Lecture Notes
ON
Fundamentals of Database Systems

CoSc2071

Prerequisites: Introduction to Computer Science (CoSc1011)

BSc. II Semester, Evening (CS Second year)

AMBO UNIVERSITY WOLISO CAMPUS, TECHNOLOGY AND INFORMATICS SCHOOL

Department of Computer Science

Prepared by: Abraham A.

Email: abrahamojip210@gmail.com
Phone No. : +251 910272054 or 900272244

2019/20 G.C (2012 E.C)
Woliso, Ethiopia

CHAPTER ONE

Introduction to Database Systems

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-Wide-Web, 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
- 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 follows 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 has 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 is 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

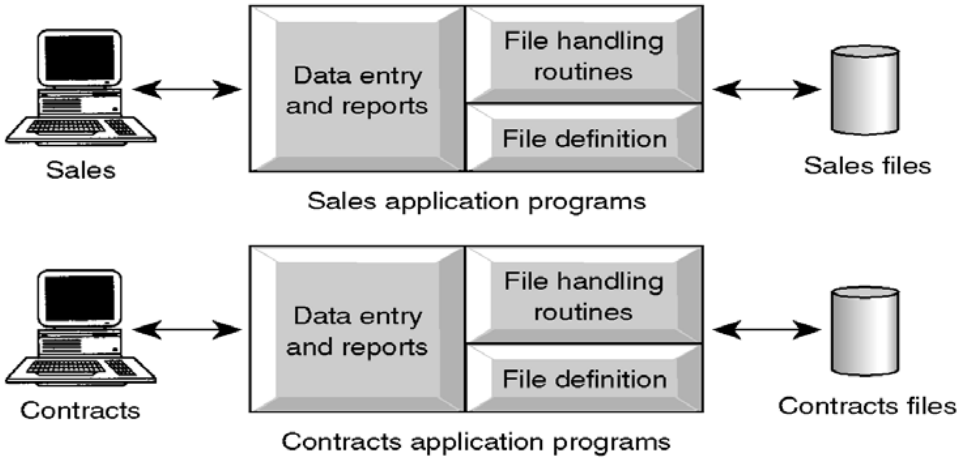
An alternative approach of data handling is a computerized way of dealing with the information. The computerized approach could also be either decentralized or centralized base on where the data resides in the system.

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 or 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 cannot 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.

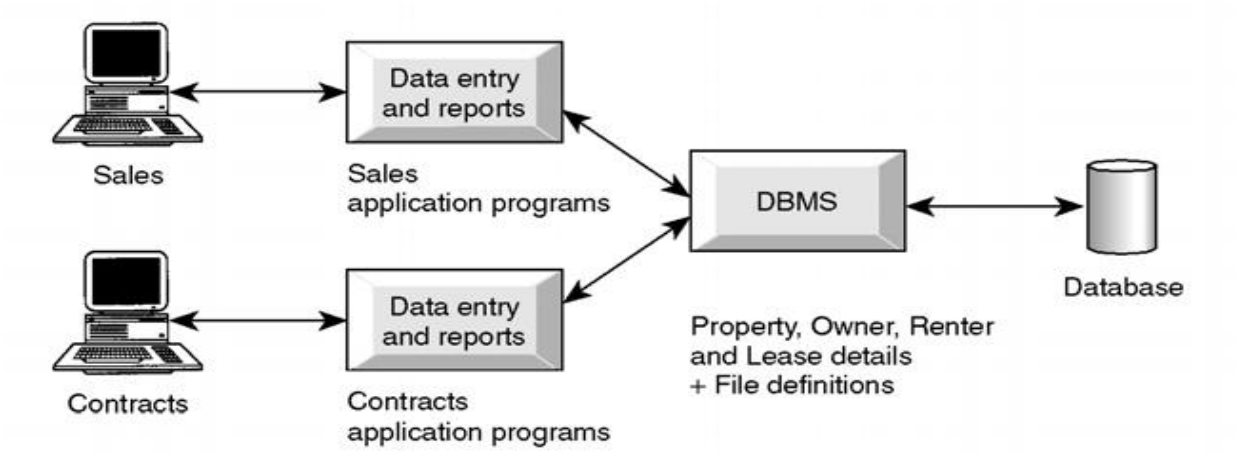
Thus in Database Approach:

- 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 the 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 e.g. SQL
  - Also called "query language"
- **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

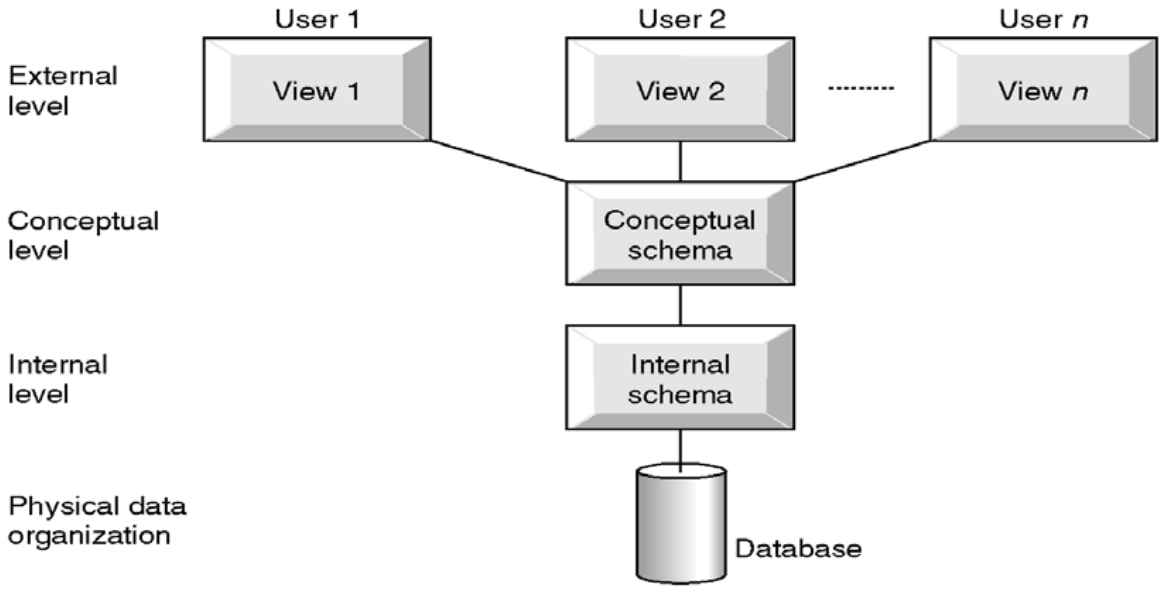
- **DBMS** designers and implementers: who design and implement different DBMS software.
- **Tool Developers**: experts who develop software packages that facilitates database system designing and use. Prototype, simulation, code generator developers could be an example. Independent software vendors 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.

CHAPTER TWO
Database System Concepts and Architecture

History of Database Systems

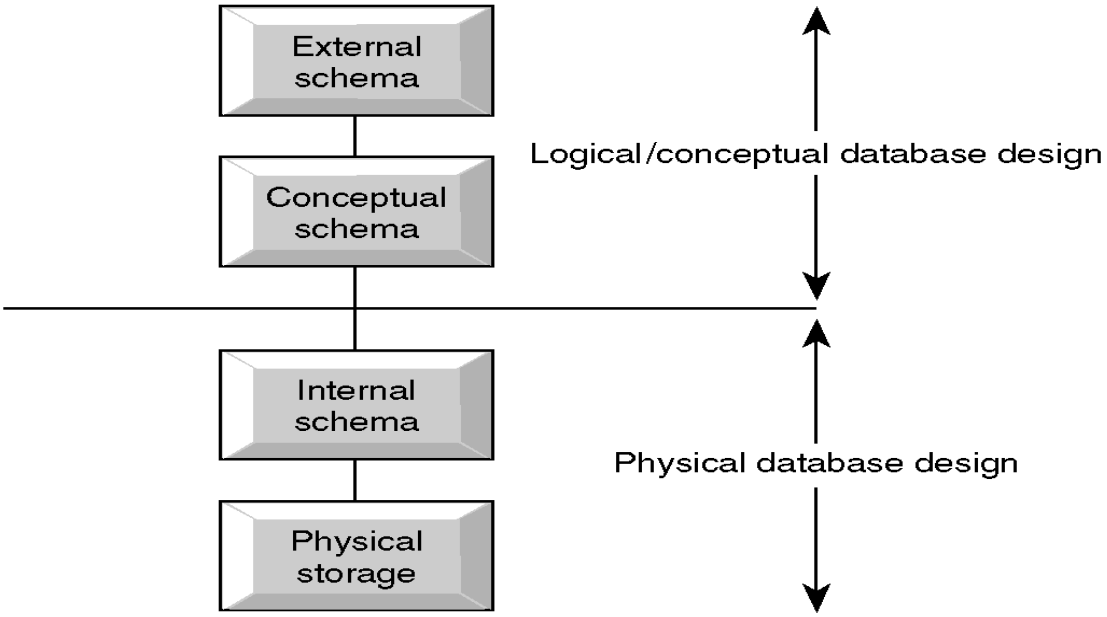
The purpose and origin of the Three-Level database architecture

- 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

A NSI-SPARC Architecture and Database Design Phases



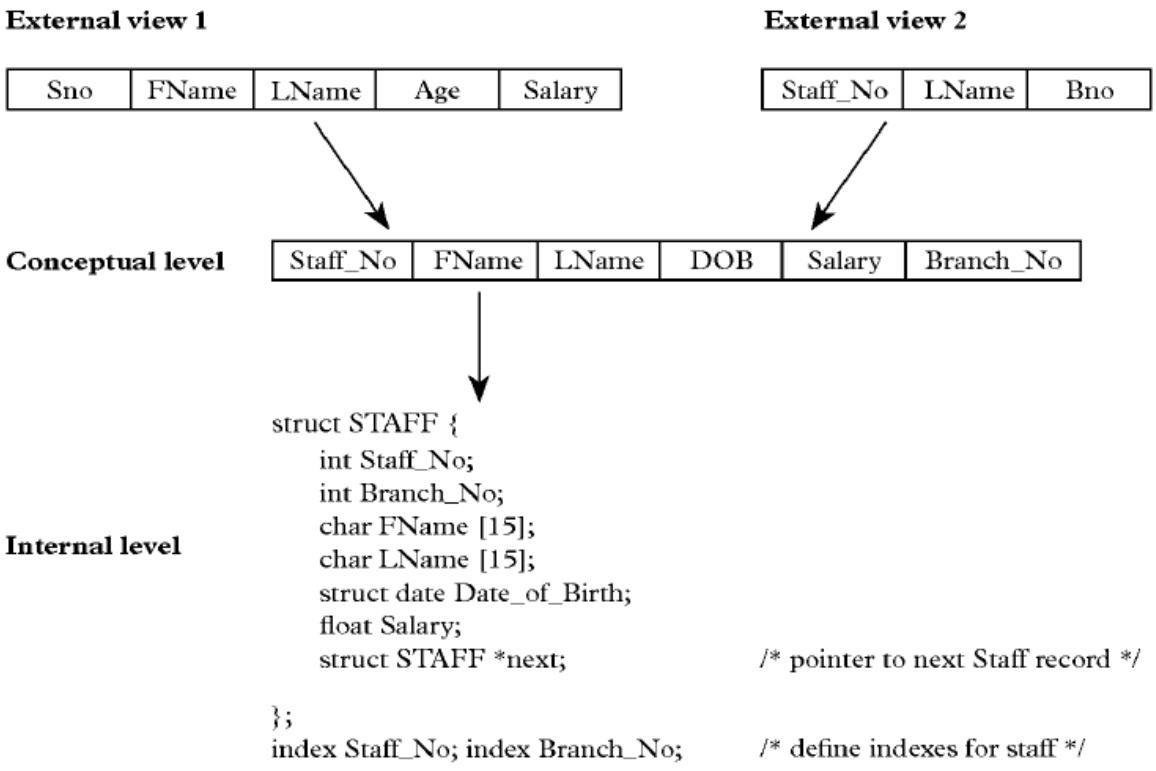
The contents of the external, conceptual and internal levels

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

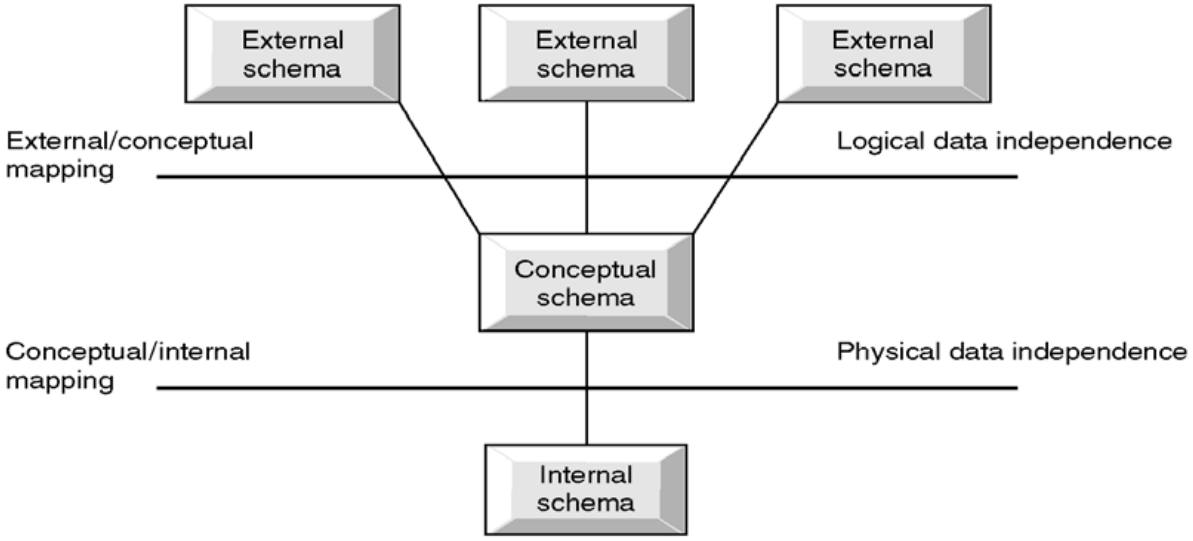
1. Logical Data Independence

- 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.

2. *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

Fourth Generation Language (4GL)

- Query Languages
- Forms Generators
- Report Generators
- Graphics Generators
- Application Generators

A Classification of data models

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.

- Relational Data Model
- Network Data Model
- Hierarchical Data Model

We have three major types of data models

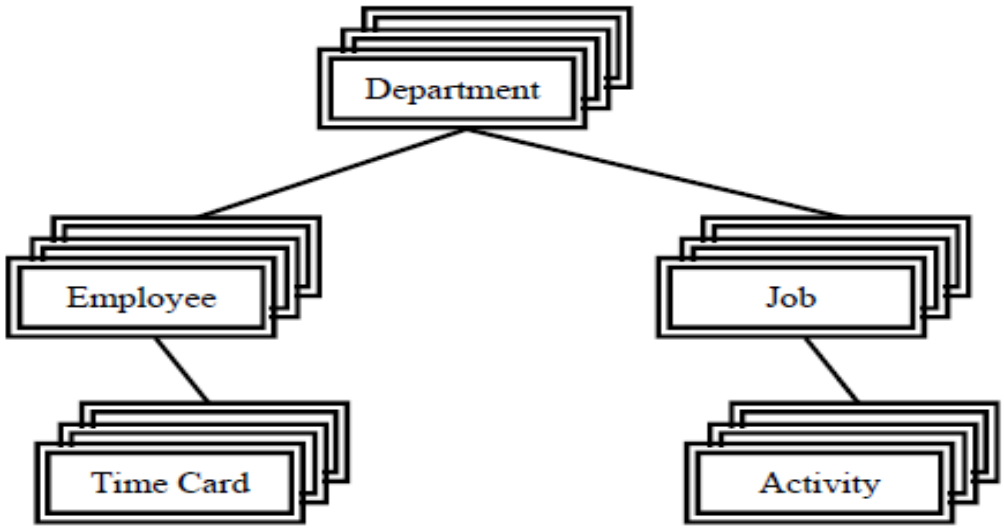
1. *Hierarchical Model*
2. *Network Model*
3. *Relational Data Model*

- **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.

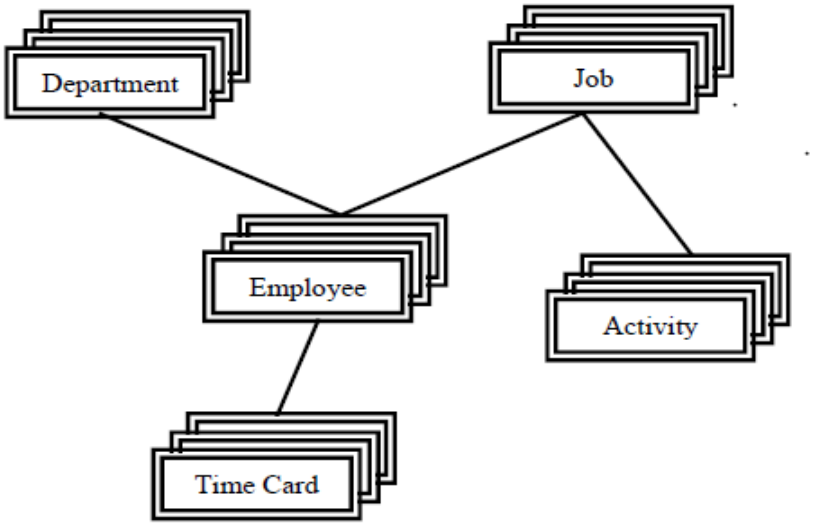
DISADVANTAGES of Hierarchical Data Model:

- 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 some how.
- 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

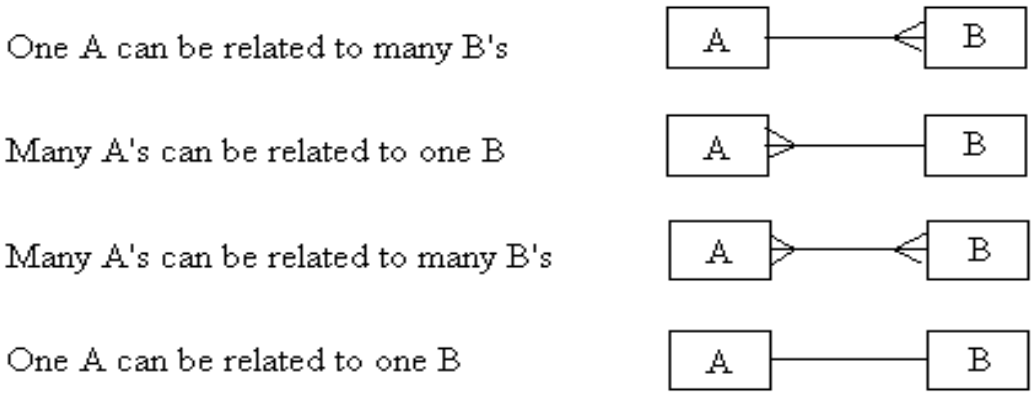
- NULL applies to attributes which are not applicable or which do not have values.
- You may enter the value NA (meaning not applicable)
- Value of a key attribute cannot be null.

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 entity sets*
    - ONE-TO-ONE, e.g. Building - Location,
    - ONE-TO-MANY, e.g. hospital - patient,
    - MANY-TO-ONE, e.g. Employee - Department
    - MANY-TO-MANY, e.g. Author - Book.
- 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.



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

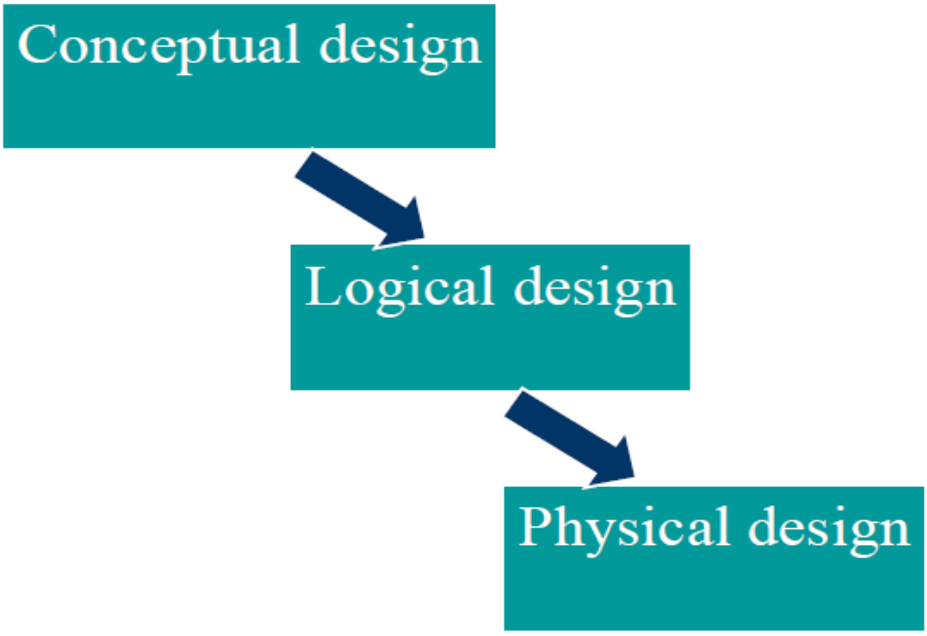
- **Database state (snapshot or extension)**: is the collection of data in the database at a particular point of time (snap-shot).
    - Refers to the actual data in the database at a specific time
    - State of database is changed any time we add or delete a record
    - ***Valid state***: the state 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

CHAPTER THREE
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.

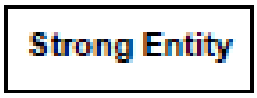
Developing an E-R Diagram

- 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 E-R 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

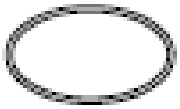
- Entity is represented by a RECTANGLE containing the name of the entity.



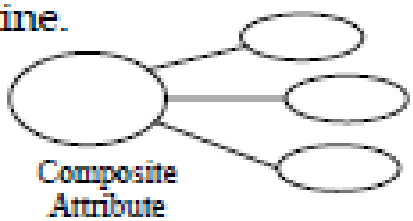
- Connected entities are called relationship participants

- Attributes are represented by OVALS and are connected to the entity by a line.

Attribute



Multi-valued Attribute



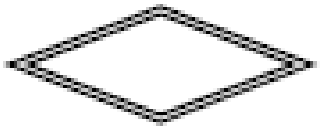
Composite Attribute

- A derived attribute is indicated by a DOTTED LINE. (……..)

- PRIMARY KEYS are underlined.

- Relationships are represented by DIAMOND shaped symbols

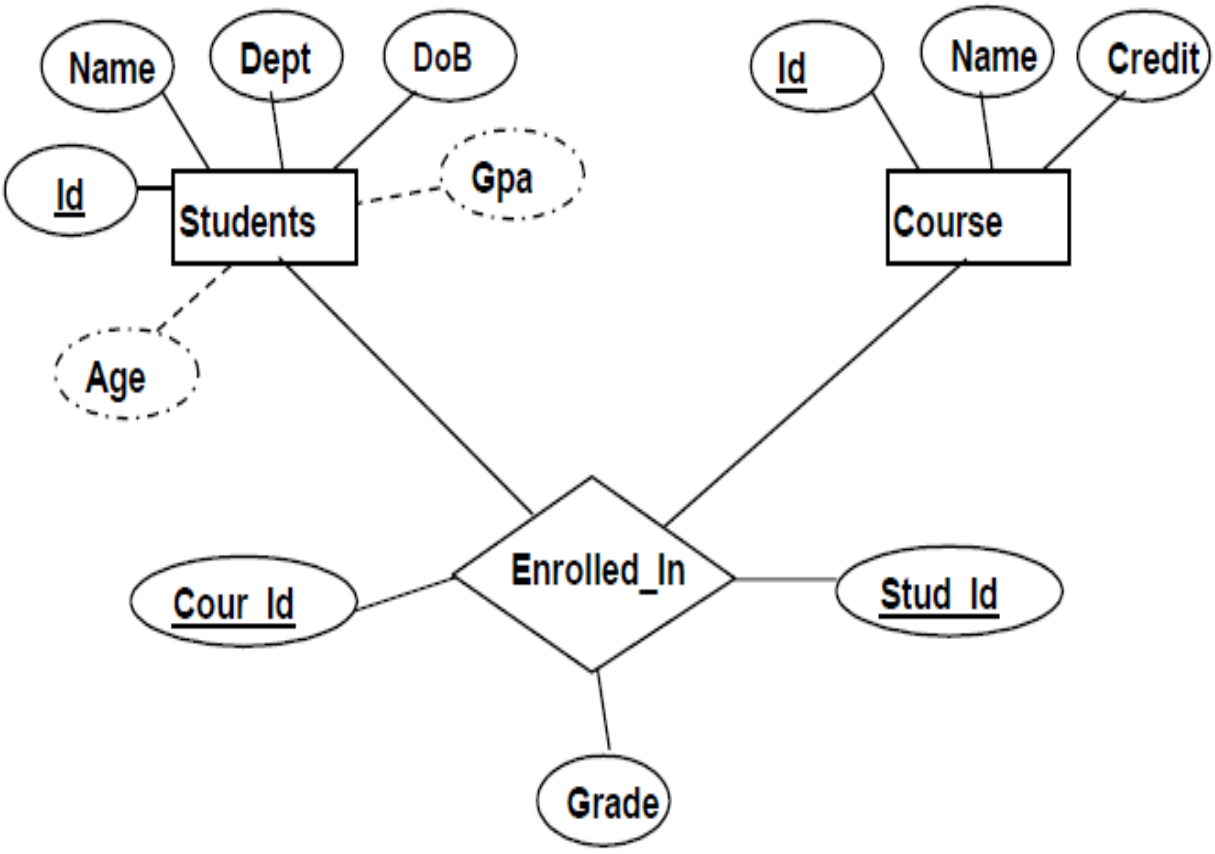
Relationship



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
- Participating entity in a relationship is either optional or mandatory.

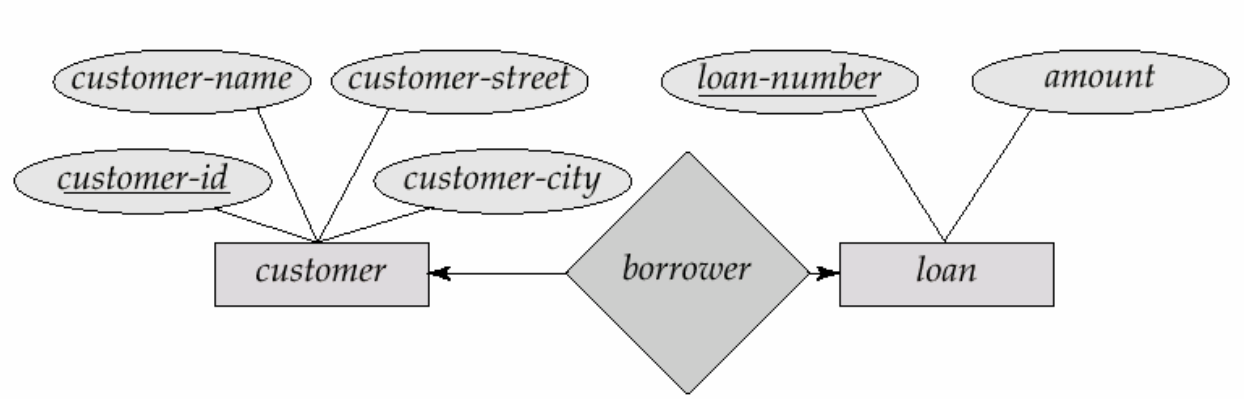
Structural Constraints on Relationship

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

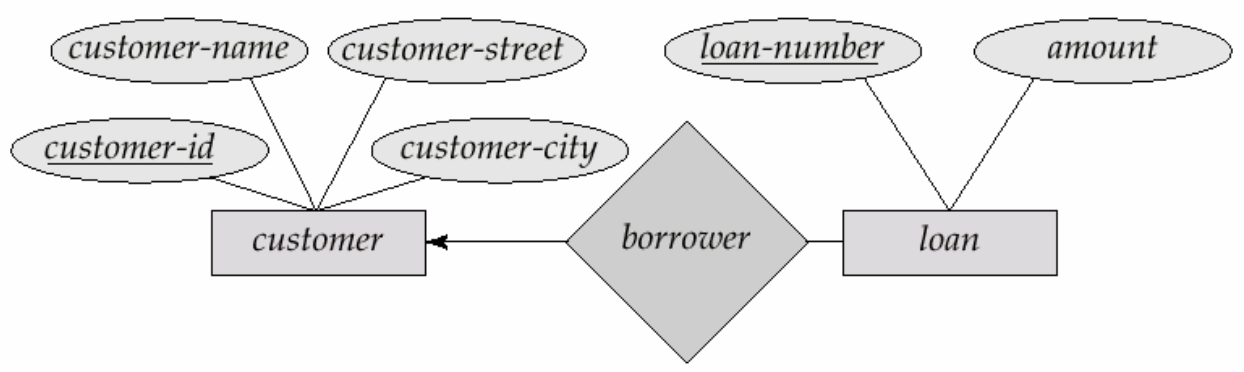
The multiplicity of the relationship is:

One branch can only have one manager

One employee could manage either one or no branches

One-To-Many Relationships

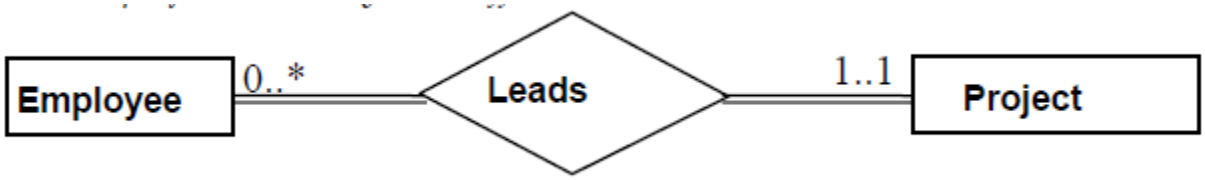
- In the one-to-many relationship a loan is associated with at most one customer via *borrower*, a customer is associated with several (including 0) loans via *borrower*



E.g.: Relationship ***Leads*** between ***STAFF*** and ***PROJECT***

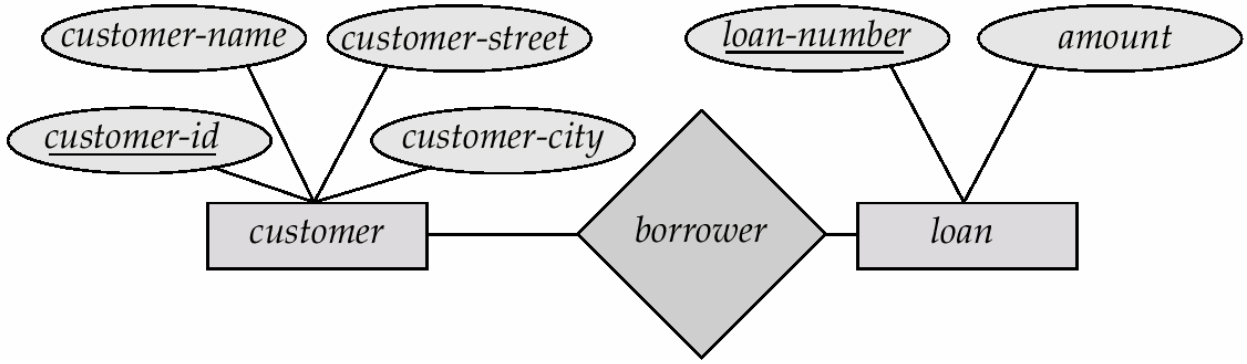
The multiplicity of the relationship

- *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

- **Total participation** (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

- **Partial participation**: 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;

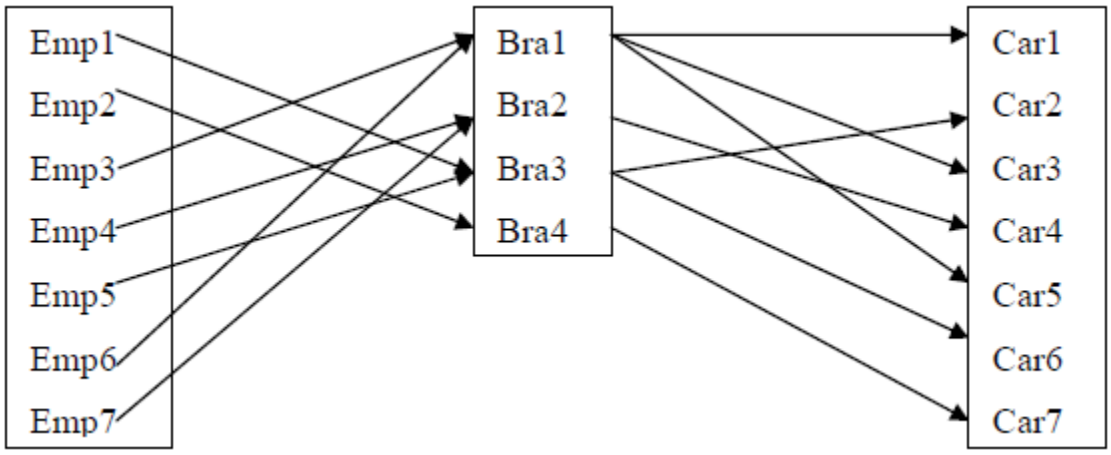
1. **Fan trap:**

   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 singe entity and all the semantics of the relationship is preserved.

Example:

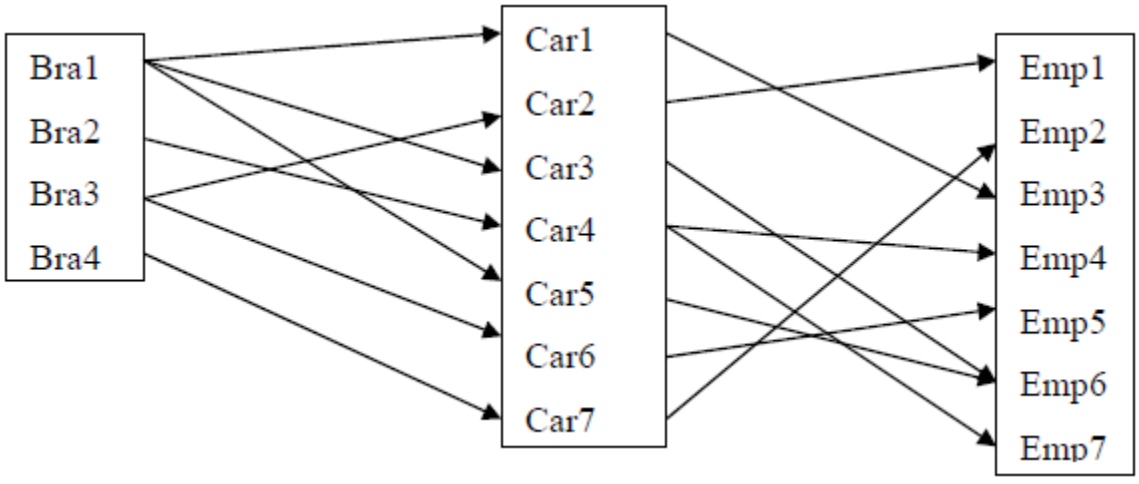
Semantics description of the problem;



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.

Semantic 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.

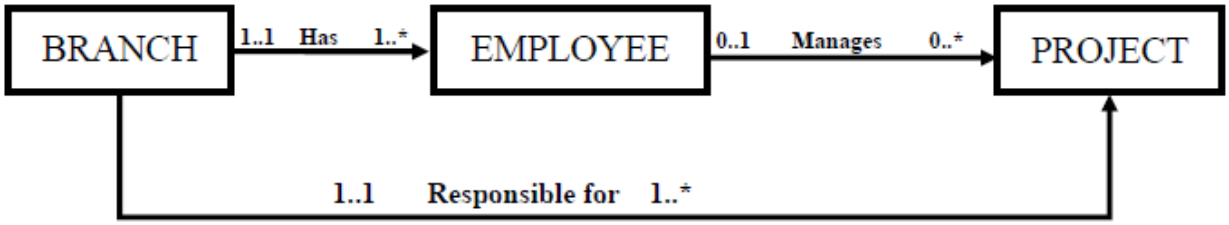
Example:

If we have a set of projects that are not active currently then we can not 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 relation ship between the extreme entities (BRANCH and PROJECT)



Enhanced E-R (EER) Models)

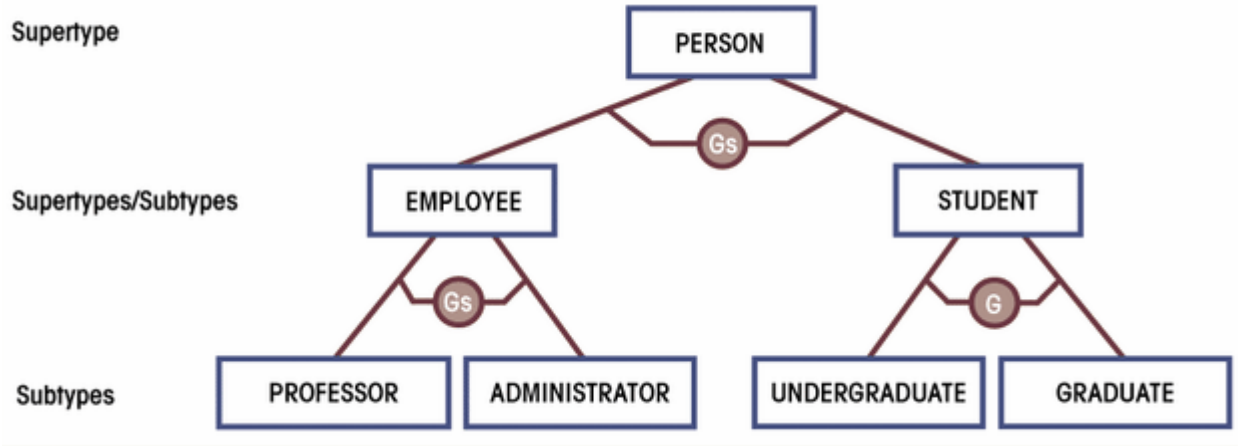
- Object-oriented extensions to E-R model
- EER is important when we have a relationship between two entities and the participation is partial between entity occurrences. In such cases EER is used to reduce the complexity in participation and relationship complexity.
- ER diagrams consider entity types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

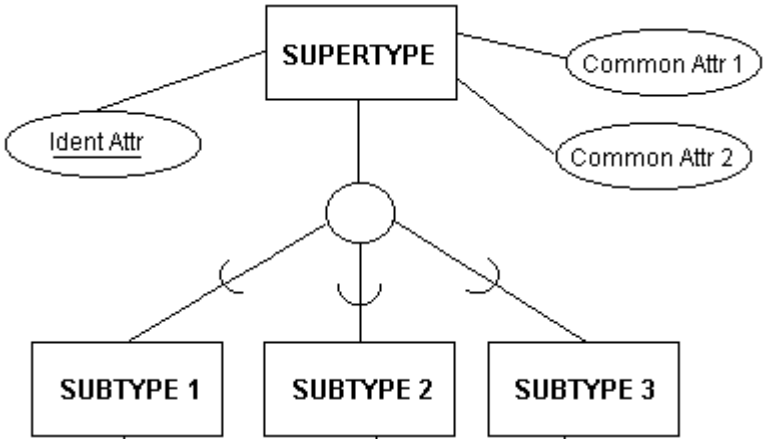
- An entity type whose tuples have attributes that distinguish its members from tuples of the generalized or Superclass entities.
- When one generalized Superclass has various subgroups with distinguishing features and these subgroups are represented by specialized form, the groups are called subclasses.
- 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.

❖ 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

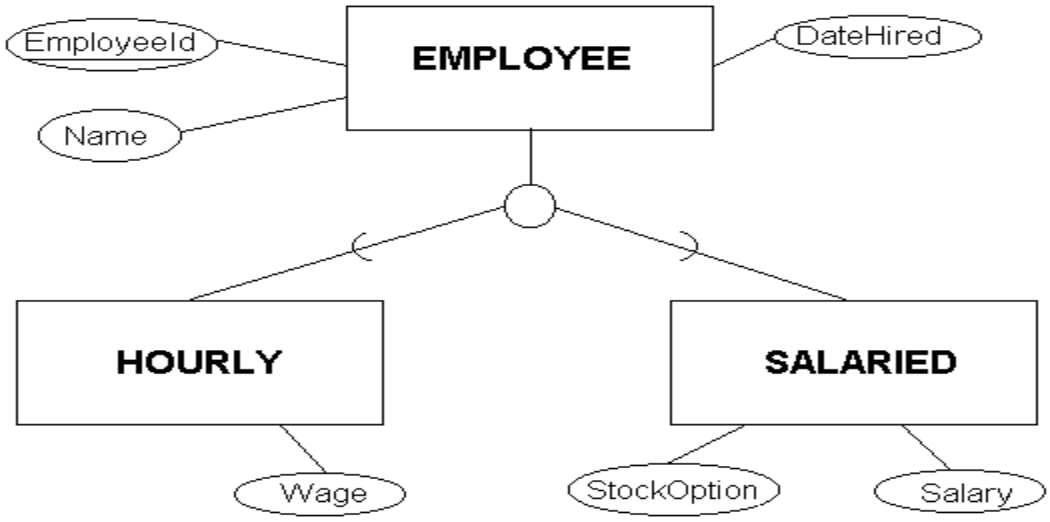
- The relationship between a superclass and any of its subclasses is called a superclass/subclass or class/subclass relationship
- An instance can not only be a member of a subclass. i.e. Every instance of a subclass is also an instance in the Superclass.
- A member of a subclass is represented as a 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 super-class.
- 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.

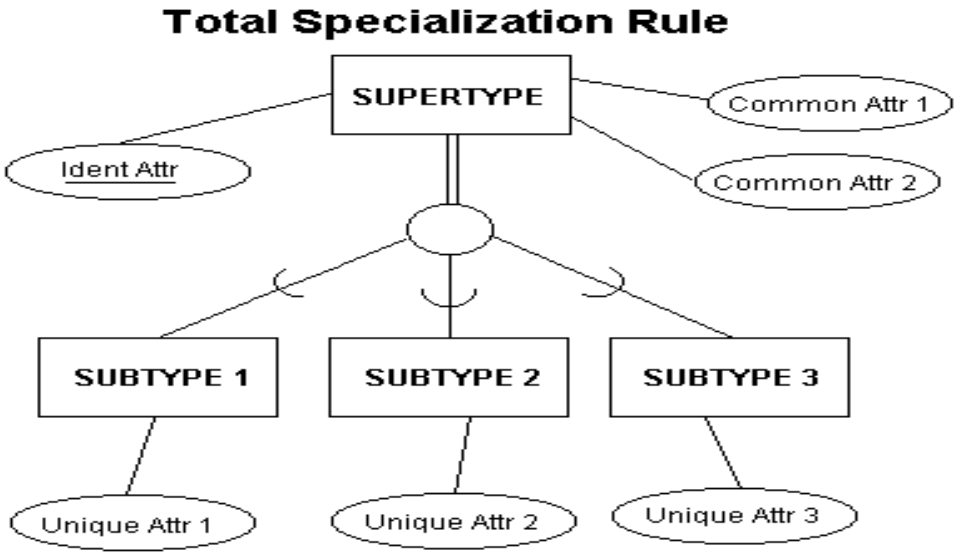
Constraints on specialization and generalization

❖ 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

CHAPTER FOUR
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 reqre them,
(4) ui(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, whatever 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 refered 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 appears 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: $A \rightarrow B$ 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 achiving 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

EmpID	**EmpName**

PROJECT

ProjNo	ProjName	ProjLoc	ProjFund	ProjMangID

EMP_PROJ

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**

This 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:
  - o definition of Primary key
  - o definition of Foreign key
  - o definition of Alternate key
  - o definition of Domains
  - o Referential integrity constraints
  - o 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
2.2. Choose file organization
2.3. Choose indexes
2.4. Estimate disk space and system requirement

3. Design user view
4. Design security mechanisms
5. Consider controlled redundancy
6. Monitor and tune the operational system

CHAPTER FIVE
STRUCTURED QUERY LANGUAGE (SQL)

What is SQL?

Structured Query Language, commonly abbreviated to SQL and pronounced as "sequel", is not a conventional computer programming language in the normal sense of the phrase. It allows users to access data in relational database management systems. SQL is about data and results, each SQL statement returns a result, whether that result be a query, an update to a record or the creation of a database table. SQL is most often used to address a relational database, which is what some people refer to as a SQL database.So in brief we can describe SQL as follows:

- SQL stands for Structured Query Language
- SQL allows you to access a database
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert new records in a database
- SQL can delete records from a database
- SQL can update records in a database
- SQL is easy to learn

Creating a Database

Many database systems have graphical interfaces which allow developers (and users) to create, modify and otherwise interact with the underlying database management system (DBMS). However, for the purposes of this chapter all interactions with the DBMS will be via SQL commands rather than via menus.

SQL Commands

There are three groups of commands in SQL:

1. Data Definition
2. Data Manipulation and
3. Transaction Control

Characteristics of SQL Commands

Here you can see that SQL commands follow a number of basic rules:

- SQL keywords are not normally case sensitive, though this in this tutorial all commands (SELECT, UPDATE etc) are upper-cased.
- Variable and parameter names are displayed here as lower-case.

- New-line characters are ignored in SQL, so a command may be all on one line or broken up across a number of lines for the sake of clarity.
- Many DBMS systems expect to have SQL commands terminated with a semi-colon character.

Data Definition Language (DDL) in SQL

The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. We can also define indexes (keys), specify links between tables, and impose constraints between database tables.

The most important DDL statements in SQL are:

- CREATE TABLE - creates a new database table
- ALTER TABLE - alters (changes) a database table
- DROP TABLE - deletes a database table

How to create table

Creating a database is remarkably straightforward. The SQL command which you have to give is just:

CREATE DATABASE dbname;

In this example you will call the database GJUniv, so the command which you have to give is:

CREATE DATABASE GJUniv;

Once the database is created it, is possible to start implementing the design sketched out previously.

So you have created the database and now it's time to use some SQL to create the tables required by the design. Note that all SQL keywords are shown in upper case, variable names in a mixture of upper and lower case.

The SQL statement to create a table has the basic form:

CREATE TABLE name(col1 datatype, col2 datatype, …);

So, to create our User table we enter the following command:

CREATE TABLE User (FirstName TEXT, LastName TEXT, UserID TEXT, Dept TEXT, EmpNo INTEGER, PCType TEXT);

The TEXT datatype, supported by many of the most common DBMS, specifies a string of characters of any length. In practice there is often a default string length which varies by product. In some DBMS TEXT is not supported, and instead a specific string length has to be declared. Fixed length strings are often called CHAR(x), VCHAR(x) or VARCHAR(x), where x is the string

length. In the case of INTEGER there are often multiple flavors of integer available. Remembering that larger integers require more bytes for data storage, the choice of int size is usually a design decision that ought to be made up front.

How to Modify table

Once a table is created it's structure is not necessarily fixed in stone. In time requirements change and the structure of the database is likely to evolve to match your wishes. SQL can be used to change the structure of a table, so, for example, if we need to add a new field to our User table to tell us if the user has Internet access, then we can execute an SQL ALTER TABLE command as shown below:

ALTER TABLE User ADD COLUMN Internet BOOLEAN;

To delete a column the ADD keyword is replaced with DROP, so to delete the field we have just added the SQL is:

ALTER TABLE User DROP COLUMN Internet;

How to delete table

If you have already executed the original CREATE TABLE command your database will already contain a table called User, so let's get rid of that using the DROP command:

DROP TABLE User;

And now we'll recreate the User table we'll use throughout the rest of this tutorial:

CREATE TABLE User (FirstName VARCHAR (20), LastName VARCHAR (20), UserID VARCHAR(12) UNIQUE, Dept VARCHAR(20), EmpNo INTEGER UNIQUE, PCType VARCHAR(20);

Data Manipulation Language in SQL (DML)

SQL language also includes syntax to update, insert, and delete records. These query and update commands together form the Data Manipulation Language (DML) part of SQL:

- INSERT INTO - inserts new data into a database table
- UPDATE - updates data in a database table
- DELETE - deletes data from a database table
- SELECT - extracts data from a database table

How to Insert Data

Having now built the structure of the database it is time to populate the tables with some data. In the vast majority of desktop database applications data entry is performed via a user interface built around some kind of GUI form. The form gives a representation of the information required for the application,

rather than providing a simple mapping onto the tables. So, in this sample application you would imagine a form with text boxes for the user details, drop-down lists to select from the PC table, drop-down selection of the software packages etc. In such a situation the database user is shielded both from the underlying structure of the database and from the SQL which may be used to enter data into it. However we are going to use the SQL directly to populate the tables so that we can move on to the next stage of learning SQL.

The command to add new records to a table (usually referred to as an append query), is:

INSERT INTO target [(field1[, field2[, ...]])]
VALUES (value1[, value2[, ...]);

So, to add a User record for user Jim Jones, we would issue the following INSERT query:

INSERT INTO User (FirstName, LastName, UserID, Dept, EmpNo, PCType) 6
VALUES ("Jim", "Jones", "Jjones","Finance", 9, "DellDimR450");

Obviously populating a database by issuing such a series of SQL commands is both tedious and prone to error, which is another reason why database applications have front-ends. Even without a specifically designed front-end, many database systems - including MS Access - allow data entry direct into tables via a spreadsheet-like interface.

The INSERT command can also be used to copy data from one table into another. For example, The SQL query to perform this is:

INSERT INTO User (FirstName, LastName, UserID, Dept, EmpNo, PCType, Internet)
SELECT FirstName, LastName, UserID, Dept, EmpNo, PCType, Internet
FROM NewUsers;

How to Update Data

The INSERT command is used to add records to a table, but what if you need to make an amendment to a particular record? In this case the SQL command to perform updates is the UPDATE command, with syntax:

UPDATE table
SET newvalue
WHERE criteria;

For example, let's assume that we want to move user Jim Jones from the Finance department to Marketing. Our SQL statement would then be:

UPDATE User
SET Dept="Marketing"
WHERE EmpNo=9;

Notice that we used the EmpNo field to set the criteria because we know it is unique. If we'd used another field, for example LastName, we might have accidentally updated the records for any other user with the same surname.

The UPDATE command can be used for more than just changing a single field or record at a time. The SET keyword can be used to set new values for a number of different fields, so we could have moved Jim Jones from Finance to marketing and changed the PCType as well in the same statement (SET Dept="Marketing", PCType="PrettyPC"). Or if all of the Finance department were suddenly granted Internet access then we could have issued the following SQL query:

```
UPDATE User
SET Internet=TRUE
WHERE Dept="Finance";
```

You can also use the SET keyword to perform arithmetical or logical operations on the values. For example if you have a table of salaries and you want to give everybody a 10% increase you can issue the following command:

```
UPDATE PayRoll
SET Salary=Salary * 1.1;
```

How to Delete Data

Now that we know how to add new records and to update existing records it only remains to learn how to delete records before we move on to look at how we search through and collate data. As you would expect SQL provides a simple command to delete complete records. The syntax of the command is:

```
DELETE [table.*]
FROM table
WHERE criteria;
```

Let's assume we have a user record for John Doe, (with an employee number of 99), which we want to remove from our User we could issue the following query:

```
DELETE *
FROM User
WHERE EmpNo=99;
```

In practice delete operations are not handled by manually keying in SQL queries, but are likely to be generated from a front end system which will handle warnings and add safe-guards against accidental deletion of records.

Note that the DELETE query will delete an entire record or group of records. If you want to delete a single field or group of fields without destroying that record then use an UPDATE query and set the fields to Null to over-write the data that needs deleting. It is also worth noting that the DELETE query does not do anything to the structure of the table itself, it deletes data only. To delete a table, or part of a table, then you have to use the DROP clause of an ALTER TABLE query.

Constraints in SQL

Data types are a way to limit the kind of data that can be stored in a table. For many applications, however, the constraint they provide is too coarse. For example, a column containing a product price

should probably only accept positive values. But there is no data type that accepts only positive numbers. Another issue is that you might want to constrain column data with respect to other columns or rows. For example, in a table containing product information, there should only be one row for each product number.

To that end, SQL allows you to define constraints on columns and tables. Constraints give you as much control over the data in your tables as you wish. If a user attempts to store data in a column that would violate a constraint, an error is raised. This applies even if the value came from the default value definition.

Check Constraints

A check constraint is the most generic constraint type. It allows you to specify that the value in a certain column must satisfy an arbitrary expression. For instance, to require positive product prices, you could use:

CREATE TABLE products (product_no integer, name text, price numeric CHECK (price > 0));

As you see, the constraint definition comes after the data type, just like default value definitions. Default values and constraints can be listed in any order. A check constraint consists of the key word *CHECK* followed by an expression in parentheses. The check constraint expression should involve the column thus constrained, otherwise the constraint would not make too much sense.

4.7.2 Not-Null Constraints

A not-null constraint simply specifies that a column must not assume the null value. A syntax example:

CREATE TABLE products (product_no integer NOT NULL, name text NOT NULL, price numeric);

A not-null constraint is always written as a column constraint. A not-null constraint is functionally equivalent to creating a check constraint CHECK (column_name IS NOT NULL), but in PostgreSQL creating an explicit not-null constraint is more efficient. The drawback is that you cannot give explicit names to not-null constraints created that way.

4.7.3 Unique Constraints

Unique constraints ensure that the data contained in a column or a group of columns is unique with respect to all the rows in the table. The syntax is

CREATE TABLE products (product_no integer UNIQUE, name text, price numeric);

when written as a column constraint, and

CREATE TABLE products (product_no integer, name text, price numeric,UNIQUE (product_no));

when written as a table constraint.

4.7.4 Primary Key Constraints

Technically, a primary key constraint is simply a combination of a unique constraint and a not-null constraint. So, the following two table definitions accept the same data:

CREATE TABLE products (product_no integer UNIQUE NOT NULL, name text, price numeric);

CREATE TABLE products (product_no integer PRIMARY KEY,name text,
price numeric);

Primary keys can also constrain more than one column; the syntax is similar to unique constraints:

CREATE TABLE example (a integer,b integer,c integer, PRIMARY KEY (a, c));

A primary key indicates that a column or group of columns can be used as a unique identifier for rows in the table. (This is a direct consequence of the definition of a primary key. Note that a unique constraint does not, in fact, provide a unique identifier because it does not exclude null values.) This is useful both for documentation purposes and for client applications. For example, a GUI application that allows modifying row values probably needs to know the primary key of a table to be able to identify rows uniquely.

Foreign Keys Constraints

A foreign key constraint specifies that the values in a column (or a group of columns) must match the values appearing in some row of another table. We say this maintains the referential integrity between two related tables.

Say you have the product table that we have used several times already:

CREATE TABLE products (product_no integer PRIMARY KEY, name text,
price numeric);

Let's also assume you have a table storing orders of those products. We want to ensure that the orders table only contains orders of products that actually exist. So we define a foreign key constraint in the orders table that references the products table:

CREATE TABLE orders (order_id integer PRIMARY KEY,product_no integer REFERENCES products (product_no), quantity integer);

Now it is impossible to create orders with product_no entries that do not appear in the products table. We say that in this situation the orders table is the referencing table and the products table is the referenced table. Similarly, there are referencing and referenced columns.

CHAPTER SIX

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:

$$\sigma_{<Selection\ Condition>}\ <Relation\ Name>$$

Example: Find all Employees with skill type of Database.

$$\sigma_{<SkillType="Database">}\ (Employee)$$

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.

$$\sigma_{\text{< SkillType ="Database" AND School="Unity">}} \textit{(Employee)}$$

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:

$$\pi_{\text{<Selected Attributes>}} \textit{<Relation Name>}$$

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

$$\pi_{\text{<FName, LName, Skill, Skill\_Level>}} \textit{(Employee)}$$

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:

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

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

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

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

$R - S \neq S - R$

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 | 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 ∧**(AND)**, ∨**(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:

∧ (AND), ∨ (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))}

CHAPTER SEVEN

RECORD STORAGE AND PRIMARY FILE ORGANIZATION

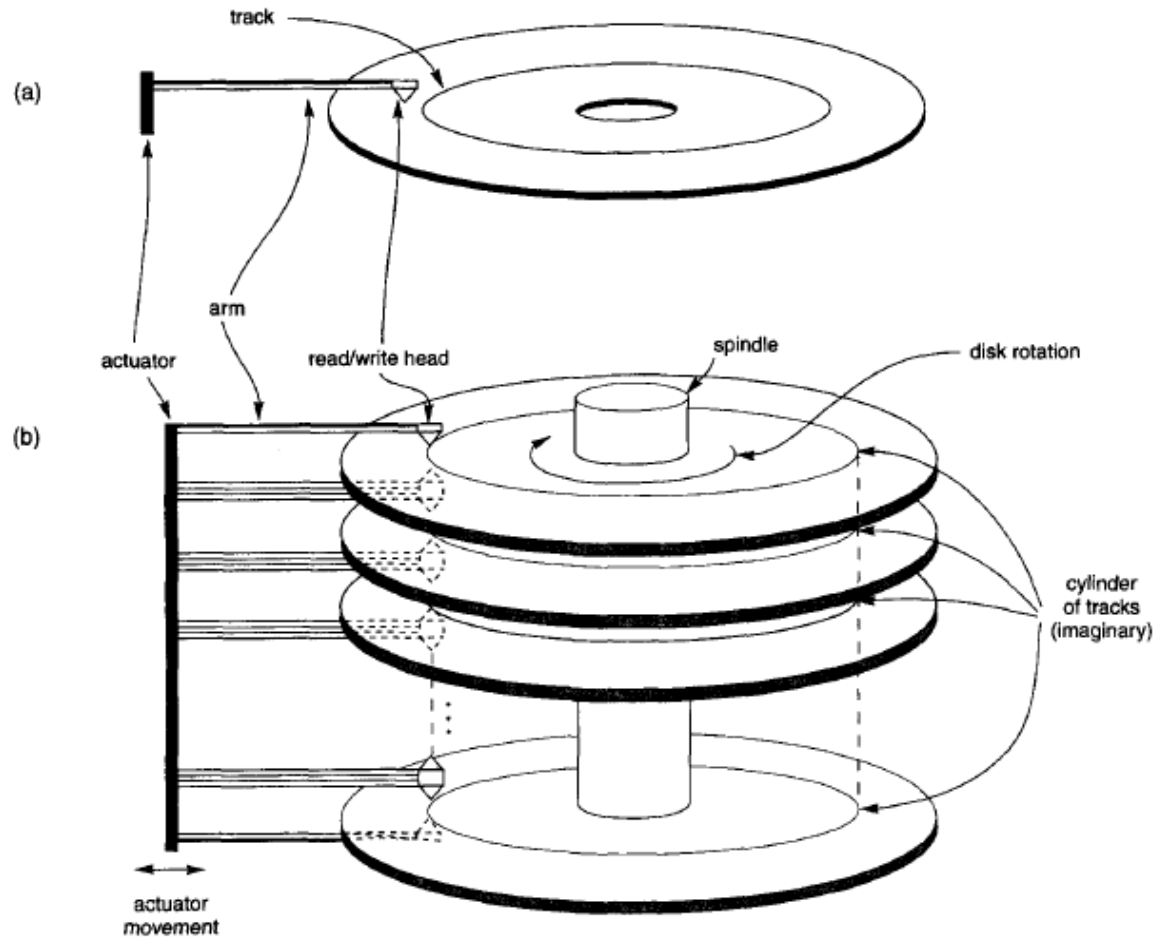
Introduction

Primary storage:
This category includes storage media that can be operated on directly by the computer central processing unit (CPU), such as the computer main memory and smaller but faster cache memories. Primary storage usually provides fast access to data but is of limited storage capacity.

Secondary storage:
This category includes magnetic disks, optical disks, and tapes. These devices usually have a larger capacity, cost less, and provide slower access to data than do primary storage devices. Data in secondary storage cannot be processed directly by the CPU; it must first be copied into primary storage.

Secondary storage Devices

- Whatever their capacity, disks are all made of magnetic material shaped as a thin circular disk (Figure 1 a) and protected by a plastic or acrylic cover.
- A disk is single sided if it stores information on only one of its surfaces and double-sided if both surfaces are used.
- To increase storage capacity, disks are assembled into a disk pack (Figure 1 b), which may include many disks and hence many surfaces.
- Information is stored on a disk surface in concentric circles of small width,4 each having a distinct diameter.
- Each circle is called a track.



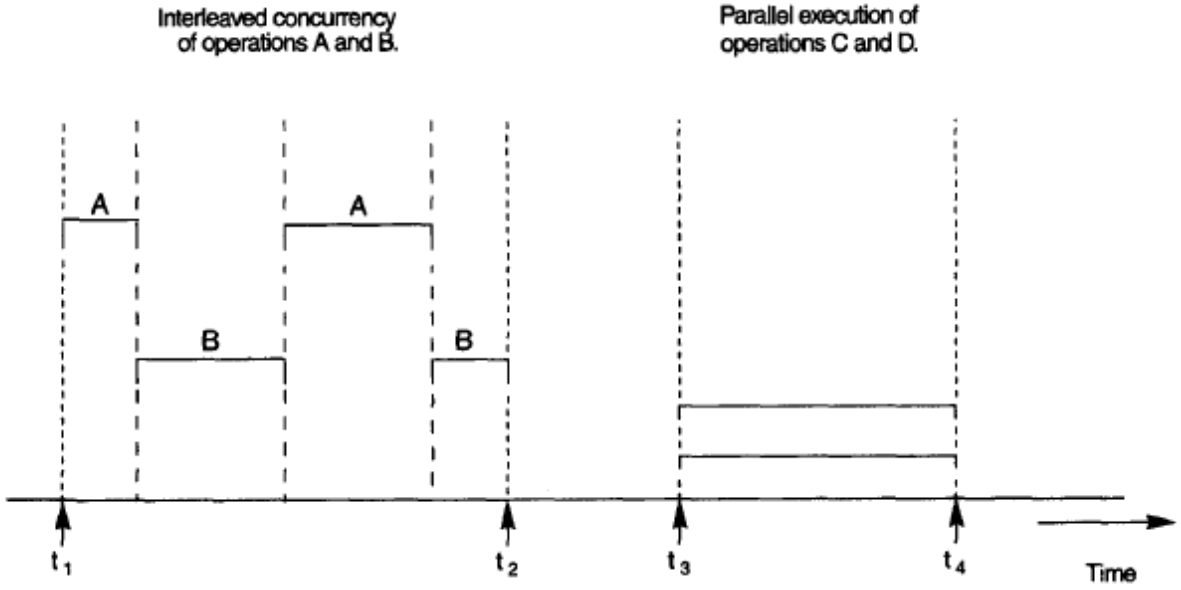
- The division of a track into equal-sized disk blocks (or pages) is set by the operating system during disk **formatting (or initialization).**
- Typical disk block sizes range from 512 to 4096 bytes.
- Blocks are separated by fixed-size **interblock gaps**, which include specially coded control information written during disk initialization.
- A disk is a random access addressable device.
- The **hardware address** of a block is a combination of a cylinder number, track number (surface number within the cylinder on which the track is located), and block number (within the track) is supplied to the disk i/o hardware.
- In many modern disk drives, a single number called **LBA (Logical Block Address)** which is a number between 0 and n (assuming the total capacity of the disk is n+1 blocks), is mapped automatically to the right block by the disk drive controller.
- For a read command, the block from disk is copied into the buffer; whereas for a write command, the contents of the buffer are copied into the disk block.

- Sometimes several contiguous blocks, called a **cluster**, may be transferred as a unit.
- The actual hardware mechanism that reads or writes a block is the disk **read/write head**, which is part of a system called a disk drive.
- A disk or disk pack is mounted in the disk drive, which includes a **motor** that rotates the disks.
- A read/write head includes an electronic component attached to a **mechanical arm**.
- All arms are connected to an **actuator** attached to another electrical motor, which moves the read/write heads in unison and positions them precisely over the cylinder of tracks specified in a block address.

Magnetic Tape Storage Devices

- The main characteristic of a tape is its requirement that we access the data blocks in sequential order.
- To get to a block in the middle of a reel of tape, the tape is mounted and then scanned until the required block gets under the read/write head.
- For this reason, tape access can be slow and tapes are not used to store online data, except for some specialized applications.
- However, tapes serve a very important function-that of backing up the database.

Buffering Of Blocks



- When a single CPU controls multiple processes, parallel execution is not possible. However, the processes can still run concurrently in an interleaved way.

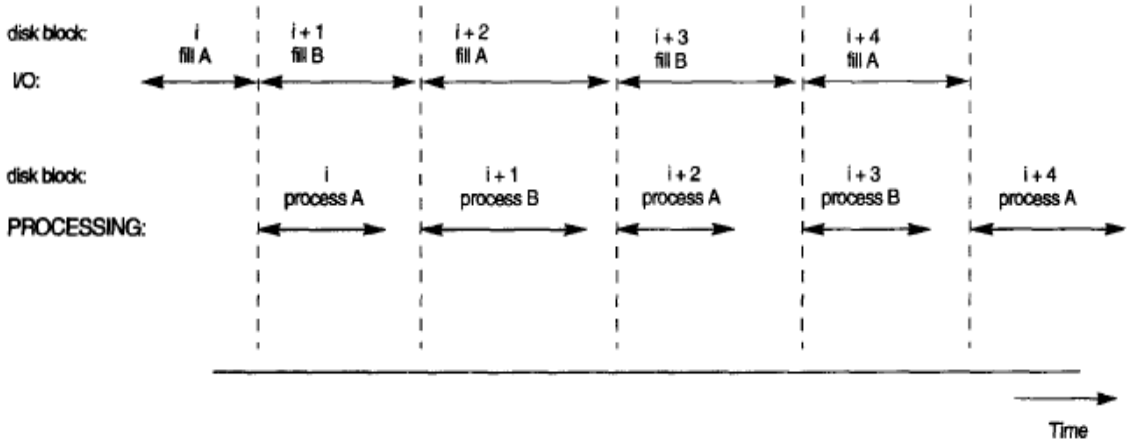


Figure illustrates how reading and processing can proceed in parallel when the time required to process a disk block in memory is less than the time required to read the next block and fill a buffer. The CPU can start processing a block once its transfer to main memory is completed; at the same time the disk I/O processor can be reading and transferring the next block into a different buffer. This technique is called **double buffering**.

Placing File Records On Disk

❖ Records and Record Types

- Data is usually stored in the form of records.
- Each record consists of a collection of related data values or items, where each value is formed of one or more bytes and corresponds to a particular field of the record.
- Records usually describe entities and their attributes.
- A collection of field names and their corresponding data types constitutes a record type or record format definition
- A data type, associated with each field, specifies the types of values a field can take.
- The data type of a field is usually one of the standard data types used in programming.

❖ Files, Fixed-length Records, and Variable length Records

- A file is a sequence of records.
- If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records.
- If different records in the file have different sizes, the file is said to be made up of variable-length records.

A file may have variable-length records for several reasons:

- The file records are of the same record type, but one or more of the fields are of varying size (variable-length fields).
- The file records are of the same record type, but one or more of the fields may have multiple values for individual records; such a field is called a repeating field and a group of values for the field is often called a repeating group.
- The file records are of the same record type, but one or more of the fields are optional; that is, they may have values for some but not all of the file records (optional fields).
- The file contains records of different record types and hence of varying size (mixed file).
- The fixed-length record has the same fields, and field lengths are fixed, so the system can identify the starting byte position of each field relative to the starting position of the record.
- This facilitates locating field values by programs that access such files.

❖ **Record Blocking and Spanned Versus Unspanned Records**

- The records of a file must be allocated to disk blocks because a block is the *unit of data transfer* between disk and memory.
- When the block size is larger than the record size, each block will contain numerous records, although some files may have unusually large records that cannot fit in one block.
- Suppose that the block size is B bytes for a file of fixed-length records of size R bytes, with B ≥R, we can fit ***bfr* = |B/R|**records per block, where the |(x)| *(floor function) rounds down* the number x to an integer. The value *bfr* is called the **blocking factor** for the file.
- To utilize this unused space, we can store part of a record on one block and the rest on another.
- A pointer at the end of the first block points to the block containing the remainder of the record in case it is not the next consecutive block on disk.
- This organization is called **spanned**, because records can span more than one block.
- Whenever a record is larger than a block, we *must* use a spanned organization. If records are not allowed to cross block boundaries, the organization is called **unspanned**.

❖ Allocating File Blocks on Disk

- There are several standard techniques for allocating the blocks of a file on disk.
- In **contiguous allocation** the file blocks are allocated to consecutive disk blocks. This makes reading the whole file very fast using double buffering, but it makes expanding the file difficult.
- In **linked allocation** each file block contains a pointer to the next file block. This makes it easy to expand the file but makes it slow to read the whole file.
- Another possibility is to use **indexed allocation**, where one or more index blocks contain pointers to the actual file blocks. It is also common to use combinations of these techniques.

❖ File Headers

- A file header or file descriptor contains information about a file that is needed by the system programs that access the file records.

- The header includes information to determine the disk addresses of the file blocks as well as to record format descriptions.

Operations On Files

- Operations on files are usually grouped into retrieval operations and update operations.
- The former do not change any data in the file, but only locate certain records so that their field values can be examined and processed.
- The latter change the file by insertion or deletion of records or by modification of field values.
- In either case, we may have to select one or more records for retrieval, deletion, or modification based on a selection condition (or filtering condition).
- ***Open:*** Prepares the file for reading or writing. Allocates appropriate buffers to hold file blocks from disk, and retrieves the file header. Sets the file pointer to the beginning of the file.
- ***Reset:*** Sets the file pointer of an open file to the beginning of the file.
- ***Find* (or *Locate):*** Searches for the first record that satisfies a search condition. Transfers the block containing that record into a main memory buffer.
- ***Read (or Get):*** Copies the current record from the buffer to a program variable in the user program.
- ***FindNext:*** Searches for the next record in the file that satisfies the search condition. Transfers the block containing that record into a main memory buffer.
- ***Delete:*** Deletes the current record and (eventually) updates the file on disk to reflect the deletion.
- ***Modify:*** Modifies some field values for the current record and (eventually) updates the file on disk to reflect the modification.
- ***Insert:*** Inserts a new record in the file by locating the block where the record is to be inserted, transferring that block into a main memory buffer, writing the record into the buffer, and (eventually) writing the buffer to disk to reflect the insertion.
- ***Close:*** Completes the file access by releasing the buffers and performing any other needed cleanup operations.
- ***FindAll:*** Locates *all* the records in the file that satisfy a search condition.
- ***Find* (or *Locate) n:*** Searches for the first record that satisfies a search condition and then continues to locate the next n - 1 records satisfying the same condition.
- ***FindOrdered:*** Retrieves all the records in the file in some specified order.
- ***Reorganize:*** Starts the reorganization process.

Files Of Unordered Records (Heap Files)

- In this simplest and most basic type of organization, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file.
- Such an organization is called a heap or pile file.
- Inserting a new record is *very efficient:* the last disk block of the file is copied into a buffer; the new record is added; and the block is then rewritten back to disk.
- To delete a record, a program must first find its block, copy the block into a buffer, then delete the record from the buffer, and finally rewrite the block back to the disk.
- This leaves unused space in the disk block.
- During reorganization, the file blocks are accessed consecutively, and records are packed by removing deleted records.
- To read all records in order of the values of some field, we create a sorted copy of the file.

Files of Ordered Records (Sorted Files)

- We can physically order the records of a file on disk based on the values of one of their fields-called the ordering field.
- This leads to an ordered or sequential file.
- If the ordering field is also a key field of the file-a field guaranteed to have a unique value in each record-then the field is called the ordering key for the file.
- Ordered records have some advantages over unordered files.
- First, reading the records in order of the ordering key values becomes extremely efficient, because no sorting is required.
- Second, finding the next record from the current one in order of the ordering key usually requires no additional block accesses, because the next record is in the same block as the current one (unless the current record is the last one in the block).
- Third, using a search condition based on the value of an ordering key field results in faster access when the binary search technique is used, which constitutes an improvement over linear searches, although it is not often used for disk files.

Hashing Technique

- Another type of primary file organization is based on hashing, which provides very fast access to records on certain search conditions. This organization is usually called a **hash file**.
- The search condition must be an equality condition on a single field, called the **hash field** of the file.
- In most cases, the hash field is also a key field of the file, in which case it is called the hash key.

- The idea behind hashing is to provide a function *h*, called a **hash function** or randomizing function, which is applied to the hash field value of a record and yields the *address* of the disk block in which the record is stored.

Internal Hashing

- For internal files, hashing is typically implemented as a hash table through the use of an array of records.
- Suppose that the array index range is from 0 to M – 1 then we have M slots whose addresses correspond to the array indexes.
- We choose a hash function that transforms the hash field value into an integer between 0 and M - 1.
- One common hash function is the *h(K)* = K mod M function, which returns the remainder of an integer hash field value K after division by M; this value is then used for the record address.
- Other hashing functions can be used.
- One technique, called **folding**, involves applying an arithmetic function such as *addition* or a logical function such as *exclusive* or to different portions of the hash field value to calculate the hash address.
- Another technique involves picking some digits of the hash field value-for example, the third, fifth, and eighth digits-to form the hash address.
- The problem with most hashing functions is that they do not guarantee that distinct values will hash to distinct addresses
- A **collision** occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record.
- In this situation, we must insert the new record in some other position, since its hash address is occupied.
- The process of finding another position is called **collision resolution.**
- There are numerous methods for collision resolution, including the following:
    - ***Open addressing:*** Proceeding from the occupied position specified by the hash address, the program checks the subsequent positions in order until an unused (empty) position is found.
    - ***Chaining:*** For this method, various overflow locations are kept, usually by extending the array with a number of overflow positions. In addition, a pointer field is added to each record location. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location

- ***Multiple hashing:*** The program applies a second hash function if the first results in a collision. If another collision results, the program uses open addressing or applies a third hash function and then uses open addressing if necessary.