheritance
        - Constraints on specialization and generalization

Generalization

- Generalization occurs when two or more entities represent categories of the same real-world object.
- Generalization is the process of defining a more general entity type from a set of more specialized entity types.
- A generalization hierarchy is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher-level entity type.
- Is considered as bottom-up definition of entities.
- Generalization hierarchy depicts relationship between higher level superclass and lower-level subclass.

Generalization hierarchies can be nested. That is, a subtype of one hierarchy can be a supertype of another. The level of nesting is limited only by the constraint of simplicity.

Example: Account is a generalized form for Saving and Current Accounts



Specialization

- Is the result of subset of a higher-level entity set to form a lower-level entity set.
- The specialized entities will have additional set of attributes (distinguishing characteristics) that distinguish them from the generalized entity.
- Is considered as Top-Down definition of entities.
- Specialization process is the inverse of the Generalization process. Identify the distinguishing features of some entity occurrences, and specialize them into different subclasses.
- Reasons for Specialization
    - Attributes only partially applying to super classes
    - Relationship types only partially applicable to the superclass
- In many cases, an entity type has numerous sub-groupings of its entities that are meaningful and need to be represented explicitly. This need requires the representation of each subgroup in the ER model. The generalized entity is a superclass and the set of specialized entities will be subclasses for that specific Superclass.
- **Example**: Saving Accounts and Current Accounts are Specialized entities for the generalized entity Accounts. Manager, Sales, Secretary: are specialized employees.

Subclass/Subtype

- ✓ An entity type whose tuples have attributes that distinguish its members from tuples of the generalized or Superclass entities.
- ✓ When one generalized Superclass has various subgroups with distinguishing features and these subgroups are represented by specialized form, the groups are called subclasses.
- ✓ Subclasses can be either mutually exclusive (disjoint) or overlapping (inclusive).
- ✓ A single subclass may inherit attributes from two distinct superclasses.
- ✓ A mutually exclusive category/subclass is when an entity instance can be in only one of the subclasses.
    - E.g.: An EMPLOYEE can either be SALARIED or PART-TIMER but not both.
- ✓ An overlapping category/subclass is when an entity instance may be in two or more subclasses.
    - E.g.: A PERSON who works for a university can be both EMPLOYEE and a STUDENT at the same time.

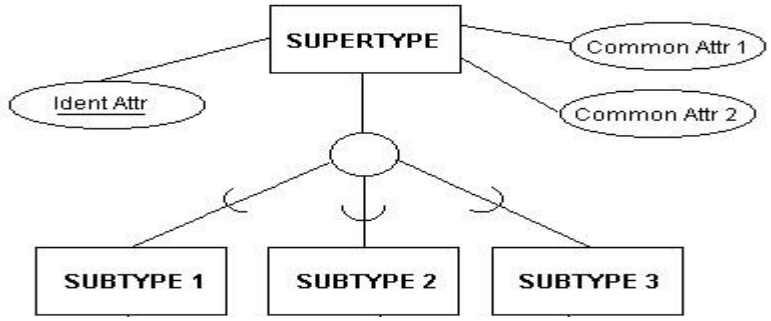
Superclass /Supertype

- ✓ An entity type whose tuples share common attributes. Attributes that are shared by all entity occurrences (including the identifier) are associated with the supertype.
    - ✓ Is the generalized entity

Relationship Between Superclass and Subclass

- ✓ The relationship between a superclass and any of its subclasses is called a superclass/subclass or class/subclass relationship
- ✓ An instance can not only be a member of a subclass. i.e. Every instance of a subclass is also an instance in the Superclass.
- ✓ A member of a subclass is represented as a distinct database object, a distinct record that is related via the key attribute to its super-class entity.
- ✓ An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass.
- ✓ An entity occurrence of a sub class not necessarily should belong to any of the subclasses unless there is full participation in the specialization.
- ✓ A member of a subclass is represented as a distinct database object, a distinct record that is related via the key attribute to its super-class entity.

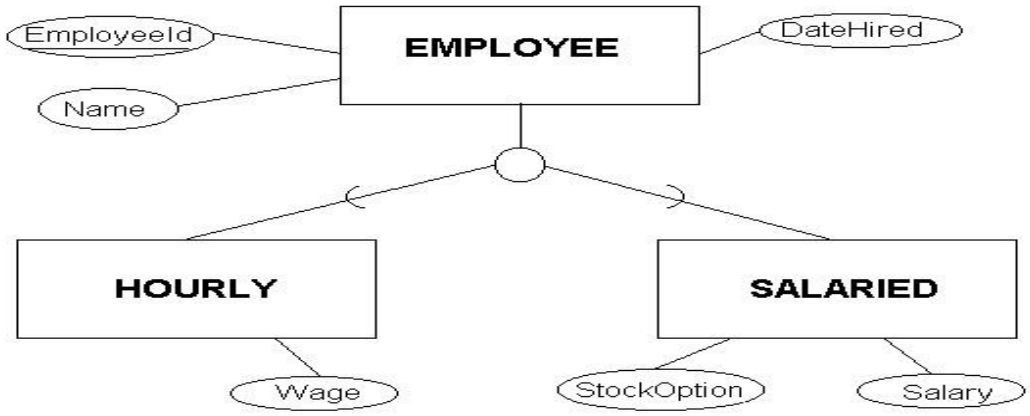
- ✓ The relationship between a subclass and a Superclass is an "IS A" or "IS PART OF" type.
  - ***Subclass IS PART OF Superclass***
  - ***Manager IS AN Employee***
- ✓ All subclasses or specialized entity sets should be connected with the superclass using a line to a circle where there is a subset symbol indicating the direction of subclass/superclass relationship.



- ✓ We can also have subclasses of a subclass forming a hierarchy of specialization.
- ✓ Superclass attributes are shared by all subclasses f that superclass
- ✓ Subclass attributes are unique for the subclass.

Attribute Inheritance

- ✓ An entity that is a member of a subclass inherits all the attributes of the entity as a member of the superclass.
- ✓ The entity also inherits all the relationships in which the superclass participates.
- ✓ An entity may have more than one subclass categories.
- ✓ All entities/subclasses of a generalized entity or superclass share a common unique identifier attribute (primary key). i.e. The primary key of the superclass and subclasses are always identical.



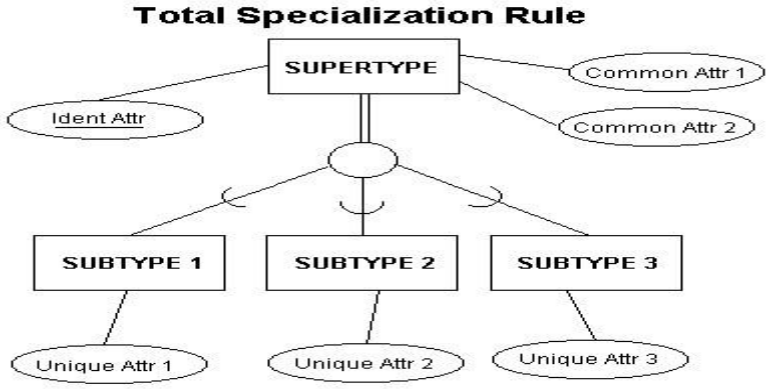
- ✓ Consider the **EMPLOYEE** supertype entity shown above. This entity can have several different subtype entities (for example: **HOURLY** and **SALARIED**), each with distinct properties not shared by other subtypes. But whether the employee is **HOURLY** or **SALARIED**, same attributes (EmployeeId, Name, and DateHired) are shared.
- ✓ The Supertype **EMPLOYEE** stores all properties that subclasses have in common. And **HOURLY** employees have the unique attribute Wage (hourly wage rate), while **SALARIED** employees have two unique attributes, StockOption and Salary.

Constraints on specialization and generalization

④ Completeness Constraint.

- ✓ The **Completeness Constraint** addresses the issue of whether or not an occurrence of a Superclass must also have a corresponding Subclass occurrence.
- ✓ The completeness constraint requires that all instances of the subtype be represented in the supertype.
- ✓ The **Total Specialization Rule** specifies that an entity occurrence should at least be a member of one of the subclasses. Total Participation of superclass instances on subclasses is diagrammed with a **double line** from the Supertype to the circle as shown below.

E.g.: If we have EXTENTION and REGULAR as subclasses of a superclass STUDENT, then it is mandatory that each student to be either EXTENTION or REGULAR student. Thus the participation of instances of STUDENT in EXTENTION and REGULAR subclasses will be total.



- The **Partial Specialization Rule** specifies that it is not necessary for all entity occurrences in the superclass to be a member of one of the subclasses. Here we have an optional participation on the specialization. Partial Participation of superclass instances on subclasses is diagrammed with a **single line** from the Supertype to the circle.

  E.g.: If we have MANAGER and SECRETARY as subclasses of a superclass EMPLOYEE, then it is not the case that all employees are either manager or secretary. Thus the participation of instances of employee in MANAGER and SECRETARY subclasses will be partial.

Disjointness Constraints.

- Specifies the rule whether one entity occurrence can be a member of more than one subclasses. i.e. it is a type of business rule that deals with the situation where an entity occurrence of a Superclass may also have more than one Subclass occurrence.
- The **Disjoint Rule** restricts one entity occurrence of a superclass to be a member of only one of the subclasses. Example: a EMPLOYEE can either be SALARIED or PART-TIMER, but not the both at the same time.
- The **Overlap Rule** allows one entity occurrence to be a member f more than one subclass. Example: EMPLOYEE working at the university can be both a STUDENT and an EMPLOYEE at the same time.

- This is diagrammed by placing either the letter "**d**" for disjoint or "**o**" for overlapping inside the circle on the Generalization Hierarchy portion of the E-R diagram.

  The two types of constraints on generalization and specialization (Disjointness and Completeness constraints) are not dependent on one another. That is, being disjoint will not favour whether the tuples in the superclass should have Total or Partial participation for that specific specialization.

  From the two types of constraints, we can have four possible constraints

  - Disjoint AND Total
  - Disjoint AND Partial
  - Overlapping AND Total
  - Overlapping AND Partial

Logical database design

- Logical Database Design is the process of translating the conceptual design into a logical database design that can be implemented on a chosen DBMS
  - Input: conceptual model (ERD)
  - Output: relational schema, normalized relations
- The goal of logical database design is to create well-structured tables that properly reflect the company's business environment.

Steps to Build Logical Data Model

- The objective of logical database design is to interpret the conceptual data model into a logical data model and then authorize this model to check whether it is structurally correct and able to support the required transactions or not.
- This objective can be achieved by following the activities (steps) given below:

1. Derive the relations for the logical data model
2. Validate those relations using normalization

3. Validate those relations against user transactions
4. Check key and Integrity Constraints
5. Evaluate the logical data model with user
6. Review logical data models into the global model

1.Derive the relations for the logical data model

- To create relations for the logical data model to represent the entities, relationships, and attributes that have been identified
- In this step we describe how relations are obtained for the following structures that are occur in a conceptual data model:
    - strong entity types
    - weak entity types
    - one-to-many (1:*) binary relationship types
    - one-to-one (1:1) binary relationship types
    - one-to-one (1:1) recursive relationship types
    - superclass/subclass relationship types
    - many-to-many (*:*) binary relationship types
    - complex relationship types
    - multi-valued attributes

2. **Validate those relations using normalization**

- In this step, we have to validate the groupings of attributes in each relation using the rules of normalization.
- The purpose of normalization is to ensure that the position of relations has a minimal and yet sufficient number of attributes necessary to support the data requirements of the enterprise.

3.Validate those relations against user transactions

- The primary purpose of this step is to validate the logical data model to make certain that the model supports the required transactions, as the users' requirements specification.
- By using the relations, the primary key / foreign key links within the relations, the ER diagram, and the data dictionary, you can attempt to perform the operations manually.
- If you can resolve all transactions in this way, you can validate the logical data model against the transactions.
- To ensure relations in logical data model support the required transactions

4. Check key and Integrity Constraints

- To check key and integrity constraints represented in logical data model
  - Keep database accurate, consistent, complete
- This includes identifying:
  - Required data
    - Non null attributes must contain valid value
  - Attribute domain constraints
    - Every attribute has domain
    - Identify a set of values that are legal
  - Multiplicity
    - Constraints on relationships valid
  - Entity integrity
    - Primary key not null
  - Referential integrity
    - Foreign key must reference existing value in parent relation

5. Evaluate the logical data model with user

- To review the final logical data model with users to ensure they consider model to be true and accurate representation of data requirements of enterprise.

6. Merge Logical Data Models into Global Model

- To merge logical data models based on one or more user views into single global logical data model that represents all user views of database.

Normalization

A relational database is merely a collection of data, organized in a particular manner. As the father of the relational database approach, **Codd** created a series of rules called ***normal forms*** that help define that organization

One of the best ways to determine what information should be stored in a database is to clarify what questions will be asked of it and what data would be included in the answers.

Database normalization is a series of steps followed to obtain a database design that allows for consistent storage and efficient access of data in a relational database. These steps reduce data redundancy and the risk of data becoming inconsistent.

NORMALIZATION is the process of identifying the logical associations between data items and designing a database that will represent such associations but without suffering the update anomalies which are;

1. **Insertion Anomalies**
2. **Deletion Anomalies**
3. **Modification Anomalies**

Normalization may reduce system performance since data will be cross referenced from many tables. Thus denormalization is sometimes used to improve performance, at the cost of reduced consistency guarantees.

Normalization normally is considered as good if it is lossless decomposition.

Mnemonic for remembering the rationale for normalization could be the following:

1. No Repeating or Redunduncy: *no repeting fields in the table*
2. The Fields Depend Upon the Key: *the table should solely depend on the key*
3. The Whole Key: *no partial keybdependency*
4. And Nothing But The Key: *no inter data dependency*
5. So Help Me Codd: *the rules of Codd*

All the normalization rules will eventually remove the update anomalies that may exist during data manipulation after the implementation. The update anomalies are;

Pitfalls of Normalization

- ✓ Requires data to see the problems
- ✓ May reduce performance of the system
- ✓ Is time consuming,
- ✓ Difficult to design and apply and
- ✓ Prone to human error

The underlying ideas in normalization are simple enough. Through normalization we want to design for our relational database a set of tables that;

1. Contain all the data necessary for the purposes that the database is to serve,
2. Have as little redundancy as possible,
3. Accommodate multiple values for types of data that require them,
4. Permit efficient updates of the data in the database, and
5. Avoid the danger of losing data unknowingly.

The type of problems that could occur in insufficiently normalized table is called update anomalies which includes;

1. Insertion anomalies

An "insertion anomaly" is a failure to place information about a new database entry into all the places in the database where information about that new entry needs to be stored. In a properly normalized database, information about a new entry needs to be inserted into only one place in the database; in an inadequately normalized database, information about a new entry may need to be inserted into more than one place and, human fallibility being what it is, some of the needed additional insertions may be missed.

2. Deletion anomalies

A "deletion anomaly" is a failure to remove information about an existing database entry when it is time to remove that entry. In a properly normalized database, information about an old, to-be-gotten-rid-of entry needs to be deleted from only one place in the database; in an inadequately normalized database, information about that old entry may need to be deleted from more than one place, and, human fallibility being what it is, some of the needed additional deletions may be missed.

3. Modification anomalies

A modification of a database involves changing some value of the attribute of a table.

In a properly normalized database table, what ever information is modified by the user, the change will be effected and used accordingly.

The purpose of normalization is to reduce the chances for anomalies to occur in a database.

Example of problems related with Anomalies

EmpID	*FName*	*LName*	*SkillID*	*Skill*	*SkillType*	*School*	*SchoolAdd*	*Skill Level*
12	Abebe	Mekuria	2	SQL	Database	AAU	Sidist_Kilo	5
16	Lemma	Alemu	5	C++	Programming	Unity	Gerji	6
28	Chane	Kebede	2	SQL	Database	AAU	Sidist_Kilo	10
25	Abera	Taye	6	VB6	Programming	Helico	Piazza	8
65	Almaz	Belay	2	SQL	Database	Helico	Piazza	9
24	Dereje	Tamiru	8	Oracle	Database	Unity	Gerji	5
51	Selam	Belay	4	Prolog	Programming	Jimma	Jimma City	8
94	Alem	Kebede	3	Cisco	Networking	AAU	Sidist_Kilo	7
18	Girma	Dereje	1	IP	Programming	Jimma	Jimma City	4
13	Yared	Gizaw	7	Java	Programming	AAU	Sidist_Kilo	6

Deletion Anomalies:

If employee with ***ID 16*** is deleted then ever information about skill ***C++*** and the type of skill is deleted from the database. Then we will not have any information about ***C++*** and its skill type.

Insertion Anomalies:

What if we have a new employee with a skill called ***Pascal***? We can not decide weather ***Pascal*** is allowed as a value for skill and we have no clue about the ***type of skill*** that ***Pascal*** should be categorized as.

Modification Anomalies:

What if the address for ***Helico*** is changed fro ***Piazza*** to ***Mexico***? We need to look for every occurrence of ***Helico*** and change the value of ***School_Add*** from ***Piazza*** to ***Mexico***, which is prone to error.

Database-management system can work only with the information that we put explicitly into its tables for a given database and into its rules for working with those tables, where such rules are appropriate and possible.

Functional Dependency (FD)

Before moving to the definition and application of normalization, it is important to have an understanding of "functional dependency."

Data Dependency

The logical association between data items that point the database designer in the direction of a good database design are referred to as determinant or dependent relationships.

Two data items A and B are said to be in a determinant or dependent relationship if certain values of data item B always appear with certain values of data item A. if the data item A is

the determinant data item and B the dependent data item then the direction of the association is from A to B and not vice versa.

The essence of this idea is that if the existence of something, call it A, implies that B must exist and have a certain value, then we say that ***"B is functionally dependent on A."*** We also often express this idea by saying that "A determines B," or that "B is a function of A," or that "A functionally governs B." Often, the notions of functionality and functional dependency are expressed briefly by the statement, "If A, then B." It is important to note that the value B must be *unique* for a given value of A, i.e., any given value of A must imply just one and only one value of B, in order for the relationship to qualify for the name "function." (However, this does not necessarily prevent different values of A from implying the same value of B.)

X➔Y ***holds if whenever two tuples have the same value for X, they must have the same value for Y***

The notation is: A➔B which is read as; B is functionally dependent on A

In general, a ***functional dependency*** is a relationship among attributes. In relational databases, we can have a determinant that governs one other attribute or several other attributes.

FDs are derived from the real-world constraints on the attributes

Example

Dinner Course	***Type of Wine***
Meat	Red
Fish	White
Cheese	Rose

Since the type of ***Wine*** served depends on the type of ***Dinner***, we say ***Wine*** is functionally dependent on ***Dinner***. ***Dinner➔ Wine***

Dinner Course	***Type of Wine***	***Type of Fork***
Meat	Red	Meat fork
Fish	White	Fish fork
Cheese	Rose	Cheese fork

Since both ***Wine*** type and ***Fork*** type are determined by the ***Dinner*** type, we say ***Wine*** is functionally dependent on ***Dinner*** and ***Fork*** is functionally dependent on ***Dinner***.

Dinner➔ Wine
Dinner ➔Fork

Partial Dependency

If an attribute which is not a member of the primary key is dependent on some part of the primary key (if we have composite primary key) then that attribute is partially functionally dependent on the primary key.
Let {A,B} is the Primary Key and C is no key attribute.

Then if **{A,B}➔C and B➔C**

Then C is partially functionally dependent on {A,B}

Full Dependency

If an attribute which is not a member of the primary key is not dependent on some part of the primary key but the whole key (if we have composite primary key) then that attribute is fully functionally dependent on the primary key.

Let {A,B} is the Primary Key and C is no key attribute

Then if **{A,B}➔C and B➔C and A➔C** does not hold

Then C Fully functionally dependent on {A,B}

Transitive Dependency

In mathematics and logic, a transitive relationship is a relationship of the following form: "If A implies B, and if also B implies C, then A implies C."

Example:

If Abebe is a Human, and if every Human is an Animal, then Abebe must be an Animal.

Generalized way of describing transitive dependency is that:

If A functionally governs B, AND If
B functionally governs C
THEN A functionally governs C

Provided that neither C nor B determines A (B /➔ A and C /➔ A) In the normal notation:

{(A➔B) AND (B/➔C)} ==> A/➔C

Steps of Normalization:

We have various levels or steps in normalization called Normal Forms. The level of complexity, strength of the rule and decomposition increases as we move from one lower level Normal Form to the higher.

A table in a relational database is said to be in a certain normal form if it satisfies certain constraints.

normal form below represents a stronger condition than the previous one

Normalization towards a logical design consists of the following steps:

UnNormalized Form:

Identify all data elements

First Normal Form:

Find **the key** with which you can find **all** data

Second Normal Form:

Remove part-key dependencies. Make all data dependent on **the whole key**.

Third Normal Form

Remove non-key dependencies. Make all data dependent on **nothing but the key**. For most practical purposes, databases are considered normalized if they adhere to third normal form.

First Normal Form (1NF)

Requires that all column values in a table are ***atomic*** (e.g., a number is an atomic value, while a list or a set is not).
We have two ways of achieving this:

1. Putting each repeating group into a separate table and connecting them with a ***primary key-foreign key*** relationship
2. Moving this repeating groups to a new row by repeating the common attributes. If so then Find the key with which you can find all data

Definition of a table (relation) in 1NF
If

- **There are no duplicated rows in the table. Unique identifier**
- **Each cell is single-valued (i.e., there are no repeating groups).**
- **Entries in a column (attribute, field) are of the same kind.**

Example for First Normal form (1NF)

UNNORMALIZED

EmpID	***FirstName***	***LastName***	***Skill***	***SkillType***	***School***	***SchoolAdd***	***SkillLevel***
12	Abebe	Mekuria	SQL, VB6	Database, Programming	AAU, Helico	Sidist_Kilo Piazza	5 8
16	Lemma	Alemu	C++ IP	Programming Programming	Unity Jimma	Gerji Jimma City	6 4
28	Chane	Kebede	SQL	Database	AAU	Sidist_Kilo	10
65	Almaz	Belay	SQL Prolog Java	Database Programming Programming	Helico Jimma AAU	Piazza Jimma City Sidist_Kilo	9 8 6
24	Dereje	Tamiru	Oracle	Database	Unity	Gerji	5
94	Alem	Kebede	Cisco	Networking	AAU	Sidist_Kilo	7

FIRST NORMAL FORM (1NF)

Remove all repeating groups. Distribute the multi-valued attributes into different rows and identify a unique identifier for the relation so that is can be said is a relation in relational database.

EmpID	***FirstName***	***LastName***	***SkillID***	***Skill***	***SkillType***	***School***	***SchoolAdd***	***SkillLevel***
12	Abebe	Mekuria	1	SQL	Database	AAU	Sidist_Kilo	5
12	Abebe	Mekuria	3	VB6	Programming	Helico	Piazza	8
16	Lemma	Alemu	2	C++	Programming	Unity	Gerji	6
16	Lemma	Alemu	7	IP	Programming	Jimma	Jimma City	4
28	Chane	Kebede	1	SQL	Database	AAU	Sidist_Kilo	10
65	Almaz	Belay	1	SQL	Database	Helico	Piazza	9
65	Almaz	Belay	5	Prolog	Programming	Jimma	Jimma City	8
65	Almaz	Belay	8	Java	Programming	AAU	Sidist_Kilo	6
24	Dereje	Tamiru	4	Oracle	Database	Unity	Gerji	5
94	Alem	Kebede	6	Cisco	Networking	AAU	Sidist_Kilo	7

Second Normal form 2NF

No partial dependency of a non-key attribute on part of the primary key. This will result in a set of relations with a level of Second Normal Form.

Any table that is in 1NF and has a single-attribute (i.e., a non-composite) key is automatically also in 2NF.

Definition of a table (relation) in 2NF

- **It is in 1NF and**
- **If all non-key attributes are dependent on all of the key. i.e. no partial dependency.**

Since a partial dependency occurs when a non-key attribute is dependent on only a part of the (composite) key, the definition of 2NF is sometimes phrased as, "A table is in 2NF if it is in 1NF and if it has no partial dependencies."

Example for 2NF:

EMP_PROJ

EmpID	***EmpName***	***ProjNo***	***ProjName***	***ProjLoc***	***ProjFund***	***ProjMangID***

EMP_PROJ rearranged

EmpID	***ProjNo***	***EmpName***	***ProjName***	***ProjLoc***	***ProjFund***	***ProjMangID***

This schema is in its 1NF since we don't have any repeating groups or attributes with multi-valued property. To convert it to a 2NF we need to remove all partial dependencies of non-key attributes on part of the primary key.

{EmpID, ProjNo}➔***EmpName, ProjName, ProjLoc, ProjFund, ProjMangID***

But in addition to this we have the following dependencies

EmpID➔EmpName
ProjNo➔ProjName, ProjLoc, ProjFund, ProjMangID

As we can see some non-key attributes are partially dependent on some part of the primary key. Thus, these collections of attributes should be moved to a new relation.

EMPLOYEE PROJECT EMP_PROJ

EmpID	***EmpName***

ProjNo	***ProjName***	***ProjLoc***	***ProjFund***	***ProjMangID***

EmpID	***ProjNo***

Third Normal Form (3NF)

Eliminate Columns Not Dependent on Key - If attributes do not contribute to a description of the key, remove them to a separate table. This level avoids update and delete anomalies.

Def of a Table (Relation) in 3NF

- **It is in 2NF and**
- **There are no transitive dependencies between attributes.**

Example for (3NF)

Assumption: Students of same batch (same year) live in one building or dormitory

STUDENT

StudID	***Stud_F_Name***	***Stud_L_Name***	***Dept***	***Year***	***Dormitary***
125/97	Abebe	Mekuria	***Info Sc***	***1***	***401***
654/95	Lemma	Alemu	***Geog***	***3***	***403***
842/95	Chane	Kebede	***CompSc***	***3***	***403***
165/97	Alem	Kebede	***InfoSc***	***1***	***401***
985/95	Almaz	Belay	***Geog***	***3***	***403***

his schema is in its 2NF since the primary key is a single attribute.

Let's take ***StudID***, ***Year*** and ***Dormitary and see the dependencies.***

StudID➔Year AND ***Year➔Dormitary***

Then transitively ***StudID➔Dormitary***

To convert it to a 2NF we need to remove all partial dependencies of non key attributes on part of the primary key.

STUDENT

StudID	***Stud F_Name***	***Stud L_Name***	***Dept***	***Year***
125/97	Abebe	Mekuria	***Info Sc***	***1***
654/95	Lemma	Alemu	***Geog***	***3***
842/95	Chane	Kebede	***CompSc***	***3***
165/97	Alem	Kebede	***InfoSc***	***1***
985/95	Almaz	Belay	***Geog***	***3***

DORM

Year	***Dormitary***
1	***401***
3	***403***

Boyce-Codd Normal Form (BCNF):

Isolate Independent Multiple Relationships - No table may contain two or more 1:n or N:M relationships that are not directly related.

The correct solution, to cause the model to be in 4th normal form, is to ensure that all M:M relationships are resolved independently if they are indeed independent, as shown below.

Def: A table is in BCNF if it is in 3NF and if every determinant is a candidate key.

Forth Normal form (4NF)

Isolate Semantically Related Multiple Relationships - There may be practical constrains on information that justify separating logically related many-to-many relationships.

Def: A table is in 4NF if it is in BCNF and if it has no multi-valued dependencies.

Fifth Normal Form (5NF)

A model limited to only simple (elemental) facts, as expressed in ORM.

Def: A table is in 5NF, also called "Projection-Join Normal Form" (PJNF), if it is in 4NF and if every join dependency in the table is a consequence of the candidate keys of the table.

Domain-Key Normal Form (DKNF)

A model free from all modification anomalies.

Def: A table is in DKNF if every constraint on the table is a logical consequence of the definition of keys and domains.

Physical Database Design Methodology for Relational Database

- The Logical database design is concerned with the *what*;
- The Physical database design is concerned with the *how*.
- Physical database design is the process of producing a description of the implementation of the database on secondary storage. It describes the base relations, file organization, and indexes used to achieve effective access to the data along with any associated integrity constraints and security measures.
- Physical design describes the base relation, file organization, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.
- Sources of information for the physical design process include global logical data model and documentation that describes model.
- Describes the storage structures and access methods used to achieve efficient access to the data
- Knowledge of the DBMS that is selected to host the database systems, with all its functionalities, is required since functionalities of current DBMS vary widely.

Steps in physical database design

1. Translate logical data model for target DBMS

- To determine the file organizations and access methods that will be used to store the base relations; i.e. the way in which relations and tuples will be held on secondary storage
- To decide how to represent the base relations we have identified in the global logical data model in the target DBMS.
- Design enterprise constraints for target DBMS

1.1. Design base relation

1.2. Design representation of derived data

1.3. Design enterprise constraint

2. Design physical representation

2.1. Analyze transactions

To understand the functionality of the transactions that will run on the database and to analyze the important transactions

2.2. Choose file organization

To determine an efficient file organization for each base relation

2.3. Choose indexes

2.4. Estimate disk space and system requirement

To estimate the amount of disk space that will be required by the database

3. Design user view

To design the user views that were identified in the conceptual database design methodology

4. Design security mechanisms

5. Consider controlled redundancy

To determine whether introducing redundancy in a controlled manner by relaxing the normalization rules will improve the performance of the system.

6. Monitor and tune the operational system

Design access rules

To design the access rules to the base relations and user views

1. Translate logical data model for target DBMS

This phase is the translation of the global logical data model to produce a relational database schema in the target DBMS. This includes creating the data dictionary based on the logical model and information gathered. After the creation of the data dictionary, the next activity is to understand the

functionality of the target DBMS so that all necessary requirements are fulfilled for the database intended to be developed.

Knowledge of the DBMS includes:

- how to create base relations
- whether the system supports:
  - definition of Primary key
  - definition of foreign key
  - definition of Alternate key
  - definition of Domains
  - Referential integrity constraints
  - definition of enterprise level constraints

1.1. Design base relation

Designing base relation involves identification of all necessary requirements about a relation starting from the name up to the referential integrity constraints.

The implementation of the physical model is dependent on the target DBMS since some has more facilities than the other in defining database definitions. The base relation design along with every justifiable reason should be fully documented.

1.2. Design representation of derived data

While analyzing the requirement of users, we may encounter that there are some attributes holding data that will be derived from existing or other attributes. A decision on how to represent such data should be devised.

Most of the time derived attributes are not expressed in the logical model but will be included in the data dictionary. Whether to store stored attributes

in a base relation or calculate them when required is a decision to be made by the designer considering the performance impact.

The representation of derived attributes should be fully documented.

1.3. Design enterprise constraint

Data in the database is not only subjected to constraints on the database and the data model used but also with some enterprise dependent constraints. This constraint definition is also dependent on the DBMS selected and enterprise level requirements.

All the enterprise level constraints and the definition method in the target DBMS should be fully documented.

2. Design physical representation

This phase is the level for determining the optimal file organizations to store the base relations and indexes that are required to achieve acceptable performance, that is, the way in which relations and tuples will be held on the secondary storage.

2.1. Analyze transactions

- To understand the functionality of the transactions that will run on the database and to analyze the important transactions.

2.2. Choose file organization

- To determine whether adding indexes will improve the performance of the system.

2.3. Choose indexes

- To determine an efficient file organization for each base relation. **Example**. If we want to retrieve staff tuples in alphabetical order of name, sorting the file by staff name is a good file organization.
- File organizations include Heap, Hash, Indexed Sequential Access Method (ISAM), B+-Tree, and Clusters.

2.4. Estimate disk space and system requirement

- To estimate the amount of **disk space** that will be required by the database.
- The estimation based on
    - Tuple size
    - Number of tuple
    - Growth factor

3. Design user view

- To design the user views that were identified during the Requirements Collection and Analysis stage of the database system development lifecycle.
    - Director
    - Manager
    - Supervisor
    - Assistance

4. Design security mechanisms

- To design the security measures for the database as specified by the users.
- System security
    - Username and password
- Data security
    - Access and use of database object (relation and object)

5. Consider controlled redundancy

- To determine whether introducing redundancy in a controlled manner by relaxing normalization rules will improve the performance of the system.
- Redundancy is when the same fact is stored multiple times in several places in a database.
    - For example, say the name of the student with StudentID=28 is XY is stored multiple times.
- Redundancy is controlled when the RDBMS ensures that multiple copies of the same data are consistent.
    - For example, if a new record with StudentID=28 is stored in the database, the RDBMS will ensure that this record is for Student XY.
- If the RDBMS has no control over this, we have uncontrolled redundancy.

6. Monitor and tune the operational system

- To monitor operational system and improve performance of system to correct inappropriate design decisions or reflect changing requirements
- Number of factors may be used to measure efficiency:
  - **Transaction throughput**: number of transactions processed in given time interval.
  - **Response time**: elapsed time for completion of a single transaction.
  - **Disk storage**: amount of disk space required to store database files.
- Many RDBMSs provide the Database Administrator (DBA) with utilities to monitor and tune the operation of the system.
- Benefits of tuning include:
  - Tuning can avoid the procurement of additional hardware.
  - It may be possible to downsize the hardware configuration.
  - This results in less and cheaper hardware and consequently less expensive maintenance.
  - A well-tuned system produces faster response times and better throughput.

DBMS Storage System

- Databases are stored in file formats, which contain records. At physical level, the actual data is stored in electromagnetic format on some device. These storage devices can be broadly categorized into three types.

1. Primary Storage:

- The memory storage, which is directly accessible by the CPU.
- CPU's internal memory (registers), fast memory (cache) and main memory (RAM) are directly accessible to CPU as they all are placed on the motherboard or CPU chipset.
- This storage is typically very small, ultra fast and volatile.
- It needs continuous power supply in order to maintain its state, i.e. in case of power failure all data are lost.

2. Secondary Storage:

- Secondary storage devices are used to store data for future use or as backup.
- it is 'non-volatile storage, i.e., unlike primary storage, it doesn't lose data if the system restarts, crashes, or gets a power cut.
- All memory devices, which are not part of CPU chipset or motherboard comes under this category.

- Hard disk drives, which contain the operating system and generally not removed from the computers are, considered secondary storage and all other are called tertiary storage.

3. **Tertiary Storage:**
   - Third level in memory hierarchy is called tertiary storage.
   - This is used to store huge amount of data. since this storage is external to the computer system, it is the slowest in speed.
   - These storage devices are mostly used to backup the entire system.
   - Optical disk and magnetic tapes are widely used storage devices as tertiary storage.

DBMS File Structure

- A database consist of a huge amount of data. The data is grouped within a table in RDBMS, and each table have related records.
- A user can see that the data is stored in form of tables, but in actual this huge amount of data is stored in physical memory in form of files.
- **File –** A file is named collection of related information that is recorded on storage.

What is File Organization?

- It refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record.
- In simple terms, Storing the files in certain order is called file Organization.
- **File Structure** refers to the format of the label and data blocks and of any logical control record.
- We have four types of File Organization to organize file records

1. **Heap File Organization**
   - When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details.
   - File records can be placed anywhere in that memory area.
   - It is the responsibility of the software to manage the records.
   - Heap File does not support any ordering, sequencing, or indexing on its own.
2. **Sequential File Organization**
   - It is one of the simple methods of file organization. Here each file/records are stored one after the other in a sequential manner.
   - Every file record contains a data field (attribute) to uniquely identify that record.
   - In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. This can be achieved in **two** ways:
   - Records are stored one after the other as they are inserted into the tables. This method is called **pile file method**.
   - When a new record is inserted, it is placed at the end of the file.
   - In the case of any modification or deletion of record, the record will be searched in the memory blocks.
   - In the second method, records are sorted (either ascending or descending) each time they are inserted into the system.
   - This method is called **sorted file method**. Sorting of records may be based on the primary key or on any other columns.
   - Whenever a new record is inserted, it will be inserted at the end of the file and then it will sort – ascending or descending based on key value and placed at the correct position.
   - In the case of update, it will update the record and then sort the file to place the updated record in the right place. Same is the case with delete.
3. **Hash File Organization**
   - Hash File Organization uses Hash function computation on some fields of the records.
   - The output of the hash function determines the location of disk block where the records are to be placed.

4. **Clustered File Organization**
    - Clustered file organization is not considered good for large databases.
    - In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

File Operations

- Operations on database files can be broadly classified into two categories
    - **Update Operations**
    - **Retrieval Operations**
- Update operations change the data values by insertion, deletion, or update.
- Retrieval operations, on the other hand, do not alter the data but retrieve them after optional conditional filtering.
- In both types of operations, selection plays a significant role.

Indexing and Hashing

- **Indexing** is a data structure technique which allows you to quickly retrieve records from a database file.
- An Index is a small table having only two columns.

    - The first column comprises a copy of the primary or candidate key of a table.
    - Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.
- An index
    - Takes a search key as input
    - Efficiently returns a collection of matching records.

Indexing can be of the following types −

- **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a **key field**.
- The key field is generally the primary key of the relation.

- **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

  Ordered Indexing is of two types −

  - Sparse Index
  - Dense Index
- In dense index, there is an index record for every search key value in the database.
- This makes searching faster but requires more space to store index records itself.
- Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index

- In sparse index, index records are not created for every search key.
- Sparse Index: contains index records for only some search-key values.
- Applicable when records are sequentially ordered on search-key
- Compared to dense indices Less space and less maintenance overhead for insertions and deletions

Multilevel Index

- Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices.
- There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses
- Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory
- A B tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B tree denote actual data pointers.

What is Hashing in DBMS?

- In a huge database structure, it is very inefficient to search all the index values and reach the desired data.
- In RDBMS, Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.
- Hashing method is used to index and retrieve items in a database as it is faster to search that specific item using the shorter hashed key instead of using its original value.
- Data is stored in the form of data blocks whose address is generated by applying a hash function in the memory location where these records are stored

Types of Hashing Techniques

There are mainly two types of SQL hashing methods/techniques:

1. Static Hashing

- In static hashing, when a search-key value is provided, the hash function always computes the same address.
    - For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address shall always be same for that function.
- Therefore, in this static hashing method, the number of data buckets in memory always remains constant at all times.

2. Dynamic Hashing

- The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks.
- It offers a mechanism in which data buckets are added and removed dynamically and on demand.
- In this hashing, the hash function helps you to create a large number of values.

Relational Query Languages

- ***Query languages:*** Allow manipulation and retrieval of data from a database.
- Query Languages!**=** programming languages!
- QLs not intended to be used for complex calculations.
- QLs support easy, efficient access to large data sets.
- Relational model supports simple, powerful query languages.

Formal Relational Query Languages

- There are varieties of Query languages used by relational DBMS for manipulating relations.
- Some of them are **procedural**
- User tells the system exactly ***what*** and ***how*** to manipulate the data
- Others are **non-procedural**
- User states ***what*** data is needed rather than ***how*** it is to be retrieved.

Two mathematical Query Languages form the basis for Relational languages

Relational Algebra**:**
Relational Calculus**:**

- We may describe the ***relational algebra as procedural language***: it can be used to tell the DBMS how to build a new relation from one or more relations in the database.

- We may describe ***relational calculus as a non-procedural language***: it can be used to formulate the definition of a relation in terms of one or more database relations.
- Formally the relational algebra and relational calculus are equivalent to each other. ***For every expression in the algebra, there is an equivalent expression in the calculus.***
- Both are non-user-friendly languages. They have been used as the basis for other, higher-level data manipulation languages for relational databases.

A query is applied to relation instances, and the result of a query is also a relation instance.

- *Schemas* of input relations for a query are fixed
- The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.

Relational Algebra

The basic set of operations for the relational model is known as the relational algebra. These operations enable a user to specify basic retrieval requests.

The result of the retrieval is a new relation, which may have been formed from one or more relations. The **algebra operations** thus produce new relations, which can be further manipulated using operations of the same algebra.

A sequence of relational algebra operations forms a **relational algebra expression**, whose result will also be a relation that represents the result of a database query (or retrieval request).

- Relational algebra is a theoretical language with operations that work on one or more relations to define another relation without changing the original relation.
- The output from one operation can become the input to another operation
  - (Nesting is possible)

There are different basic operations that could be applied on relations on a database based on the requirement.

- ***Selection*** **(σ)** Selects a subset of rows from a relation.
- ***Projection*** **(π)** Deletes unwanted columns from a relation.
- ***Renaming:*** assigning intermediate relation for a single operation
- ***Cross-Product (x)*** Allows us to combine two relations.
- ***Set-Difference (-)*** Tuples in relation1, but not in relation2.
- ***Union (∪)*** Tuples in relation1 or in relation2.
- ***Intersection (∩)*** Tuples in relation1 and in relation2
- ***Join*** ⋈ Tuples joined from two relations based on a condition

- Using these we can build up sophisticated database queries.

Table1:

Sample table used to illustrate different kinds of relational operations. The relation contains information about employees, IT skills they have and the school where they attend each skill.

Employee

EmpID	***FName***	***LName***	***SkillID***	***Skill***	***SkillType***	***School***	***SchoolAdd***	***SkillLevel***
12	Abebe	Mekuria	2	SQL	Database	AAU	Sidist_Kilo	5
16	Lemma	Alemu	5	C++	Programming	Unity	Gerji	6
28	Chane	Kebede	2	SQL	Database	AAU	Sidist_Kilo	10
25	Abera	Taye	6	VB6	Programming	Helico	Piazza	8
65	Almaz	Belay	2	SQL	Database	Helico	Piazza	9
24	Dereje	Tamiru	8	Oracle	Database	Unity	Gerji	5
51	Selam	Belay	4	Prolog	Programming	Jimma	Jimma City	8
94	Alem	Kebede	3	Cisco	Networking	AAU	Sidist_Kilo	7
18	Girma	Dereje	1	IP	Programming	Jimma	Jimma City	4
13	Yared	Gizaw	7	Java	Programming	AAU	Sidist_Kilo	6

Selection

- ✓ Selects subset of tuples/rows in a relation that satisfy ***selection condition***.
- ✓ Selection operation is a unary operator (it is applied to a single relation)
- ✓ The Selection operation is applied to each tuple individually
- ✓ The degree of the resulting relation is the same as the original relation but the cardinality (no. of tuples) is less than or equal to the original relation.
- ✓ The Selection operator is commutative.
- ✓ Set of conditions can be combined using Boolean operations (∧**(AND)**, ∨**(OR)**, and ~**(NOT)**)
- ✓ No duplicates in result!
- ✓ *Schema* of result identical to schema of (only) input relation.
- ✓ *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)
- ✓ It is a filter that keeps only those tuples that satisfy a qualifying condition
    - (Those satisfying the condition are selected while others are discarded.)

Notation:

σ *<Selection Condition>* ***<Relation Name>***

Example: Find all Employees with skill type of Database.

σ *< SkillType ="Database">* ***(Employee)***

This query will extract every tuple from a relation called Employee with all the attributes where the SkillType attribute with a value of "Database".

The resulting relation will be the following.

EmpID	*FName*	*LName*	*SkillID*	*Skill*	*SkillType*	*School*	*SchoolAdd*	*SkillLevel*
12	Abebe	Mekuria	2	SQL	Database	AAU	Sidist_Kilo	5
28	Chane	Kebede	2	SQL	Database	AAU	Sidist_Kilo	10
65	Almaz	Belay	2	SQL	Database	Helico	Piazza	9
24	Dereje	Tamiru	8	Oracle	Database	Unity	Gerji	5

If the query is all employees with a SkillType *Database* and School *Unity* the relational algebra operation and the resulting relation will be as follows.

σ *< SkillType ="Database" AND School="Unity">* ***(Employee)***

EmpID	*FName*	*LName*	*SkillID*	*Skill*	*SkillType*	*School*	*SchoolAdd*	*SkillLevel*
24	Dereje	Tamiru	8	Oracle	Database	Unity	Gerji	5

Projection

- ✓ Selects certain attributes while discarding the other from the base relation.
- ✓ The PROJECT creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns.
- ✓ Deletes attributes that are not in *projection list*.
- ✓ *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.

- ✓ Projection operator has to eliminate *duplicates*!
- ✓ Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it.
- ✓ If the Primary Key is in the *projection list*, then duplication will not occur
- ✓ Duplication removal is necessary to insure that the resulting table is also a relation.

Notation:

π ***<Selected Attributes>*** ***<Relation Name>***

Example: To display Name, Skill, and Skill Level of an employee, the query and the resulting relation will be:

π ***<FName, LName, Skill, Skill_Level>*** ***(Employee)***

FName	***LName***	***Skill***	***SkillLevel***
Abebe	Mekuria	SQL	5
Lemma	Alemu	C++	6
Chane	Kebede	SQL	10
Abera	Taye	VB6	8
Almaz	Belay	SQL	9
Dereje	Tamiru	Oracle	5
Selam	Belay	Prolog	8
Alem	Kebede	Cisco	7
Girma	Dereje	IP	4
Yared	Gizaw	Java	6

If we want to have the Name, Skill, and Skill Level of an employee with Skill SQL and SkillLevel greater than 5 the query will be: π***<FName, LName, Skill, Skill_Level>***

(σ***<****Skill****="SQL" ∧ SkillLevel>5>****(Employee))***

FName	***LName***	***Skill***	***SkillLevel***
Chane	Kebede	SQL	10
Almaz	Belay	SQL	9

Rename Operation

- We may want to apply several relational algebra operations one after the other. The query could be written in two different forms:
    1. Write the operations as a single relational algebra expression by nesting the operations.
    2. Apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results➔Rename Operation

If we want to have the Name, Skill, and Skill Level of an employee with salary greater than 1500 and working for department 5, we can write the expression for this query using the two alternatives:

1. A single algebraic expression:

The above used query is using a single algebra operation, which is:

$$\pi_{<FName,\ LName,\ Skill,\ Skill\_Level>}\ (\sigma_{<Skill="SQL" \wedge SkillLevel>5>}(Employee))$$

2. Using an intermediate relation by the Rename Operation:

$$Step1:\quad Result1 \leftarrow \sigma_{<DeptNo=5 \wedge Salary>1500>}(Employee)$$

$$Step2:\quad Result \leftarrow \pi_{<FName,\ LName,\ Skill,\ Skill\_Level>}(Result1)$$

Then Result will be equivalent with the relation we get using the first alternative.

UNION Operation

The result of this operation, denoted by R U S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
The two operands must be "type compatible".

Type Compatibility

The operand relations $R_1(A_1, A_2, ..., A_n)$ and $R_2(B_1, B_2, ..., B_n)$ must have the same number of attributes, and the domains of corresponding attributes must be compatible; that is, Dom (A_i)=Dom(B_i) for i=1, 2, ..., n.

The resulting relation for;

- R1 ∪ R2,
- R1 ∩ R2, or • R1-R2 has the same attribute names as the first operand relation R1 (by convention).

INTERSECTION Operation

The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that are in both R and S. The two operands must be "*type compatible*"

Set Difference (or MINUS) Operation

The result of this operation, denoted by R - S, is a relation that includes all tuples that are in R but not in S. The two operands must be "*type compatible*".

Some Properties of the Set Operators

Notice that both union and intersection are commutative operations; that is

$R \cup S = S \cup R$, *and* $R \cap S = S \cap R$

Both union and intersection can be treated as n-nary operations applicable to any number of relations as both are associative operations; that is $R \cup (S \cup T) = (R \cup S) \cup T$, *and* $(R \cap S) \cap T = R \cap (S \cap T)$

The minus operation is not commutative; that is, in general $R - S \neq S - R$

CARTESIAN (cross product) Operation

This operation is used to combine tuples from two relations in a combinatorial fashion. That means, every tuple in Relation1(R) one will be related with every other tuple in Relation2 (S).

In general, the result of $R(A_1, A_2, \ldots, A_n) \mathbf{X} S(B_1, B_2, \ldots, B_m)$ is a relation Q with degree $n + m$ attributes $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$, in that order.

Where R has n attributes and S has m attributes.

The resulting relation Q has one tuple for each combination of tuples—one from R and one from S.

Hence, if R has n tuples, and S has m tuples, then | R x S | will have $n * m$ tuples.

The two operands do NOT have to be "*type compatible*"

Example

Employee

ID	FName	LName
123	Abebe	Lemma
567	Belay	Taye
822	Kefle	Kebede

Dept

DeptID	DeptName	MangID
2	Finance	567
3	Personnel	123

Then Employee X Dept will be of the form:

ID	FName	LName	DeptID	DeptName	MangID
123	Abebe	Lemma	2	Finance	567
123	Abebe	Lemma	3	Personnel	123
567	Belay	Taye	2	Finance	567
567	Belay	Taye	3	Personnel	123
822	Kefle	Kebede	2	Finance	567
822	Kefle	Kebede	3	Personnel	123

Then to extract employee information about managers of the departments, the algebra query and the resulting relation will be.

$\pi_{<ID, FName, LName, DeptName>} (\sigma_{<ID=MangID>}(Employee\ X\ Dept))$

ID	FName	LName	DeptName
123	Abebe	Lemma	Personnel
567	Belay	Taye	Finance

JOIN Operation

The sequence of Cartesian product followed by select is used quite commonly to identify and select related tuples from two relations, a special operation, called ***JOIN***. Thus, in JOIN operation, the Cartesian Operation and the Selection Operations are used together.

JOIN Operation is denoted by a ⋈ symbol.

This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations.
The general form of a join operation on two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$ is:

$R \bowtie_{<join\ condition>} S$ ***Is equivalent to*** $\sigma_{<selection\ condition>}(R \times S)$

where <join condition> *and* ***<selection condition>*** *are the same*

Where R and S can be any relations that result from general relational algebra expressions. Since JOIN function in two relation, it is a Binary operation.

This type of JOIN is called a ***THETA JOIN (θ - JOIN)***

Where θ is the logical operator used in the join condition.

θ Could be { <, ≤ , >, ≥, ≠, = }

Example:
Thus in the above example we want to extract employee information about managers of the departments, the algebra query using the JOIN operation will be.

$Employee \bowtie_{<ID=MangID>} Dept$

EQUIJOIN Operation

The most common use of join involves join conditions with equality comparisons only (=). Such a join, where the only comparison operator used is called an

EQUIJOIN. In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple since we used the equality logical operator.

For example, the above JOIN expression is an EQUIJOIN since the logical operator used is the equal to operator (=).

NATURAL JOIN Operation

We have seen that in EQUIJOIN one of each pair of attributes with identical values is extra, a new operation called **natural join** was created to get rid of the second (or extra) attribute that we will have in the result of an EQUIJOIN condition.

The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, have the same name in both relations. If this is not the case, a renaming operation on the attributes is applied first.

OUTER JOIN Operation

OUTER JOIN is another version of the JOIN operation where non matching tuples from the first Relation are also included in the resulting Relation where attributes of the second Relation for a non matching tuples from Relation one will have a value of NULL.

Notation:

$$R \; \supset\!\bowtie\!\triangleright_{\text{<Join Condition >}} \; S$$

When two relations are joined by a JOIN operator, there could be some tuples in the first relation not having a matching tuple from the second relation, and the query is interested to display these non matching tuples from the first relation. Such query is represented by the OUTER JOIN.

SEMIJOIN Operation

SEMI JOIN is another version of the JOIN operation where the resulting Relation will contain those attributes of Relation one that are related with tuples in the second Relation.

$$R \triangleright_{\text{<Join Condition>}} S$$

Relational Calculus

A relational calculus expression creates a new relation, which is specified in terms of variables that range over rows of the stored database relations (in tuple calculus) or over columns of the stored relations (in domain calculus).

In a calculus expression, there is no order of operations to specify how to retrieve the query result. A calculus expression specifies only ***what*** information the result should contain rather than ***how*** to retrieve it.

In Relational calculus, there is no description of how to evaluate a query, this is the main distinguishing feature between relational algebra and relational calculus.

Relational calculus is considered to be a nonprocedural language. This differs from relational algebra, where we must write a sequence of operations to specify a retrieval request; hence relational algebra can be considered as a procedural way of stating a query.

When applied to relational database, the calculus is not that of derivative and differential but in a form of first-order logic or predicate calculus, a predicate is a truth-valued function with arguments.

When we substitute values for the arguments in the predicate, the function yields an expression, called a *proposition*, which can be either true or false.

If a predicate contains a variable, as in *'x is a member of staff'*, there must be a *range for x*. When we substitute some values of this range for x, the proposition may be true; for other values, it may be false.

If COND is a predicate, then the set off all tuples evaluated to be true for the predicate COND will be expressed as follows:

$$\{t \mid COND(t)\}$$

Where t is a tuple variable and $COND\ (t)$ is a conditional expression involving t. The result of such a query is the set of all tuples t that satisfy $COND\ (t)$.

If we have set of predicates to evaluate for a single query, the predicates can be connected using $\wedge$**(AND)**, $\vee$**(OR**), and **~(NOT)**

A relational calculus expression creates a new relation, which is specified in terms of variables that range over rows of the stored database relations (in tuple calculus) or over columns of the stored relations (in domain calculus).

Tuple-oriented Relational Calculus

- The tuple relational calculus is based on specifying a number of tuple variables. Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation.
- Tuple relational calculus is interested in finding tuples for which a predicate is true for a relation. Based on use of tuple variables.
- Tuple variable is a variable that *'ranges over'* a named relation: that is, a variable whose only permitted values are tuples of the relation.
- If E is a tuple that ranges over a relation employee, then it is represented as ***EMPLOYEE(E)*** i.e. ***Range of E is EMPLOYEE***
- Then to extract all tuples that satisfy a certain condition, we will represent is as all tuples E such that COND(E) is evaluated to be true.

$$\{E / COND(E)\}$$

The predicates can be connected using the Boolean operators:

$\wedge$ **(AND),** $\vee$ **(OR), ~ (NOT)**

COND(t) is a formula, and is called a Well-Formed-Formula (WFF) if:

- Where the COND is composed of n-nary predicates (formula composed of n single predicates) and the predicates are connected by any of the Boolean operators.
- And each predicate is of the form A θ B and θ is one of the logical operators *{* **<, ≤ , >, ≥, ≠, =** *}*which could be evaluated to either true or false. And A and B are either constant or variables.
- Formulae should be unambiguous and should make sense.

Example (Tuple Relational Calculus)

- Extract all employees whose skill level is greater than or equal to 8

{E | Employee(E) ∧ E.SkillLevel >= 8}

EmpID	*FName*	*LName*	*SkillID*	*Skill*	*SkillType*	*School*	*SchoolAdd*	*SkillLevel*
28	Chane	Kebede	2	SQL	Database	AAU	Sidist_Kilo	10
25	Abera	Taye	6	VB6	Programming	Helico	Piazza	8
65	Almaz	Belay	2	SQL	Database	Helico	Piazza	9
51	Selam	Belay	4	Prolog	Programming	Jimma	Jimma City	8

- To find only the EmpId, FName, LName, Skill and the School where the skill is attended where of employees with skill level greater than or equal to 8, the tuple based relational calculus expression will be:

{E.EmpId, E.FName, E.LName, E.Skill, E.School | Employee(E) ∧ E.SkillLevel >= 8}

EmpID	*FName*	*LName*	*Skill*	*School*
28	Chane	Kebede	SQL	AAU
25	Abera	Taye	VB6	Helico
65	Almaz	Belay	SQL	Helico
51	Selam	Belay	Prolog	Jimma

- E.FName means the value of the First Name (FName) attribute for the tuple E.

Quantifiers in Relation Calculus

- To tell how many instances the predicate applies to, we can use the two quantifiers in the predicate logic.
  - One relational calculus expressed using Existential Quantifier can also be expressed using Universal Quantifier.

1. Existential quantifier ∃ ('there exists')

Existential quantifier used in formulae that must be true for at least one instance, such as:

An employee with skill level greater than or equal to 8 will be: **{E | Employee(E) ∧ (∃E)(E.SkillLevel >= 8)}**

This means, there exist at least one tuple of the relation employee where the value for the SkillLevel is greater than or equal to 8

2. Universal quantifier ∀ ('for all')

Universal quantifier is used in statements about every instance, such as:
An employee with skill level greater than or equal to 8 will be:

{E | Employee(E) ∧ (∀E)(E.SkillLevel >= 8)}

This means, for all tuples of relation employee where value for the SkillLevel attribute is greater than or equal to 8.

Example:

Let's say that we have the following Schema (set of Relations)

Employee(EID, FName, LName, Dept) Project(PID, PName, Dept) Dept(DID, DName, DMangID) WorksOn(EID, PID)

To find employees who work on projects controlled by department 5 the query will be:

{E | Employee(E) ∧ (∀x)(Project(x) ∧ (∃w)(WorksOn(w) ∧ x.Dept=5 ∧ E.EID=W.EID))}

Structured Query Languages(SQL)

- SQL is the most widely implemented standard language for relational databases.
- SQL is a computer language for storing, manipulating and retrieving data stored in relational database.
- SQL not only allows you to manage data within the database, but also manage the database itself.
- SQL supports the creation and maintenance of the relational database and the management of data within that database.
- All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL standard database language.

Database Field Types in SQL

- Each column in a database table is required to have a name and a data type.
- The data type indicates what kind of data can be stored ,thus setting the size of that location
- The SQL standard supports a variety of built-in domain types, including:
    - **Char(n)**: A fixed-length character string with user-specified length n.
    - **Varchar(n)**: A variable-length character string with user-specified maximum length n.
    - **Int**: An integer (a finite subset of the integers that is machine dependent).
    - ***Numeric(p, d): A fixed-point number with user-specified precision. The number*** consists of *p digits (plus a sign), and d of the p digits are to the right of the* decimal point. Thus, **numeric(3,1) allows 44.*5 to be stored exactly, but neither*** 444.*5 or 0.32 can be stored exactly in a field of this type.*
    - ***Real, double precision: Floating-point and double-precision floating-point*** numbers with machine-dependent precision.
    - ***Float(n): A floating-point number, with precision of at least n digits.***
    - ***Date: A calendar date containing a (four-digit) year, month, and day of the*** month.

SQL Constraints

- **Constraints are the rules enforced on data columns on table.**
- **These are** used to limit the type of data can go into a table.
- This ensures the accuracy and reliability of the data in the database.
- Commonly used constraint variables in SQL are:
    - **NOT NULL Constraint**: ensures that a column cannot have null value.
    - **UNIQUE Constraint**: ensures that all values in a column are different.
    - **PRIMARY KEY**: uniquely identify each rows/records in a database table.
    - **FOREIGN KEY:** is a key used to link two or more tables together. It is sometimes called a **referencing key.**
    - **CHECK constraint**: ensures that all values in a column satisfy certain conditions.
    - **DEFAULT constraint:** Sets a default value for a column if no value is specified
    - **Create index**: Used to create and retrieve data from the database very quickly
- Those Constraints can be specified when the **table is created** with the create table statement, or after the table is created with the **alter table** statement.
    - **Syntax :**
        - CREATE TABLE *table_name* (
          column1 datatype ***constraint***,
          column2 datatype ***constraint***,
          column3 datatype ***constraint***,
          ....);
    - **Example**
    - CREATE TABLE Orders (
      OrderID int NOT NULL PRIMARY KEY,
      OrderNumber int NOT NULL,
      PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
      );

SQL Statement

- **SQL Syntax:** SQL is followed by unique set of rules and guidelines called **Syntax.**
- All the SQL statements start with any of the keywords like ***SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW*** and all the statements end with a semicolon (;).
- **SQL is case insensitive which means SELECT** is equal to **select in SQL statements.**
- SQL language has two main components
    - Data Definition Language (DDL)
    - Data Manipulation Language(DML)

Data Definition Language (DDL)

- It provides commands for defining databases and relation schemas, deleting relations and modifying relation schemas within the data base.
- It has the following keywords
    - **CREATE**: is used to create a database and a database objects.

- **To create a database:**
    - **Syntax**: **CREATE database 'database Name'**; Where Database Name is a name of the new name.
    - Example: create a bank database with name xy.
        - CREATE database xy;

- **To create a relation or a table**
    - **Syntax**: **CREATE table 'table Name'**(A1D1, A2D2 …, An Dn, [Integrity constrain1]…[integrity constrain m]);

- Where Ai is the name of an attribute in the relation and Di is the data type of values in the domain of attribute Ai.
- There can be a number of different allowable SQL constraints, which is optional.

  Example: if we like to define a key constraint,
    - PRIMARY KEY(Ai1, Ai2…, Ai n) Where Ai's are attributes that are combined to form a key.

Ex: Create the table customer in the xy database?

Answer:

CREATE table customer (CustId char(10),

CustName varchar(20),

custCity varchar(20)

PRIMARY KEY (CustId));

Drop:

- Is used to delete existing database, table or constraints.
- Syntax: DROP [Database/Table/Constraints] name; where name is a name of the object going to be deleted.
- **Example**: Delete customer table from the xy database?
    - DROP table customer;

ALTER: is used to modify the definition of the existing relation schema.

Syntax: ALTER table name

ADD COLUMN 'column name' data type,…

|ADD CONSTRAINT 'constraint name' constraint type,…

Or

DROP COLUMN 'column name',… |DROP CONSTRAINT 'constraint name',…

Where table name is name of the table, column name and constraint name represent name of the column and name of the constraint that we want to modify respectively.

Example: Add column AccountNumber on customer table, and define a referential integrity constraint for it.

Answer:

ALTER customer

ADD COLUMN AccountNumber char(10),

ADD CONSTRAINT AccountNumber FOREIGN

KEY(customer) REFERENCES account

(AccountNumber);

Data Manipulation Language (DML)

- It is the means to manipulate the database.
- Typical manipulations include, INSERT, DELETE, UPDATE and SELECT of data.
- It includes commands to **insert tubles into**, **delete tubles from** and **modify tubles** in the database.
- It may be has two types
    - Data updating language and
    - Data retrieval language

A. **Data Updating Language:** SQL DML includes three clauses that can be used for **updating the content of relations.**

- ✦ The three operations that are required to data updating are **Insertion, Deletion and Updating**.

Insertion: it is a SQL clause used to add a number of tuples into a specified relation.

Syntax: INSERT INTO 'tablename'[(column names separated by comma)] **VALUES** (values separated by comma and with in a single quote(')(if not type of numbers))

Example: insert the following records in to customer table?

- (cust01, selam, Ambo)

INSERT INTO customer **VALUES** ('cust01', 'selam', 'Ambo');

- (cust02, abebe, addis abeba)

INSERT INTO customer(custId, custName, custCity) **VALUES**('cust02', 'abebe','AA');

Deletion: it is a clause used to delete a specified tuples from a relation that meets specified criteria.

Syntax: DELETE FROM table name WHERE (condition)

Example: DELETE FROM customer WHERE custId='cust01';

- ✦ **Update:** is a clause used to modify an existing by new value of a tuple that satisfies a given criteria.

- **Syntax:** UPDATE 'table name' SET column1 = value1, column2=value2 …

  WHERE (condition)

Where table name is a relation, and condition is a criteria based on which a deletion of tuples effected.

Example: UPDATE customer

SET custname ='belay'

WHERE custId='cust02' ;

B. **Data Retrieval Language :** in SQL statement , data retrieval has SELECT, FROM, and WHERE key languages.

Syntax: SELECT 'List of columns FROM 'table name WHERE [condition]

- **SELECT clause**: it is used to list the attributes desired in the result of a query.
  - **DISTINCT**: this clause is used to retrieve only one of the records that contain duplicate data in the selected fields .
- **All**: it is a clause that used to list all in the result relation.
- **FROM:** is a clause used to list the relation(table) to be scanned in the evaluation of the expression.

Example:

1. find all customer information from customer table?

   SELECT * FROM customer

2. Find name of all customer ?

   SELECT custName FROM customer

3. Find name of customers from the customer table and remove duplicate tuples from the result?

   SELECT DISTINCT custName FROM customer

- ***WHERE clause:*** it is an optional clause, which is used to specify conditions to select some records .
  - Example: SELECT *

    FROM customer

    WHERE custId="cust01" ;

- **LIKE**: it is an operator that goes with the WHERE clause and used to compare a string expression to a pattern.
    - It may be used with wild card (%) for representation for multiple characters in the expression.
- **Example:** Distinct list of all customer's name whose name starts with a letter A?

```
SELECT DISTINICT custName
FROM customer
WHERE custName LIKE 'A%' ;
```

Aggregate Functions (Group Functions)

- Aggregate functions return a single value, calculated from values in a column.
- Types of group functions
    - AVG, MIN, SUM, COUNT, MAX, etc.

AVG () - returns the average value of a numeric column.

Syntax: SELECT AVG (column_name) FROM <table_name >

MAX () - Returns the largest value

Syntax: *SELECT MAX (COLUMN NAME) FROM TABLE_NAME;*

MIN () - Returns the smallest value

Syntax: SELECT MIN (COLUMN_NAME) FROM TABLE_NAME;

SUM () - Returns the sum of the values.

Syntax: SELECT SUM (COLUMN_NAME) FROM Employee;

COUNT () - Returns the number of rows

Syntax: Select count (*) from <table name> –returns the whole rows

Using the DISTINCT Keyword

COUNT (DISTINCT expression) returns the number of distinct non-null values of *expression.*

To display the number of distinct department values in the Employee table:

Creating Groups of Data:

- **GROUP BY Clause**:-it collects the rows into groups based on column values in the grouping columns.
- Operates on the rows from the FROM clause as filtered by the WHERE clause.

Syntax: *Select <group by columns or aggregate function>*

FROM <table name> WHERE <condition>

GROUP BY <group by columns>

Having Clause:

- It is an optional clause that sets up conditions that select only some groups among those created using a group by clause.
- Expressions in having clause must be single valued and it can not be specified if group by clause is not used.

ORDER BY clause:

- It helps to appear the result query in sorted order.
- By default the order by clause lists in ascending order, but the order can be specified using DESC for descending order or ASC for ascending order.

BETWEEN & AND clause

Example- Retrieve all employees whose salary is between $30,000 and $40,000?

Answer: *SELECT* *FROM EMPLOYEE

WHERE SALARY **BETWEEN** 30000 **AND** 40000

Ambiguous Attribute Names and Aliasing

- In SQL the same name can be used for two (or more) attributes as long as the attributes are in *different relations.*
- If this is the case, and a query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name to prevent ambiguity.
    - This is done by ***prefixing*** *the* ***relation name*** *to the attribute name and separating the* two by a **period**.

Example-:

SELECT E.FName, E.LName, D.FNAME, D.Lname **FROM** Employee **AS E, Department AS D WHERE** E.Dnum=D.Dnum;

- In this case, we are allowed to declare alternative relation names E and D, called **aliases** or **tuple variables**, for the Employee and Department relation. An **alias** can follow the keyword **AS**

Nested Queries

- a query that is part of another query is called a sub-query(nested query).
  - It appears in the where clause .
  - A common use of sub-queries is to perform tests for set membership, make set comparison , and determine set cardinality.

Example :

Select s.NAME, s.Id From student as s Where s.Id IN(select Id from enroll where semester ="I")

Limitations of SQL

1. **Complex Interface**
   SQL has a difficult interface that makes few users uncomfortable while dealing with the database.
2. **Cost**
   Some versions are costly and hence, programmers cannot access it.
3. **Partial Control**
   Due to hidden business rules, complete control is not given to the database.

The differences b/n them

- Relational Algebra (RA) and Relational Calculus (RC) are formal languages for the database relational model while
- SQL is the practical language in the database relational model.
- In these formal languages a conceptual database model is expressed in mathematical terms and notations while in the practical language – SQL, the mathematical expressions of the functionality and transaction of the database operations are implemented physically.
- Each language, that RA, RC, and SQL have their own notations to express their notations.
- **RA Notations – Procedural Expression Language**
- **RC Notation–Non Procedural Expression Language**

Database Security and Integrity

- A database represents an essential corporate resource that should be properly secured using appropriate controls.
- Database security encompasses hardware, software, people and data
- Multi-user database system - DBMS must provide a database security and authorization subsystem to enforce limits on individual and group access rights and privileges.
- Database security and integrity is about protecting the database from being inconsistent and being disrupted. We can also call it database misuse.
- Database misuse could be Intentional or accidental, where accidental misuse is easier to cope with than intentional misuse.

Accidental inconsistency could occur due to:

- ✓ System crash during transaction processing
- ✓ Anomalies due to concurrent access
- ✓ Anomalies due to redundancy
- ✓ Logical errors

Likewise, even though there are various threats that could be categorized in this group, intentional misuse could be:

- ✓ Unauthorized reading of data
- ✓ Unauthorized modification of data or
- ✓ Unauthorized destruction of data

- Most systems implement good **Database Integrity** to protect the system from accidental misuse while there are many computer-based measures to protect the system from intentional misuse, which is termed as **Database Security** measures.
- Database security is considered in relation to the following situations:
    - ✓ Theft and fraud
    - ✓ Loss of confidentiality (secrecy)
    - ✓ Loss of privacy
    - ✓ Loss of integrity
    - ✓ Loss of availability

Security Issues and general considerations

- **Legal**, **ethical** and **social** issues regarding the right to access information
- Physical control
- **Policy** issues regarding privacy of individual level at enterprise and national level
- **Operational** consideration on the techniques used (password, etc)
- **System** level security including operating system and hardware control
- Security levels and security policies in enterprise level
- **Database security** - the mechanisms that protect the database against intentional or accidental ***threats.*** And Database security encompasses hardware, software, people and data
- **Threat –** any situation or event, whether intentional or accidental, that may adversely affect a system and consequently the organization
- A threat may be caused by a situation or event involving a person, action, or circumstance that is likely to bring harm to an organization
- The harm to an organization may be ***tangible*** or ***intangible***

Tangible – loss of hardware, software, or data
Intangible – *loss of credibility or client confidence*

Examples of threats:

- Using another persons' means of access
- Unauthorized amendment/modification or copying of data
- Program alteration
- Inadequate policies and procedures that allow a mix of confidential and normal out put
- Wire-tapping
- Illegal entry by hacker
- Blackmail
- Creating 'trapdoor' into system
- Theft of data, programs, and equipment
- Failure of security mechanisms, giving greater access than normal
- Staff shortages or strikes
- Inadequate staff training
- Viewing and disclosing unauthorized data
- Electronic interference and radiation
- Data corruption owing to power loss or surge
- Fire (electrical fault, lightning strike, arson), flood, bomb
- Physical damage to equipment
- Breaking cables or disconnection of cables
- Introduction of viruses

- An organization needs to identify the types of threat it may be subjected to and initiate appropriate plans and ***countermeasures***, bearing in mind the costs of implementing them

Countermeasures: Computer based controls

- The types of countermeasure to threats on computer systems range from **physical controls to administrative procedures**
- Despite the range of computer-based controls that are available, it is worth noting that, generally, ***the security of a DBMS is only as good as that of the operating system***, owing to their close association
- The following are computer-based security controls for a multi-user environment:

Authorization

- The granting of a right or privilege that enables a subject to have legitimate access to a system or a system's object
- Authorization controls can be built into the software, and govern not only what system or object a specified user can access, but also what the user may do with it
- Authorization controls are sometimes referred to as ***access controls***
- The process of authorization involves authentication of ***subjects*** (i.e. a user or program) requesting access to ***objects*** (i.e. a database table, view, procedure, trigger, or any other object that can be created within the system)

Views

- A view is the dynamic result of one or more relational operations operation on the base relations to produce another relation
- A view is a virtual relation that does not actually exist in the database, but is produced upon request by a particular user
- The view mechanism provides a powerful and flexible security mechanism by hiding parts of the database from certain users
- Using a view is more restrictive than simply having certain privileges granted to a user on the base relation(s)

Backup and recovery

- Backup is the process of periodically taking a copy of the database and log file (and possibly programs) on to offline storage media
- A DBMS should provide backup facilities to assist with the recovery of a database following failure
- Database recovery is the process of restoring the database to a correct state in the event of a failure
- Journaling is the process of keeping and maintaining a log file (or journal) of all changes made to the database to enable recovery to be undertaken effectively in the event of a failure
- The advantage of journaling is that, in the event of a failure, the database can be recovered to its last known consistent state using a backup copy of the database and the information contained in the log file
- If no journaling is enabled on a failed system, the only means of recovery is to restore the database using the latest backup version of the database
- However, without a log file, any changes made after the last backup to the database will be lost

Integrity

- Integrity constraints contribute to maintaining a secure database system by preventing data from becoming invalid and hence giving misleading or incorrect results
- Domain Integrity
- Entity integrity
- Referential integrity
- Key constraints

Encryption

- The encoding of the data by a special algorithm that renders the data unreadable by any program without the decryption key
- If a database system holds particularly sensitive data, it may be deemed necessary to encode it as a precaution against possible

- external threats or attempts to access it
- The DBMS can access data after decoding it, although there is a degradation in performance because of the time taken to decode it
- Encryption also protects data transmitted over communication lines
- To transmit data securely over insecure networks requires the use of a ***Cryptosystem***, which includes:

RAID technology **(Redundant Array of Independent Disks)**

- The hardware that the DBMS is running on must be fault-tolerant, meaning that the DBMS should continue to operate even if one of the hardware components fails. This suggests having redundant components that can be seamlessly integrated into the working system whenever there is one or more component failures. The main hardware components that should be fault-tolerant include disk drives, disk controllers, CPU, power supplies, and cooling fans. Disk drives are the most vulnerable components with the shortest times between failures of any of the hardware components.
- RAID works on having a large disk array comprising an arrangement of several independent disks that are organized to improve reliability and at same time increase performance
- Performance is increased through ***data striping***
- ***Data striping*** – the data is segmented into equal size partitions (the striping unit) which are transparently distributed across multiple disks.

Levels of Security Measures

Security measures can be implemented at several levels and for different components of the system. These levels are:

1. **Physical Level:** concerned with securing the site containing the computer system should be physically secured. The backup systems should also be physically protected from access except for authorized users.

2. **Human Level:** concerned with authorization of database users for access the content at different levels and privileges.

3. **Operating System:** concerned with the weakness and strength of the operating system security on data files. Weakness may serve as a means of unauthorized access to the database. This also includes protection of data in primary and secondary memory from unauthorized access.

4. **Database System:** concerned with data access limit enforced by the database system. Access limit like password, isolated transaction and etc.

Even though we can have different levels of security and authorization on data objects and users, ***who access which data is a policy matter rather than technical.***

These policies

- should be known by the system: should be encoded in the system
- should be remembered: should be saved somewhere (the catalogue)

Any database access request will have the following three major components

1. **Requested Operation:** what kind of operation is requested by a specific query?
2. **Requested Object:** on which resource or data of the database is the operation sought to be applied?
3. **Requesting User:** who is the user requesting the operation on the specified object?

The database should be able to check for all the three components before processing any request. The checking is performed by the security subsystem of the DBMS.

Authentication

- All users of the database will have different access levels and permission for different data objects, and authentication is the process of checking whether the user is the one with the privilege for the access level.
- Is the process of checking the users are who they say they are.
- Each user is given a unique identifier, which is used by the operating system to determine who they are

- Thus, the system will check whether the user with a specific username and password is trying to use the resource.
- Associated with each identifier is a password, chosen by the user and known to the operation system, which must be supplied to enable the operating system to authenticate who the user claims to be

Forms of user authorization

There are different forms of user authorization on the resource of the database. These forms are privileges on what operations are allowed on a specific data object.

User authorization on the data/extension

1. **Read Authorization**: the user with this privilege is allowed only to read the content of the data object.

2. **Insert Authorization**: the user with this privilege is allowed only to insert new records or items to the data object.

3. **Update Authorization**: users with this privilege are allowed to modify content of attributes but are not authorized to delete the records.

4. **Delete Authorization**: users with this privilege are only allowed to delete a record and not anything else.

- Different users, depending on the power of the user, can have one or the combination of the above forms of authorization on different data objects.

Role of DBA in Database Security

The database administrator is responsible to make the database to be as secure as possible. For this the DBA should have the most powerful privilege than every other user. The DBA provides capability for database users while accessing the content of the database.

The major responsibilities of DBA in relation to authorization of users are:

1. ***Account Creation:*** involves creating different accounts for different **USERS** as well as **USER GROUPS.**

2. ***Security Level Assignment***: involves in assigning different users at different categories of access levels.

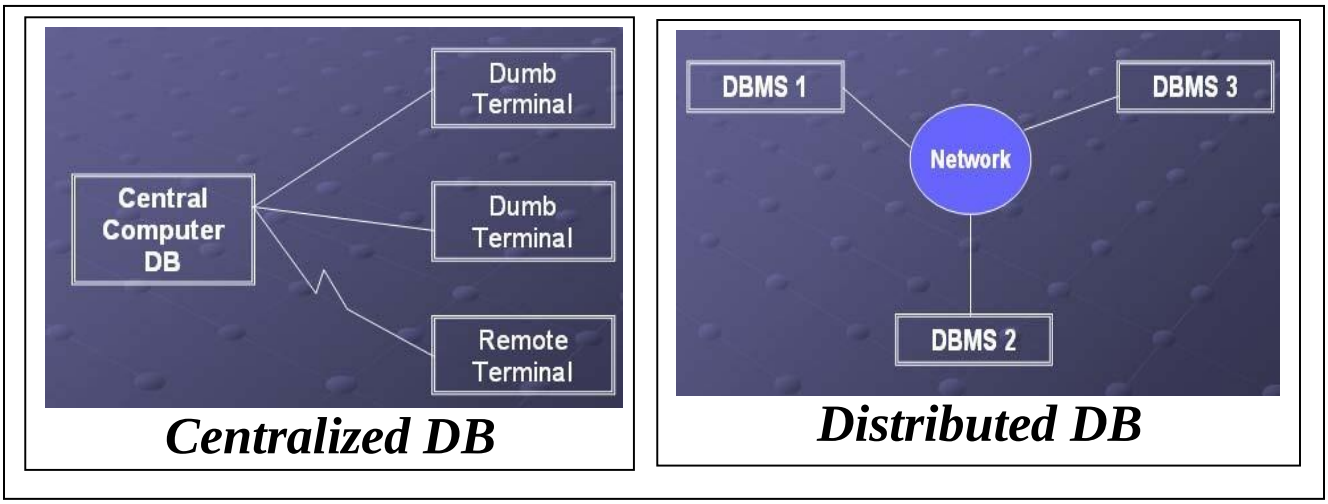
3. ***Privilege Grant:*** involves giving different levels of privileges for different users and user groups.

4. ***Privilege Revocation:*** involves denying or canceling previously granted privileges for users due to various reasons.

5. ***Account Deletion:*** involves in deleting an existing account of users or user groups. Is similar with denying all privileges of users on the database.

Distributed Database Systems

- Database development facilitates the integration of data available in an organization and enforces security on data access. But it is not always the case that organizational data reside in one site. This demand databases at different sites to be integrated and synchronized with all the facilities of database approach. This leads to Distributed Database Systems.
- *Distributed Database is not a centralized database*.



Data Distribution Strategies

- *Distributed DB stores logically related data at several independent sites connected via network*
- *Data allocation is the process of deciding where to allocate/store particular data*

There 3 data allocation strategies:

1. ***Centralized:*** the entire DB is located at a single site

2. ***Partitioned:*** the DB is split into several disjoint parts (called partitions, segments or fragments) and stored at several sites

3. ***Replicated:*** copies of one or more partitions are stored at several sites

- In a distributed database system, the database is stored on several computers. The computers in a distributed system communicate with each other through various communication media, such as high speed buses or telephone line.
- A distributed database system consists of a collection of sites, each of which maintains a local database system and also participates in global transaction where different databases are integrated together.
- ***Local Transaction***: transactions that access data only in that single site
- ***Global Transaction***: transactions that access data in several sites.

Issues in DDBMS

How is data stored in DDBMS

There are several ways of storing a single relation in distributed database systems.

Replication:

- System maintains multiple copies of similar data (identical data)
- Stored in different sites, for faster retrieval and fault tolerance.
- Duplicate copies of the tables can be kept on each system (replicated). With this option, updates to the tables can become involved (of course the copies of the tables can be read-only).
- Advantage: Availability, Increased parallelism (if only reading)
- Disadvantage: increased overhead of update

Fragmentation:

- Relation is partitioned into several fragments stored in distinct sites
- The partitioning could be ***vertical***, ***horizontal*** or ***both***.

Horizontal Fragmentation

- Systems can share the responsibility of storing information from a single table with individual systems storing groups of rows
- Performed by the ***Selection Operation***
- The whole content of the relation is reconstructed using the ***UNION*** operation

Vertical Fragmentation

- Systems can share the responsibility of storing particular attributes of a table.
- ***Needs attribute with tuple number***
- Performed by the ***Projection Operation***
- The whole content of the relation is reconstructed using the ***Natural JOIN*** operation using the attribute with ***Tuple number***

Homogeneous and Heterogeneous Distributed Databases

In a homogeneous distributed database

- All sites have identical software (DBMS)
- Are aware of each other and agree to cooperate in processing user requests.
- Each site surrenders part of its autonomy in terms of right to change schemas or software
- Appears to user as a single system

In a heterogeneous distributed database

- Different sites may use different schemas and software (DBMS)
  - Difference in schema is a major problem for query processing
  - Difference in software is a major problem for transaction processing
- Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing

Why DDBMS/Advantages

1. Many existing systems

- Maybe you have no choice.
- Possibly there are many different existing systems, with possible different kinds of systems (Oracle, Informix, others) that need to be used together.

2. Data sharing and distributed control:

- User at one site may be able access data that is available at another site.
- Each site can retain some degree of control over local data
- We will have local as well as global database administrator

3. Reliability and availability of data

- If one site fails the rest can continue operation as long as transaction does not demand data from the failed system and the data is not replicated in other sites

4. Speedup of query processing

- If a query involves data from several sites, it may be possible to split the query into sub-queries that can be executed at several sites which is parallel processing
- Query can be sent to least heavily loaded sites

5. Expansion

- In a distributed environment you can easily expand by adding more machines to the network.

Disadvantages of DDBMS

1. Software Development Cost

- Is difficult to install, thus is costly

2. Greater Potential for Bugs

- Parallel processing may endanger correctness of algorithms

3. Increased Processing Overhead

- Exchange of message between sites
- Due to communication jargons

4. Communication problems

5. Increased Complexity and Data Inconsistency Problems

- Since clients can read and modify closely related data stored in different database instances concurrently.

PLSQL (Trigger, Store Procedure and Function)

- PL/SQL stands for Procedural Language extension of SQL
- It is a procedural language designed specifically to embrace SQL statements within its syntax.
- PL/SQL program units are compiled by the Oracle Database server and stored inside the database. And at run-time, both PL/SQL and SQL run within the same

server process, bringing optimal efficiency.

- PL/SQL automatically inherits the robustness, security, and portability of the Oracle Database.

Triggers

- Triggers are named database objects that are implicitly fired when a triggering event occurs.
- Triggers provide a way of executing PL/SQL code on the occurrence of specific database events.
- A database trigger is a stored program which is automatically fired or executed when some events occur.
- A trigger can execute in response to any of the following events:

1. A database manipulation (DML) statement like DELETE, INSERT or UPDATE.
2. A database definition (DDL) statement like CREATE, ALTER or DROP.
3. A database operation like SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN.

Note: A trigger can be defined on the table, view, schema or database with which the event is associated.

Stored Procedure?

- A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
- You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.
- Stored procedures provide a powerful way to code application logic that can be stored on the server.
- The ***CREATE PROCEDURE*** command is used to create a stored procedure.

Syntax: CREATE PROCEDURE *procedure_name* AS

sql_statement

GO;

EXEC *procedure_name*; ***to Execute a Stored Procedure***

Example:

```
CREATE PROCEDURE SelectAllCustomers AS
SELECT * FROM Customers
GO;
EXEC SelectAllCustomers;
```

Non-Relational Database (NoSQL Database)

- NoSQL can be defined as a database which is employed for managing the massive collection of unstructured data and when your data is not piled up in a tabular format or relations like that of relational databases.
- The term NoSQL came from the word non-SQL or nonrelational.
- It does not follow the rules of Relational Database Management Systems (RDBMS), and hence do not use traditional SQL statements to query your data.
- Some famous examples are MongoDB, Neo4J, HyperGraphDB, etc.

Types of NoSQL database

NoSQL databases usually fall under any one of these four categories:

1. **Key-value stores:** is the most straightforward type where every item of your database gets stored in the form of an attribute name (i.e., "key") along with the value.
2. **Wide-column stores:** accumulate data collectively as a column rather than rows which are optimized for querying big datasets.
3. **Document databases:** couple every key with a composite data structure termed as a document. These documents hold a lot of different key-value pairs, as well as key-array pairs or sometimes nested documents.
4. **Graph databases:** are used for storing information about networks, like social connections.

Benefits of NoSQL Type Database

- It allows developers to create large volumes of structured, semi-structured as well as unstructured data for making the application diverse and not restricting its use because of the type of data being used within the application.
- It also allows agile development; rapid iteration along with frequent code pushes, which makes it more popular.
- It can be used with object-oriented programming (OOP), which makes it easy to use with flexibility.
- Data can be stored more efficiently, making it less expensive, providing massive architecture.

Difference between SQL and NoSQL

Here is the list of comparisons between both the DBMS:

- SQL databases are mainly coming under Relational Databases (RDBMS) whereas databases mostly come under non-relational or distributed database.
- SQL databases are table-oriented databases, whereas NoSQL databases document-oriented have key-value pairs or wide-column stores or graph databases.
- SQL databases have a predefined or static schema that is rigid, whereas NoSQL databases have dynamic or flexible schema to handle unstructured data.
- SQL is used to store structured data, whereas NoSQL is used to store structured as well as unstructured data.
- SQL databases can be considered as vertically scalable, but NoSQL databases are considered horizontally scalable.
- Scaling of SQL databases is done by mounting the horse-power of your hardware. But, scaling of NoSQL databases is calculated by mounting the databases servers for reducing the load.

- Examples of SQL databases: MySql, Sqlite, Oracle, Postgres SQL, and MS-SQL. Examples of NoSQL databases: BigTable, MongoDB, Redis, Cassandra, RavenDb, Hbase, CouchDB and Neo4j
- When your queries are complex SQL databases are a good fit for the intensive environment, and NoSQL databases are not an excellent fit for complex queries. Queries of NoSQL are not that powerful as compared to SQL query language.
- SQL databases need vertical scalability, i.e., excess of load can be managed by increasing the CPU, SSD, RAM, GPU, etc., on your server. In the case of NoSQL databases, they horizontally scalable, i.e., the addition of more servers will ease out the load management thing to handle.