MAA OMWATI DEGREE COLLLEGE HASSANPUR

IMPORTANT EXAM NOTES

CLASS – BCA 3RD

DATA BASE MANAGEMENT SYSTEM

CONTENTS

Unit – I

Database Management System: Introduction, Database System Applications, History of Database Systems, Database System Vs. File Processing System, View of Data, Data Abstraction, Instances and Schemas. DBMS Environment, Database languages, Database Models. Database design and ER Model: Physical, Conceptual and Logical Database design, Entity- Relationship Model: Entities, Relationships, Representation of entities, attributes, Representation of relationship set, Generalization, Aggregation, Conceptual design with ER Model

Unit – II

Relational Model: Introduction to the Relational Model, Attributes, Domains, Tuples, Relations and their schemes, relation representation, Keys, relationship, relational operations, , Integrity Constraint Over relations, Enforcing Integrity constraints, Querying relational data, View: Introduction to Views, Destroying / altering Views. Relational Algebra and Calculus: Relational Algebra & its operations, Relational calculus & its types, Power of Algebra and calculus. Lab Problem(s): Creation and Querying relational data with SQL

Unit – III

Normalization: Schema Refinement, Problems caused by redundancy, Decomposition & its properties; Normalization: First, Second, Third Normal forms, BCNF, Multivalued Dependencies, Join Dependencies. Transaction Management & Concurrency Control: ACID properties, Transactions and Schedules, Concurrent execution of transaction, Serializability and Recoverability, Lock based Concurrency control, Lock Management, Lock Conversion, Dealing with deadlocks, Concurrency without Locking.

Unit – IV

Crash Recovery and Backup: Failure classifications, storage structure, Recovery & Atomicity, Log base recovery, Recovery with concurrent transactions, Failure with loss of nonvolatile storage, Database backup & recovery from catastrophic failure, Remote Backup System.

Storage and File Organization : Overview of physical storage media, Storage access; File organization, Operations on Files, Serial Files, Sequential Files , Index-Sequential Files, Direct Files.

LECTURE-2: DBMS

Database:

A database is organized collection of related data of an organization stored in formatted way which is shared by multiple users.

The main feature of data in a database are:

1. It must be well organized
2. It is related
3. It is accessible in a logical order without any difficulty
4. It is stored only once

For example consider the roll no, name, address of a student stored in a student file. It is collection of related data with an implicit meaning. Data in the database may be persistent, integrated and shared.

Persistent:

If data is removed from database due to some explicit request from user to remove.

Integrated:

A database can be a collection of data from different files and when any redundancy among those files are removed from database is said to be integrated data.

Sharing Data:

The data stored in the database can be shared by multiple users simultaneously without affecting the correctness of data.

Why Database:

In order to overcome the limitation of a file system, a new approach was required. Hence a database approach emerged. A database is a persistent collection of logically related data. The initial attempts were to provide a centralized collection of data. A database has a self describing nature. It contains not only the data sharing and integration of data of an organization in a single database.

A small database can be handled manually but for a large database and having multiple users it is difficult to maintain it. In that case a computerized database is useful.

The advantages of database system over traditional, paper based methods of record keeping are:

- **Compactness**: No need for large amount of paper files
- **Speed**: The machine can retrieve and modify the data more faster way then human being
- **Less drudgery**: Much of the maintenance of files by hand is eliminated
- **Accuracy:** Accurate, up-to-date information is fetched as per requirement of the user at any time.

Database Management System (DBMS):

A database management system consists of collection of related data and refers to a set of programs for defining, creation, maintenance and manipulation of a database.

Function of DBMS:

1. **Defining database schema**: it must give facility for defining the database structure also specifies access rights to authorized users.
2. **Manipulation of the database:** The dbms must have functions like insertion of record into database, updation of data, deletion of data, retrieval of data
3. **Sharing of database:** The DBMS must share data items for multiple users by maintaining consistency of data.
4. **Protection of database:** It must protect the database against unauthorized users.
5. **Database recovery:** If for any reason the system fails DBMS must facilitate data base recovery.

Advantages of DBMS:

Reduction of redundancies:
Centralized control of data by the DBA avoids unnecessary duplication of data and effectively reduces the total amount of data storage required avoiding duplication in the elimination of the inconsistencies that tend to be present in redundant data files.

Sharing of Data:
A database allows the sharing of data under its control by any number of application programs or users.

Data Integrity:
Data integrity means that the data contained in the database is both accurate and consistent. Therefore data values being entered for storage could be checked to ensure that they fall with in a specified range and are of the correct format.

Data Security:
The DBA who has the ultimate responsibility for the data in the dbms can ensure that proper access procedures are followed including proper authentication to access to the DataBase System and additional check before permitting access to sensitive data.

Conflict Resolution:
DBA resolve the conflict on requirements of various user and applications. The DBA chooses the best file structure and access method to get optional performance for the application.

Data Independence:
Data independence is usually considered from two points of views; physically data independence and logical data independence.

Physical Data Independence allows changes in the physical storage devices or organization of the files to be made without requiring changes in the conceptual view or any of the external views and hence in the application programs using the data base.

Logical Data Independence indicates that the conceptual schema can be changed without affecting the existing external schema or any application program.

Disadvantage of DBMS:

1. DBMS software and hardware (networking installation) cost is high
2. The processing overhead by the dbms for implementation of security, integrity and sharing of the data.
3. Centralized database control
4. Setup of the database system requires more knowledge, money, skills, and time.
5. The complexity of the database may result in poor performance.

LECTURE-3: 3 level Architecture of DBMS

Database Basics:

Data Item:
The data item is also called as field in data processing and is the smallest unit of data that has meaning to its users.
Eg: “e101”, ”sumit”

Entities and attributes:
An entity is a thing or object in the real world that is distinguishable from all other objects
Eg: Bank, employee, student

Attributes are properties are properties of an entity.
Eg: Empcode, ename, rolno, name

Logical data and physical data :
Logical data are the data for the table created by user in primary memory.
Physical data refers to the data stored in the secondary memory.

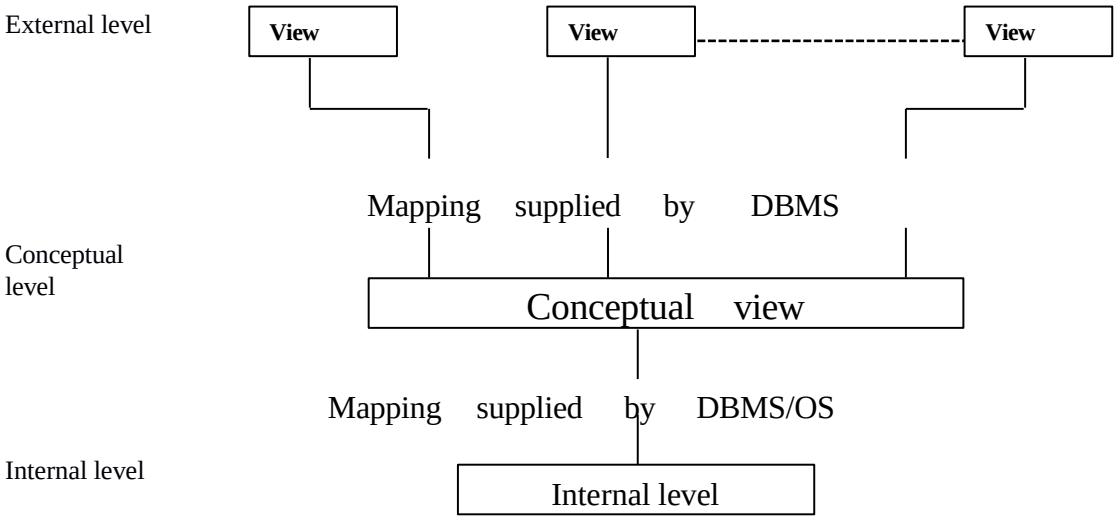
Schema and sub-schema :
A schema is a logical data base description and is drawn as a chart of the types of data that are used. It gives the names of the entities and attributes and specify the relationships between them.

A database schema includes such information as :

- Characteristics of data items such as entities and attributes .
- Logical structures and relationships among these data items .
- Format for storage representation.
- Integrity parameters such as physical authorization and back up policies.

A *subschema* is derived schema derived from existing schema as per the user requirement. There may be more then one subschema create for a single conceptual schema.

Three Level Architecture of DBMS :



A database management system that provides three level of data is said to follow three-level architecture .

- External level
- Conceptual level
- Internal level

External Level :
The external level is at the highest level of database abstraction . At this level, there will be many views define for different users requirement. A view will describe only a subset of the database. Any number of user views may exist for a given global schema(coneptual schema).

For example, each student has different view of the time table. the view of a student of BTech (CSE) is different from the view of the student of Btech (ECE). Thus this level of abstraction is concerned with different categories of users.
Each external view is described by means of a schema called sub schema.

Conceptual Level :
At this level of database abstraction all the database entities and the relationships among them are included. One conceptual view represents the entire database. This conceptual view is defined by the conceptual schema.

The conceptual schema hides the details of physical storage structures and concentrate on describing entities, data types, relationships, user operations and constraints.

It describes all the records and relationships included in the conceptual view. There is only one conceptual schema per database. It includes feature that specify the checks to relation data consistency and integrity.

Internal level :
It is the lowest level of abstraction closest to the physical storage method used. It indicates how the data will be stored and describes the data structures and access methods to be used by the database. The internal view is expressed by internal schema.

The following aspects are considered at this level:

1. Storage allocation e.g: B-tree, hashing
2. Access paths eg. specification of primary and secondary keys, indexes etc
3. Miscellaneous eg. Data compression and encryption techniques, optimization of the internal structures.

Database Users :

Naive Users :
Users who need not be aware of the presence of the database system or any other system supporting their usage are considered naïve users . A user of an automatic teller machine falls on this category.

Online Users :
These are users who may communicate with the database directly via an online terminal or indirectly via a user interface and application program. These users are aware of the database system and also know the data manipulation language system.

Application Programmers :
Professional programmers who are responsible for developing application programs or user interfaces utilized by the naïve and online user falls into this category.

Database Administration :
A person who has central control over the system is called database administrator .
The function of DBA are :

1. Creation and modification of conceptual Schema definition
2. Implementation of storage structure and access method.
3. Schema and physical organization modifications .
4. Granting of authorization for data access.
5. Integrity constraints specification.
6. Execute immediate recovery procedure in case of failures
7. Ensure physical security to database

Database language :

1) **Data definition language (DDL) :**
   DDL is used to define database objects .The conceptual schema is specified by a set of definitions expressed by this language. It also gives some details about how to implement this schema in the physical devices used to store the data. This definition includes all the entity sets and their associated attributes and their relationships. The result of DDL statements will be a set of tables that are stored in special file called data dictionary.

2) **Data Manipulation Language (DML) :**
   A DML is a language that enables users to access or manipulate data stored in the database. Data manipulation involves retrieval of data from the database, insertion of new data into the database and deletion of data or modification of existing data.

   There are basically two types of DML:
   - **Procedural**: Which requires a user to specify what data is needed and how to get it.
   - **Non-Procedural:** which requires a user to specify what data is needed with out specifying how to get it.

3) **Data Control Language (DCL):**
   This language enables user to grant authorization and canceling authorization of database objects.

LECTURE-4: Elements of DBMS

Elements of DBMS:

DML Pre-Compiler:
It converts DML statements embedded in an application program to normal procedure calls in the host language. The pre-complier must interact with the query processor in order to generate the appropriate code.

DDL Compiler:
The DDL compiler converts the data definition statements into a set of tables. These tables contains information concerning the database and are in a form that can be used by other components of the dbms.

File Manager:
File manager manages the allocation of space on disk storage and the data structure used to represent information stored on disk.

Database Manager:
A database manager is a program module which provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.

The responsibilities of database manager are:

1. **Interaction with File Manager**: The data is stored on the disk using the file system which is provided by operating system. The database manager translate the different DML statements into low-level file system commands so the database manager is responsible for the actual storing, retrieving and updating of data in the database.

2. **Integrity Enforcement:** The data values stored in the database must satisfy certain constraints (eg: the age of a person can't be less then zero). These constraints are specified by DBA. Data manager checks the constraints and if it satisfies then it stores the data in the database.

3. **Security Enforcement:** Data manager checks the security measures for database from unauthorized users.

4. **Backup and Recovery:** Database manager detects the failures occur due to different causes (like disk failure, power failure, deadlock, software error) and restores the database to original state of the database.

5. **Concurrency Control:** When several users access the same database file simultaneously, there may be possibilities of data inconsistency. It is responsible of database manager to control the problems occur for concurrent transactions.

Query Processor:
The query processor used to interpret to online user’s query and convert it into an efficient series of operations in a form capable of being sent to the data manager for execution. The query processor

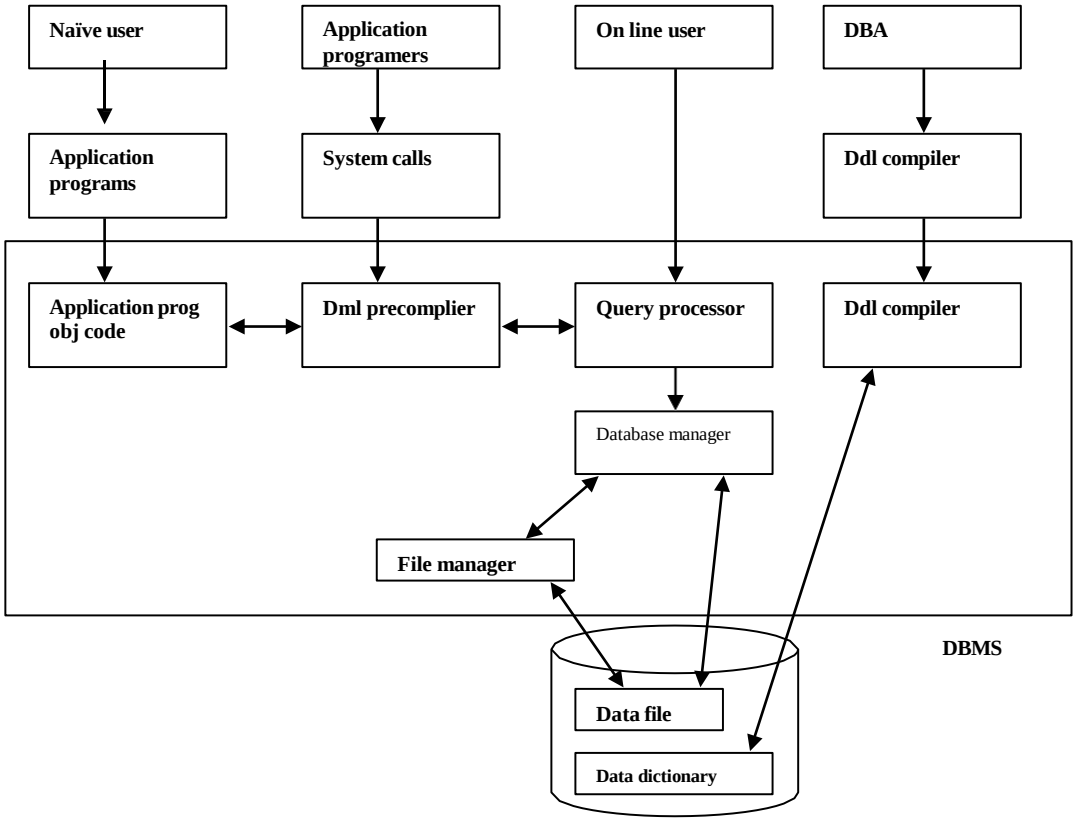
uses the data dictionary to find the details of data file and using this information it create query plan/access plan to execute the query.

Data Dictionary:
Data dictionary is the table which contains the information about database objects. It contains information like

1. external, conceptual and internal database description
2. description of entities, attributes as well as meaning of data elements
3. synonyms, authorization and security codes
4. database authorization

The data stored in the data dictionary is called *meta data.*

DBMS STRUCTURE:



Que: List four significant differences between a File-Processing System and a DBMS.

Ans: Some major differences between a database management system and a file-processing system are:

- Both systems contain a collection of data and a set of programs which access that data. A database management system coordinates both the physical and the logical access to the data, whereas a file-processing system coordinates only the physical access.

- A database management system reduces the amount of data duplication by ensuring that a physical piece of data is available to all programs authorized to have access to it, where as data written by one program in a file-processing system may not be readable by another program.

- A database management system is designed to allow flexible access to data (i.e., queries), whereas a file-processing system is designed to allow predetermined access to data (i.e., compiled programs).

- A database management system is designed to coordinate multiple users accessing the same data at the same time. A file-processing system is usually designed to allow one or more programs to access different data files at the same time. In a file-processing system, a file can be accessed by two programs concurrently only if both programs have read-only access to the file.

Que: Explain the difference between physical and logical data independence.

Ans:

- Physical data independence is the ability to modify the physical scheme without making it necessary to rewrite application programs. Such modifications include changing from unblocked to blocked record storage, or from sequential to random access files.

- Logical data independence is the ability to modify the conceptual scheme without making it necessary to rewrite application programs. Such a modification might be adding a field to a record; an application program's view hides this change from the program.

Que: List five responsibilities of a database management system. For each responsibility, explain the problems that would arise if the responsibility were not discharged.

Ans: A general purpose database manager (DBM) has five responsibilities:

a. interaction with the file manager.
b. integrity enforcement.
c. security enforcement.
d. backup and recovery.
e. concurrency control.

If these responsibilities were not met by a given DBM (and the text points out that sometimes a responsibility is omitted by design, such as concurrency control on a single-user DBM for a micro computer) the following problems can occur, respectively:

a. No DBM can do without this, if there is no file manager interaction then nothing stored in the files can be retrieved.

b. Consistency constraints may not be satisfied, when account balances could go below the minimum allowed, employees could earn too much overtime (e.g.,hours > 80) or, airline pilots may fly more hours than allowed by law.

c. Unauthorized users may access the database, or users authorized to access part of the

database may be able to access parts of the database for which they lack authority. For example, a high school student could get access to national defense secret codes, or employees could find out what their supervisors earn.

d. Data could be lost permanently, rather than at least being available in a consistent state that existed prior to a failure.

e. Consistency constraints may be violated when intgrity constraints failed in a transaction. For example, incorrect bank balances might be reflected due to simultaneous withdrawals and deposits, and so on.

Que. What are five main functions of a database administrator?

Ans: Five main functions of a database administrator are:

- To create the scheme definition
- To define the storage structure and access methods
- To modify the scheme and/or physical organization when necessary
- To grant authorization for data access
- To specify integrity constraints

Que: List six major steps that you would take in setting up a database for a particular enterprise.

Ans: Six major steps in setting up a database for a particular enterprise are:

- Define the high level requirements of the enterprise (this step generates a document known as the system requirements specification.)
- Define a model containing all appropriate types of data and data relationships.
- Define the integrity constraints on the data.
- Define the physical level.
- For each known problem to be solved on a regular basis (e.g., tasks to be carried out by clerks or Web users) define a user interface to carry out the task, and write the necessary application programs to implement the user interface.
- Create/initialize the database.

EXERCISE:

1. What is database management system?
2. What are the disadvantage of file processing system?
3. State advantage and disadvantage of database management system.
4. What are different types of database users?
5. What is data dictionary and what are its contents?
6. What are the functions of DBA?
7. What are the different database languages? Explain with example.
8. Explain the three layer architecture of DBMS.
9. Differentiate between physical data independence and logical data independence.
10. Explain the functions of database manager.
11. Explain meta data.

LECTURE-5: ER-MODEL

Data Model:
The data model describes the structure of a database. It is a collection of conceptual tools for describing data, data relationships and consistency constraints and various types of data models such as

1. Object based logical model
2. Record based logical model
3. Physical model

Types of data model:

1. Object based logical model
    a. ER-model
    b. Functional model
    c. Object oriented model
    d. Semantic model
2. Record based logical model
    a. Hierarchical database model
    b. Network model
    c. Relational model
3. Physical model

Entity Relationship Model (ER Model)
The entity-relationship data model perceives the real world as consisting of basic objects, called entities and relationships among these objects. It was developed to facilitate database design by allowing specification of an enterprise schema which represents the overall logical structure of a data base.

Main Features of ER-MODEL:

- Entity relationship model is a high level conceptual model
- It allows us to describe the data involved in a real world enterprise in terms of objects and their relationships.
- It is widely used to develop an initial design of a database
- It provides a set of useful concepts that make it convenient for a developer to move from a basic set of information to a detailed and description of information that can be easily implemented in a database system
- It describes data as a collection of entities, relationships and attributes.

Basic Concepts:
The E-R data model employs three basic notions : entity sets, relationship sets and attributes.

Entity Sets:
An entity is a “thing” or “object” in the real world that is distinguishable from all other objects. For example, each person in an enterprise is an entity. An entity has a set properties and the values for some set of properties may uniquely identify an entity. BOOK is entity and its properties (called as attributes) bookcode, booktitle, price etc.

An entity set is a set of entities of the same type that share the same properties, or attributes. The set

of all persons who are customers at a given bank.

Attributes:
An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set.

Customer is an entity and its attributes are **customerid, custmername, custaddress** etc.

An attribute as used in the E-R model, can be characterized by the following attribute types.

a) **Simple and Composite Attribute:**
Simple attributes are the attributes which can't be divided into sub parts, e.g. customerid, empno
Composite attributes are the attributes which can be divided into subparts, e.g. name consisting of first name, middle name, last name and address consisting of city, pincode, state.

b) **Single-Valued and Multi-Valued Attribute:**
The attribute having unique value is single –valued attribute, e.g. empno, customerid, regdno etc.
The attribute having more than one value is multi-valued attribute, eg: phone-no, dependent name, vehicle.

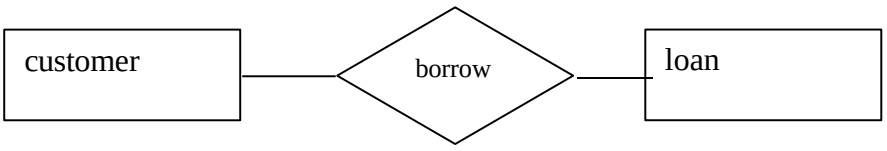
c) **Derived Attribute:**
The values for this type of attribute can be derived from the values of existing attributes, e.g. age which can be derived from currentdate – birthdate and experience_in_year can be calculated as currentdate-joindate.

d) **NULL Valued Attribute:**
The attribute value which is not known to user is called NULL valued attribute.

Relationship Sets:
A relationship is an association among several entities. A relationship set is a set of relationships of the same type. Formally, it is a mathematical relation on n>=2 entity sets. If E_1, E_2…E_n are entity sets, then a relation ship set R is a subset of

$$\{(e_1,e_2,\ldots e_n) \mid e_1 \in E_1, e_2 \in E_2.., e_n \in E_n\}$$

where $(e_1,e_2,\ldots e_n)$ is a relation ship.

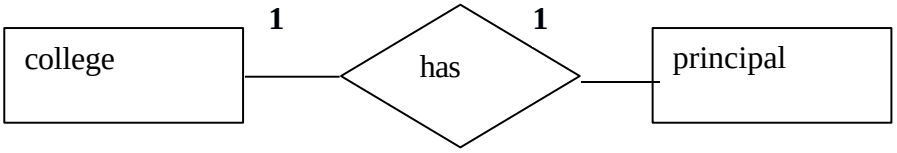


Consider the two entity sets customer and loan. We define the relationship set borrow to denote the association between customers and the bank loans that the customers have.

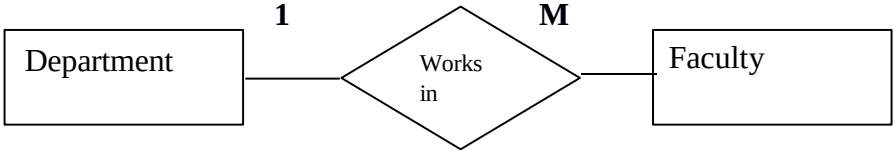
Mapping Cardinalities:
Mapping cardinalities or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set. Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets. For a binary relationship set R between entity sets A and B, the mapping

cardinalities must be one of the following:

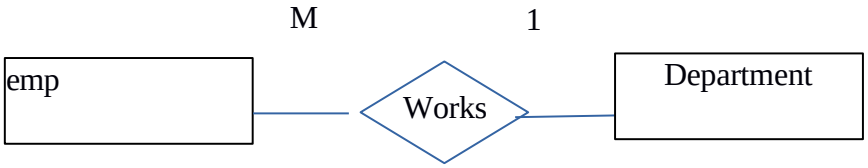
1. One to One:
An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.
Eg: relationship between college and principal



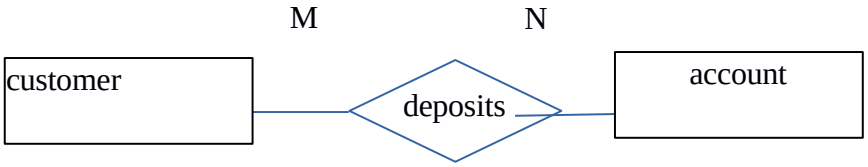
2. One to Many:
An entity in A is associated with any number of entities in B. An entity in B is associated with at the most one entity in A.
Eg: Relationship between department and faculty



3. Many to One:
An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A.



4. Many to Many:
Entities in A and B are associated with any number of entities from each other.



More about Entities and Relationship:

Recursive Relationships:
When the same entity type participates more than once in a relationship type in different roles, the relationship types are called recursive relationships.

Participation Constraints:
The participation constraints specify whether the existence of any entity depends on its being related to another entity via the relationship. There are two types of participation constraints

a) Total : When all the entities from an entity set participate in a relationship type, is called total participation. For example, the participation of the entity set student on the relationship set must 'opts' is said to be total because every student enrolled must opt for a course.

b) Partial: When it is not necessary for all the entities from an entity set to particapte in a relationship type, it is called partial participation. For example, the participation of the entity set student in 'represents' is partial, since not every student in a class is a class representative.

Weak Entity:
Entity types that do not contain any key attribute, and hence can not be identified independently are called weak entity types. A weak entity can be identified by uniquely only by considering some of its attributes in conjunction with the primary key attribute of another entity, which is called the identifying owner entity.

Generally a partial key is attached to a weak entity type that is used for unique identification of weak entities related to a particular owner type. The following restrictions must hold:

- The owner entity set and the weak entity set must participate in one to may relationship set. This relationship set is called the identifying relationship set of the weak entity set.
- The weak entity set must have total participation in the identifying relationship.

Example:
Consider the entity type Dependent related to Employee entity, which is used to keep track of the dependents of each employee. The attributes of Dependents are: name, birthdate, sex and relationship. Each employee entity set is said to its own the dependent entities that are related to it. However, not that the 'Dependent' entity does not exist of its own, it is dependent on the Employee entity.

Keys:

Super Key:
A super key is a set of one or more attributes that taken collectively, allow us to identify uniquely an entity in the entity set. For example , customer-id, (cname, customer-id), (cname, telno)

Candidate Key:
In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following properties:

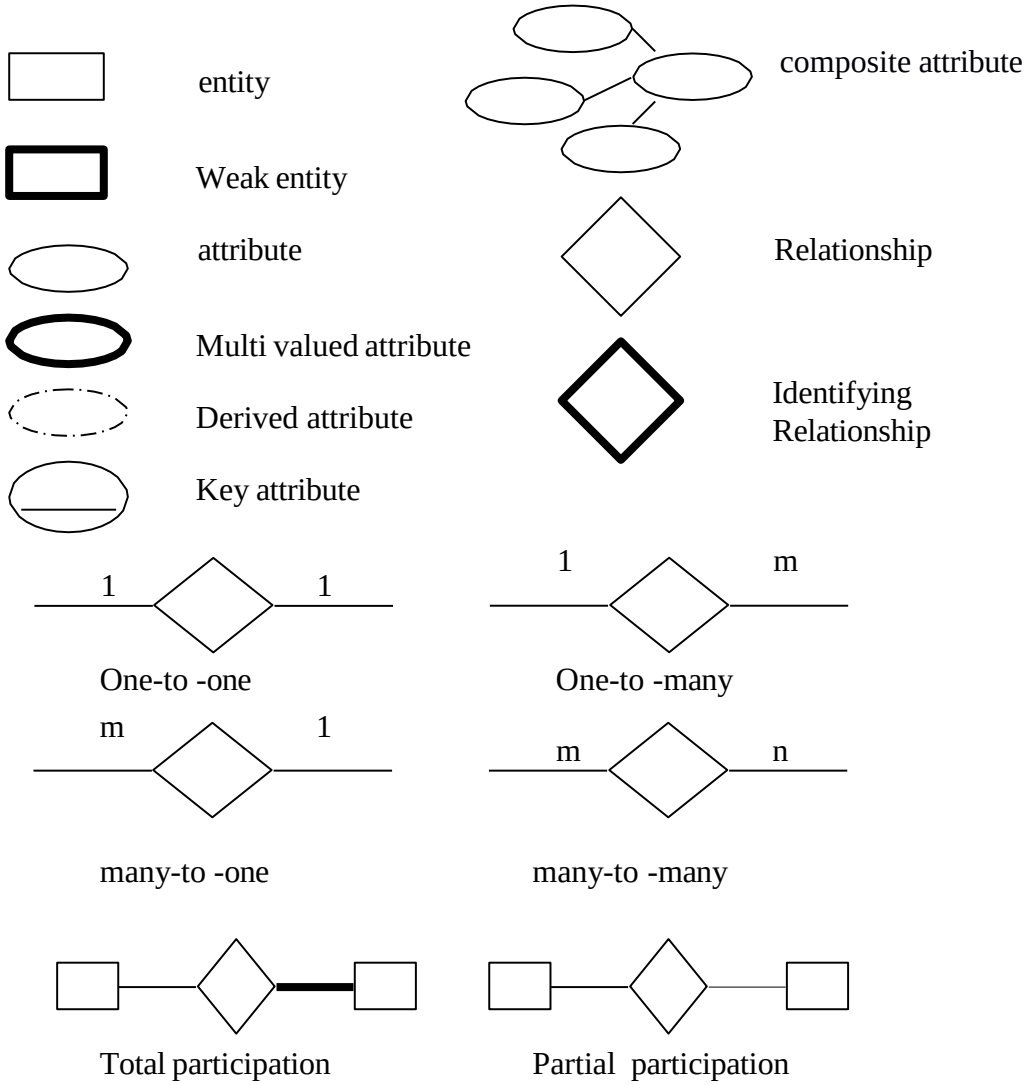
1. *Uniqueness: N*o two distinct tuples in R have the same values for the candidate key
2. *Irreducible:* No proper subset of the candidate key has the uniqueness property that is the candidate key. Eg: (cname,telno)

Primary Key:
The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. The remaining candidate keys if any, are called *Alternate Key*.

LECTURE-6: ER-DIAGRAM:

The overall logical structure of a database using ER-model graphically with the help of an ER-diagram.

Symbols use ER- diagram:



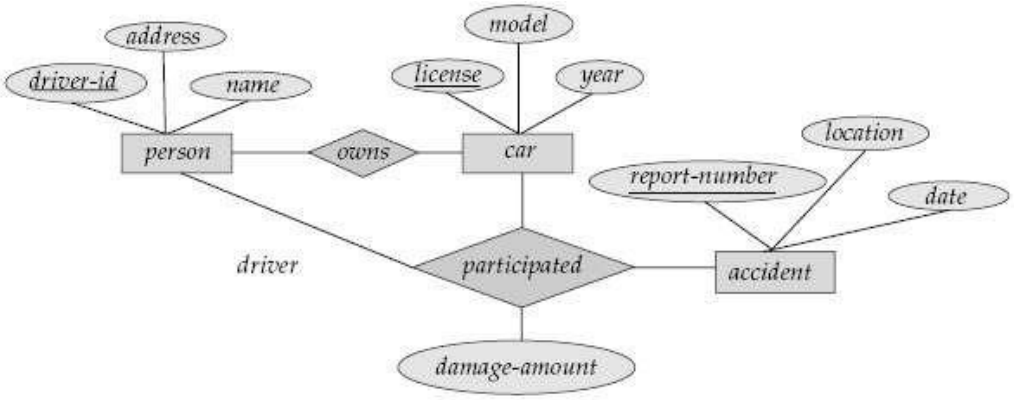


Figure 2.1 E-R diagram for a Car-insurance company.

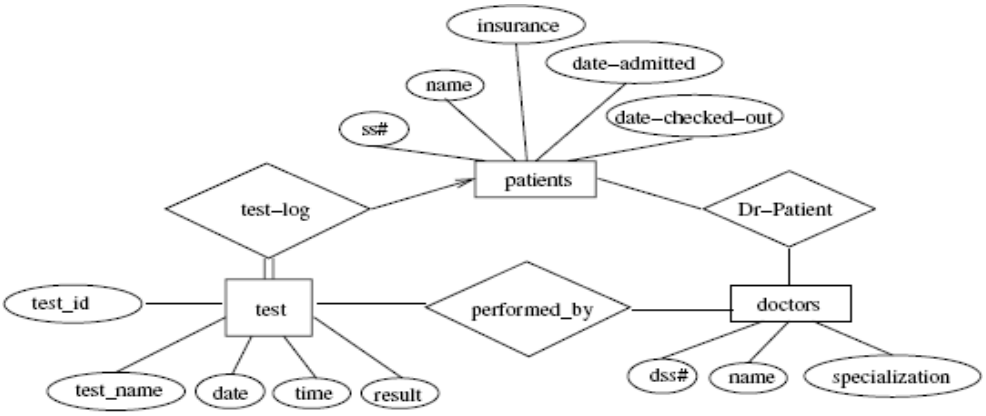


Figure 2.2 E-R diagram for a hospital.

A Univeristy registrar's office maintains data about the following entities:

(a) Course, includeing number,title,credits,syllabus and prereqisites

(b) course offering,including course number,year,semester,section number,instructor timings, and class room

(c) Students including student-id,name and program

(d) Instructors, including identification number,name,department and title

further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriate modeled.

Construct an E-R diagram for the registrar's office. Document all assumptions that you may make about the mapping constratints

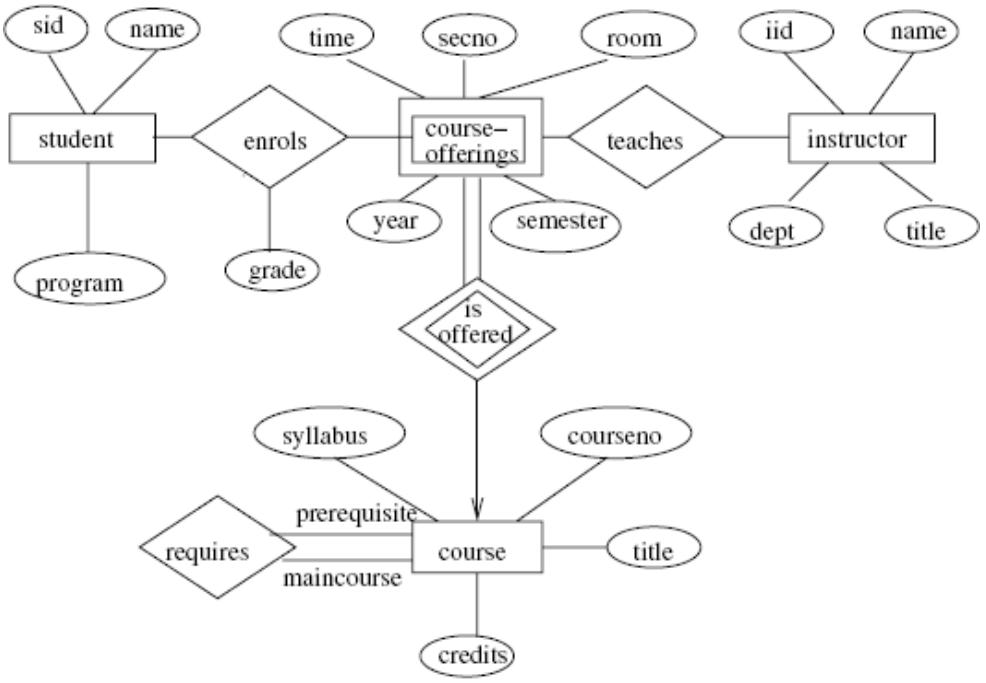


Figure 2.3 E-R diagram for a university.

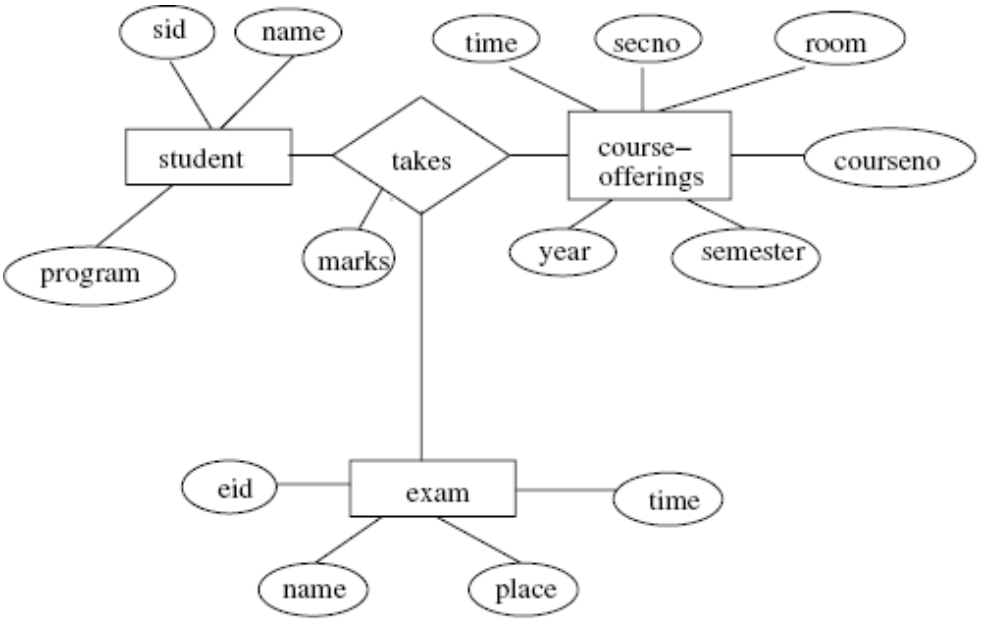


Figure 2.4 E-R diagram for marks database.

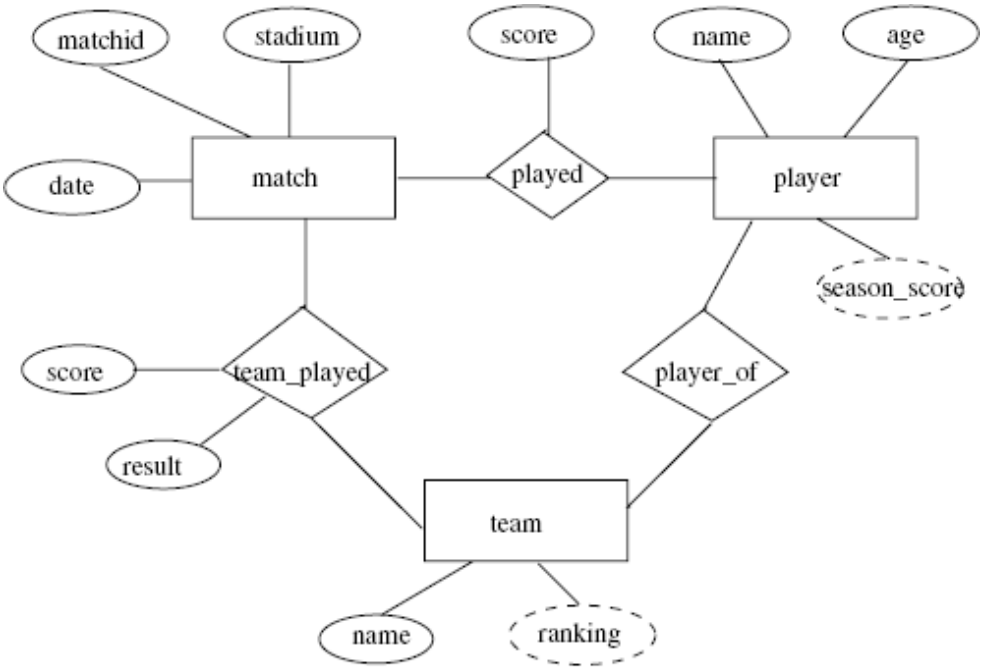


Figure 2.7 E-R diagram for all teams statistics.

Cosidet a university database for the scheduling of class rooms for final exams. This database could be modeled as the single entity set exam, with attributes course-name,section-number,room-number and time, Alternatively, one or more additional entity sets would be defined, along with relationship sets to replae some of the attributes of the exam entity set, as

- course with attributes name,department and c-number
- section with attributes s-number and enrollment and dependent as a weak entity set on course
- room with attributes r-number,capacity and building

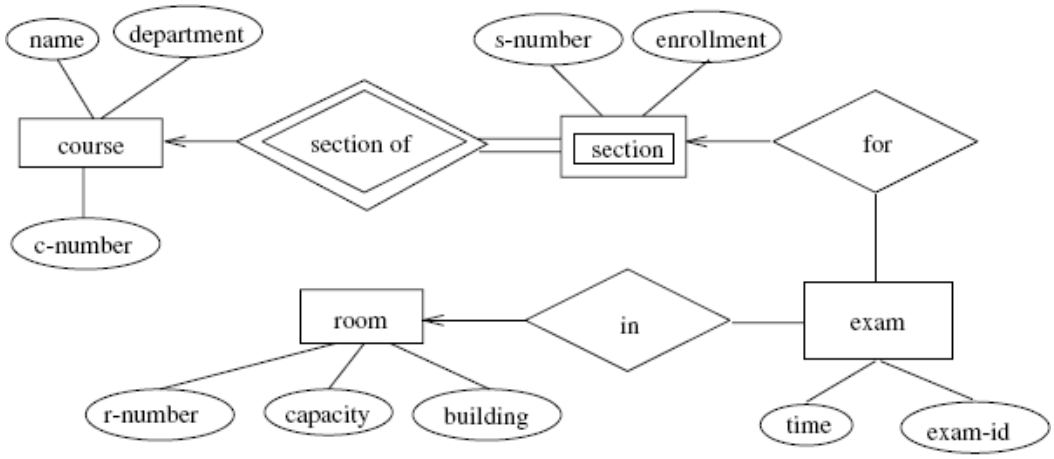


Figure 2.12 E-R diagram for exam scheduling.

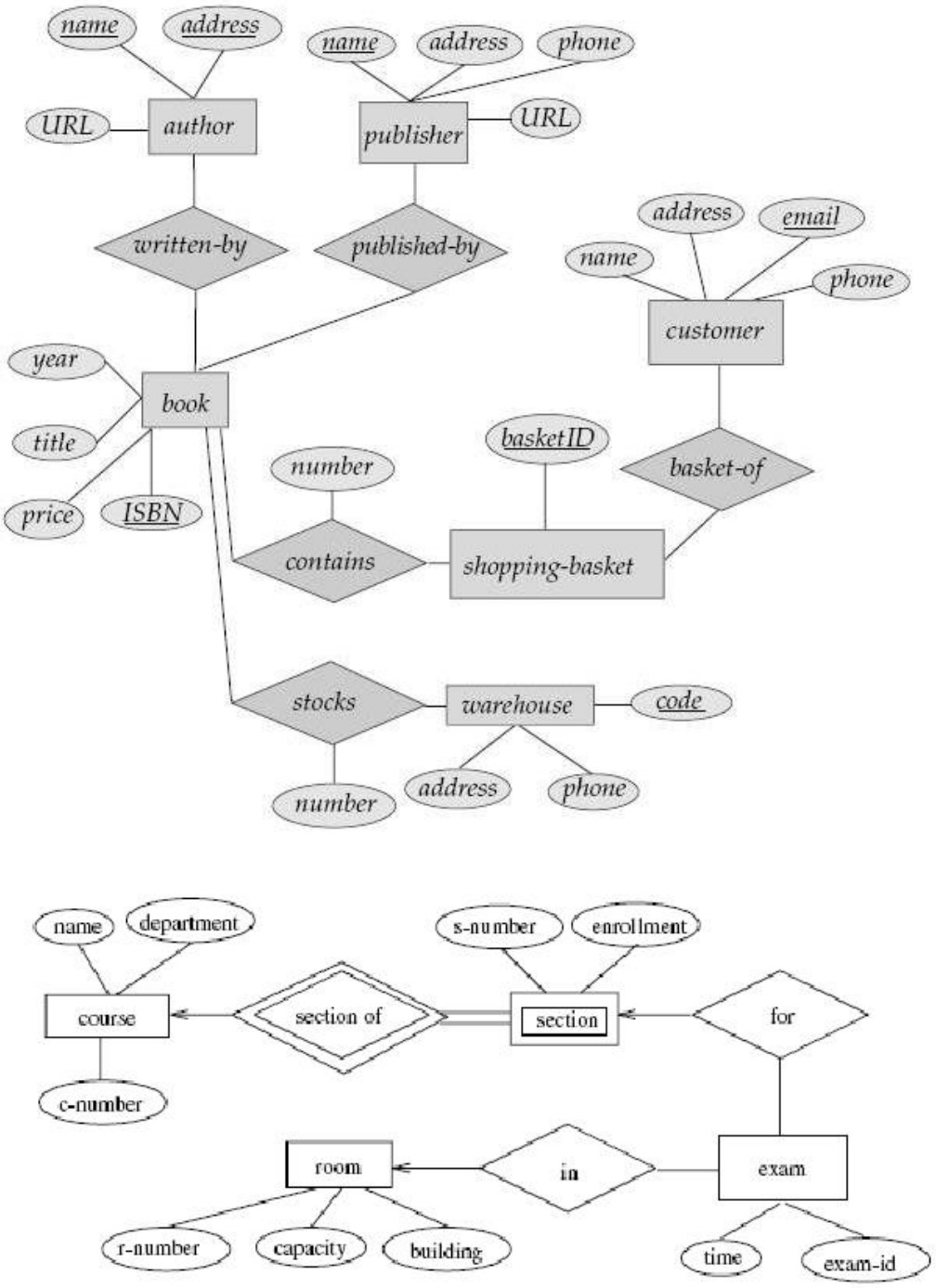


Figure 2.12 E-R diagram for exam scheduling.

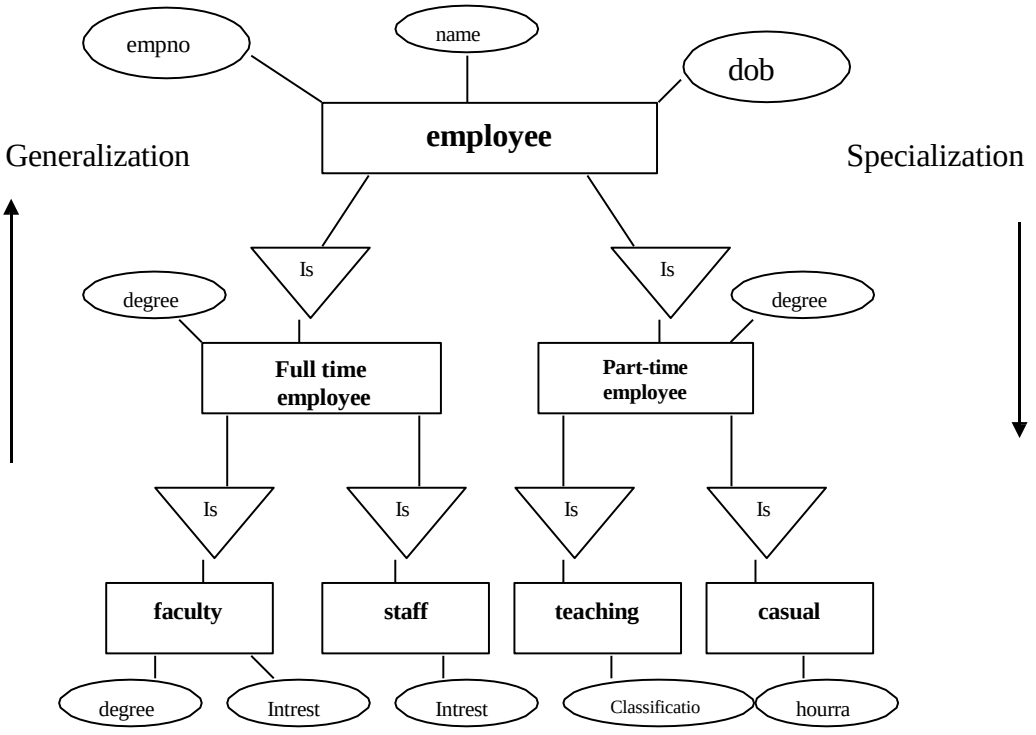
LECTURE-7: Advanced ER-Diagram:

Abstraction is the simplification mechanism used to hide superfluous details of a set of objects. It allows one to concentrate on the properties that are of interest to the application. There are two main abstraction mechanism used to model information:

Generalization and specialization:

Generalization is the abstracting process of viewing set of objects as a single general class by concentrating on the general characteristics of the constituent sets while suppressing or ignoring their differences. It is the union of a number of lower-level entity types for the purpose of producing a higher-level entity type. For instance, student is a generalization of graduate or undergraduate, full-time or part-time students. Similarly, employee is generalization of the classes of objects cook, waiter, and cashier. Generalization is an IS_A relationship; therefore, manager IS_AN employee, cook IS_AN employee, waiter IS_AN employee, and so forth.

Specialization is the abstracting process of introducing new characteristics to an existing class of objects to create one or more new classes of objects. This involves taking a higher-level, and using additional characteristics, generating lower-level entities. The lower-level entities also inherits the, characteristics of the higher-level entity. In applying the characteristics size to car we can create a full-size, mid-size, compact or subcompact car. Specialization may be seen as the reverse process of generalization addition specific properties are introduced at a lower level in a hierarchy of objects.



EMPLOYEE(**empno**,name,dob)
FULL_TIME_EMPLOYEE(**empno**,salary)
PART_TIME_EMPLOYEE(**empno**,type)

Faculty(**empno**,degree,intrest)
Staff(**empno**,hour-rate)
Teaching (**empno**,stipend)

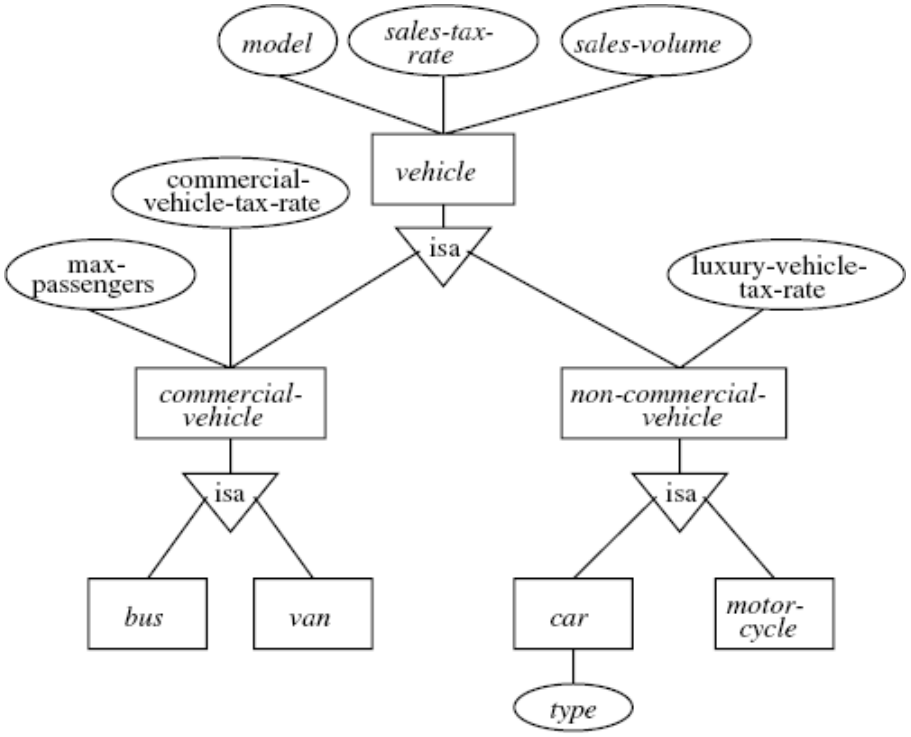
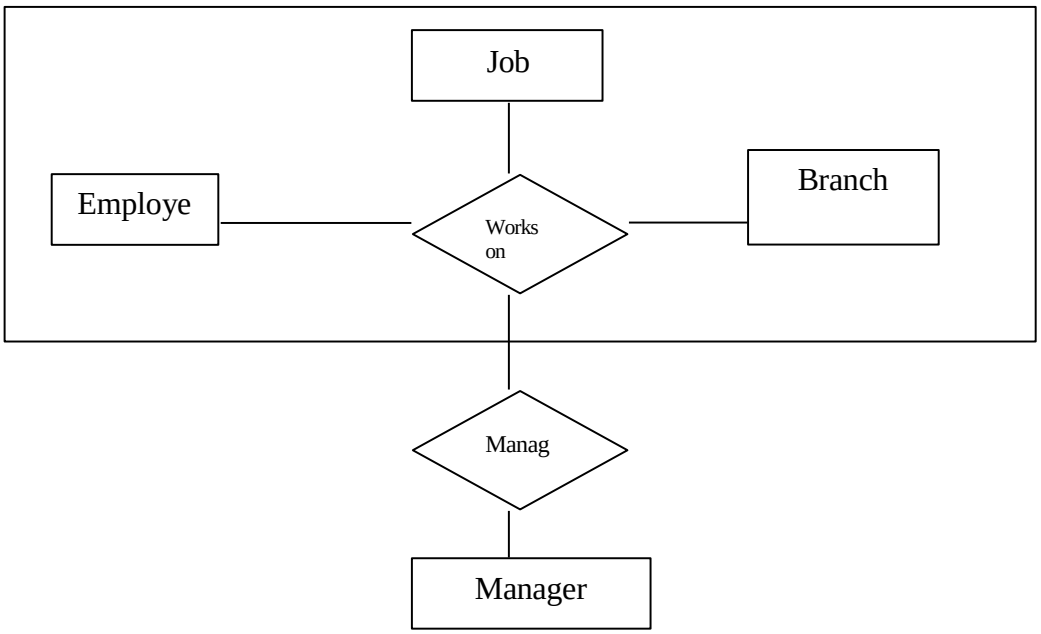


Figure 2.19 E-R diagram of motor-vehicle sales company.

Aggregation:

Aggregation is the process of compiling information on an object, there by abstracting a higher level object. The entity person is derived by aggregating the characteristics of name, address, ssn. Another form of the aggregation is abstracting a relationship objects and viewing the relationship as an object.



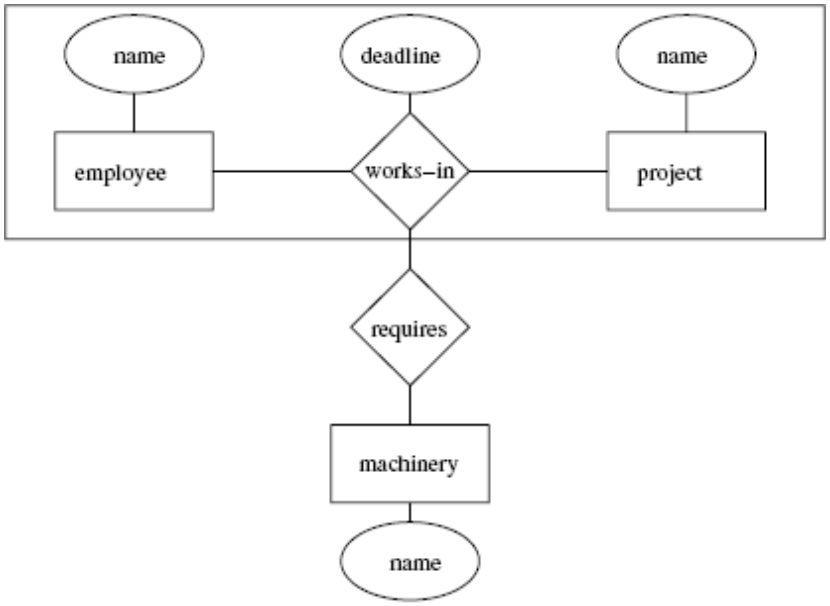


Figure 2.8 E-R diagram Example 1 of aggregation.

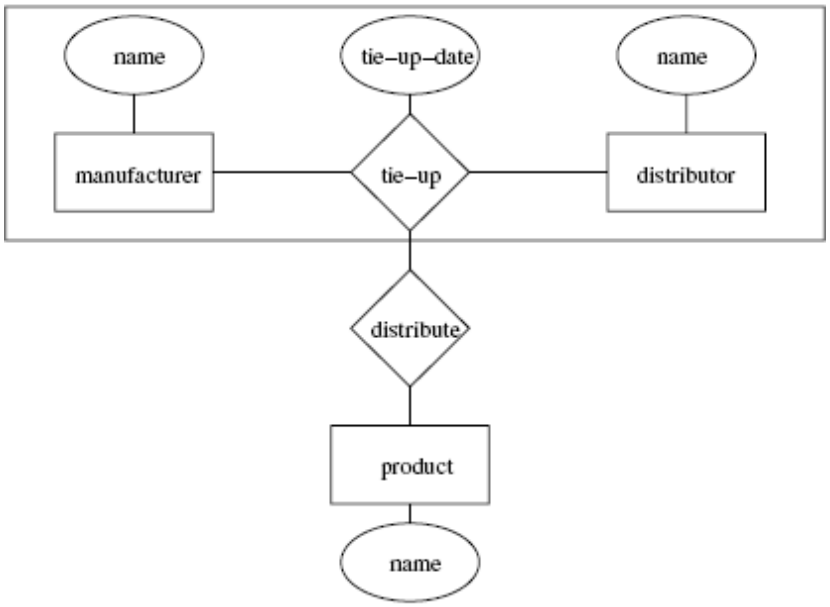
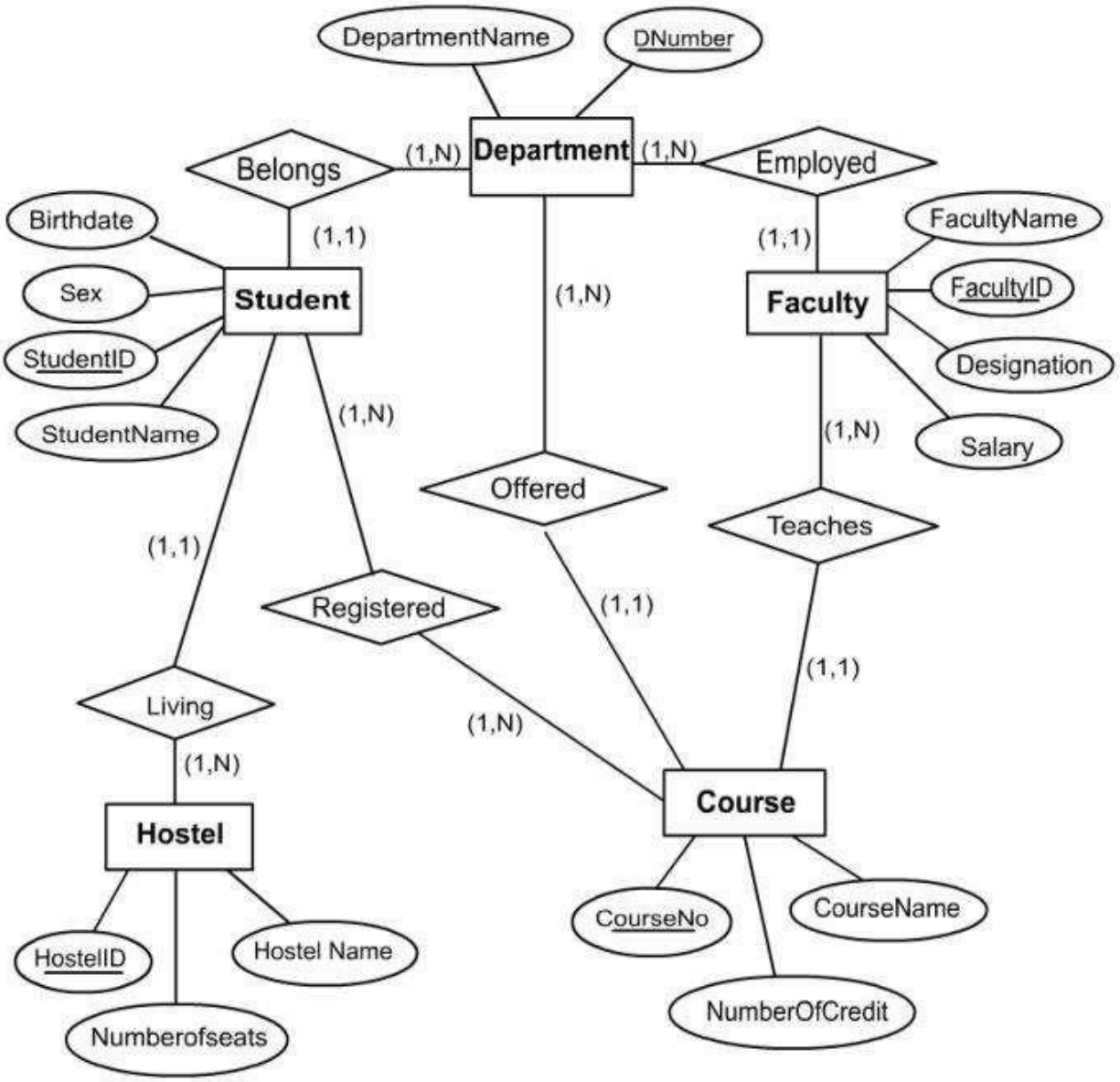


Figure 2.9 E-R diagram Example 2 of aggregation.

ER- Diagram For College Database



LECTURE-8: Conversion of ER-Diagram to Relational Database

Conversion of Entity Sets:

1. For each strong entity type E in the ER diagram, we create a relation R containing all the single attributes of E. The primary key of the relation R will be one of the key attribute of R.

   STUDENT(rollno (primary key),name, address)
   FACULTY(id(primary key),name ,address, salary)
   COURSE(course-id,(primary key),course_name,duration)
   DEPARTMENT(dno(primary key),dname)

2. For each weak entity type W in the ER diagram, we create another relation R that contains all simple attributes of W. If E is an owner entity of W then key attribute of E is also include In R. This key attribute of R is set as a foreign key attribute of R. Now the combination of primary key attribute of owner entity type and partial key of the weak entity type will form the key of the weak entity type

   GUARDIAN((rollno,name) (primary key),address,relationship)

Conversion of Relationship Sets:

Binary Relationships:

- **One-to-One Relationship:**
  For each 1:1 relationship type R in the ER-diagram involving two entities E1 and E2 we choose one of entities(say E1) preferably with total participation and add primary key attribute of another E as a foreign key attribute in the table of entity(E1). We will also include all the simple attributes of relationship type R in E1 if any, For example, the department relationship has been extended tp include head-id and attribute of the relationship.
  DEPARTMENT(D_NO,D_NAME,HEAD_ID,DATE_FROM)

- **One-to-Many Relationship:**
  For each 1:N relationship type R involving two entities E1 and E2, we identify the entity type (say E1) at the N-side of the relationship type R and include primary key of the entity on the other side of the relation (say E2) as a foreign key attribute in the table of E1. We include all simple attribute (or simple components of a composite attribute of R (if any) in the table E1)

  For example:
  The works in relationship between the DEPARTMENT and FACULTY. For this relationship choose the entity at N side, i.e, FACULTY and add primary key attribute of another entity DEPARTMENT i.e., DNO as a foreign key attribute in FACULTY.

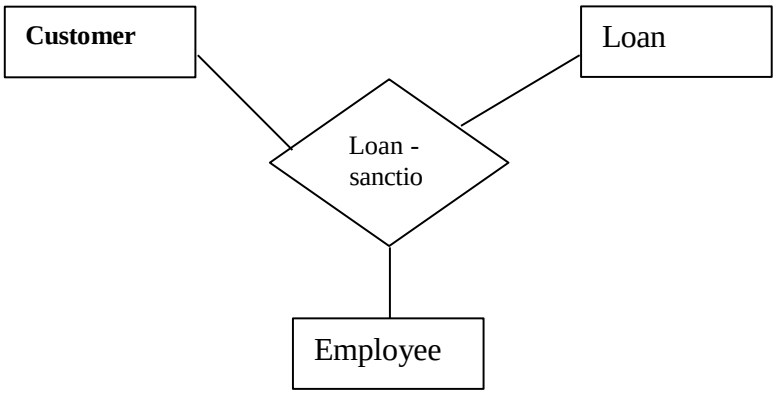
  FACULTY(CONTAINS WORKS_IN RELATIOSHIP)
  (ID, NAME, ADDRESS, BASIC_SAL, DNO)

- **Many-to-Many Relationship:**
  For each M:N relationship type R, we create a new table (say S) to represent R, we also include the primary key attributes of both the participating entity types as a foreign key attribute in S. Any simple attributes of the M:N relationship type (or simple components as a composite attribute) is also included as attributes of S.

For example:
The M:N relationship taught-by between entities COURSE and FACULTY should be represented as a new table. The structure of the table will include primary key of COURSE and primary key of FACULTY entities.

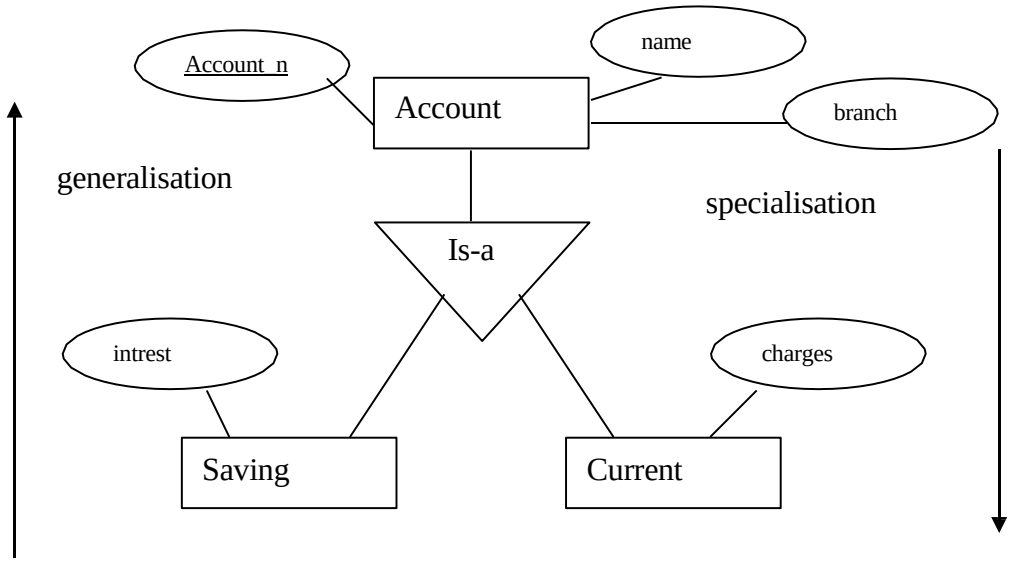
TAUGHT-BY (ID (primary key of FACULTY table), course-id (primary key of COURSE table)

- **N-ary Relationship:**
For each N-ary relationship type R where n>2, we create a new table S to represent R, We include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. We also include any simple attributes of the N-ary relationship type (or simple components of complete attribute) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types.



LOAN-SANCTION (cusomer-id, loanno, empno, sancdate, loan_amount)

- **Multi-Valued Attributes:**
For each multivalued attribute 'A', we create a new relation R that includes an attribute corresponding to plus the primary key attributes k of the relation that represents the entity type or relationship that has as an attribute. The primary key of R is then combination of A and k.
For example, if a STUDENT entity has rollno, name and phone number where phone number is a multivalued attribute then we will create table PHONE (rollno, phoneno) where primary key is the combination. In the STUDENT table we need not have phone number, instead if can be simply (rollno, name) only.
PHONE(rollno, phoneno)



- **Converting Generalisation /Specification Hierarchy to Tables:**
  A simple rule for conversion may be to decompose all the specialized entities into table in case they are disjoint, for example, for the figure we can create the three tables as:
  Account (account_no, name, branch, balance)
  Saving_Account (account-no, intrest)
  Current_Account (account-no, charges)

LECTURE-9: Record Based Logical Model

Hierarchical Model:

- A hierarchical database consists of a collection of *records* which are connected to one another through *links.*
- A record is a collection of fields, each of which contains only one data value.
- A link is an association between precisely two records.
- The hierarchical model differs from the network model in that the records are organized as collections of trees rather than as arbitrary graphs.

Tree-Structure Diagrams:

- The schema for a hierarchical database consists of
  - *boxes,* which correspond to record types
  - *lines,* which correspond to links
- Record types are organized in the form of a *rooted tree.*
  - No cycles in the underlying graph.
  - Relationships formed in the graph must be such that only one-to-many or one-to-one relationships exist between a parent and a child.

Database schema is represented as a collection of tree-structure diagrams.

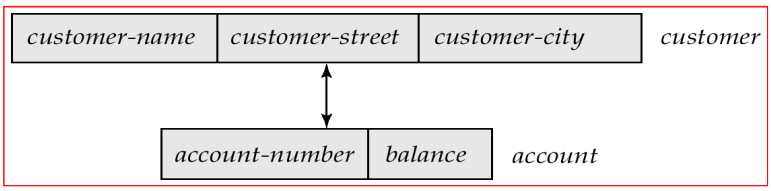
- *single* instance of a database tree
- The root of this tree is a dummy node
- The children of that node are actual instances of the appropriate record type

When transforming E-R diagrams to corresponding tree-structure diagrams, we must ensure that the resulting diagrams are in the form of rooted trees.

Single Relationships:

- Example of E-R diagram with two entity sets, *customer* and *account,* related through a binary, one-to-many relationship *depositor.*
- Corresponding tree-structure diagram has
  - the record type *customer* with three fields: *customer-name, customer-street,* and *customer-city.*
  - the record type *account* with two fields: *account-number* and *balance*
  - the link *depositor,* with an arrow pointing to *customer*

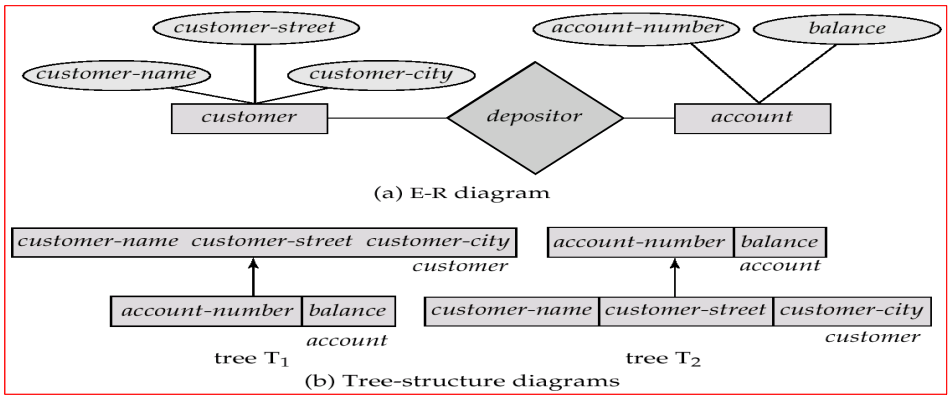
- If the relationship *depositor* is one to one, then the link *depositor* has two arrows.



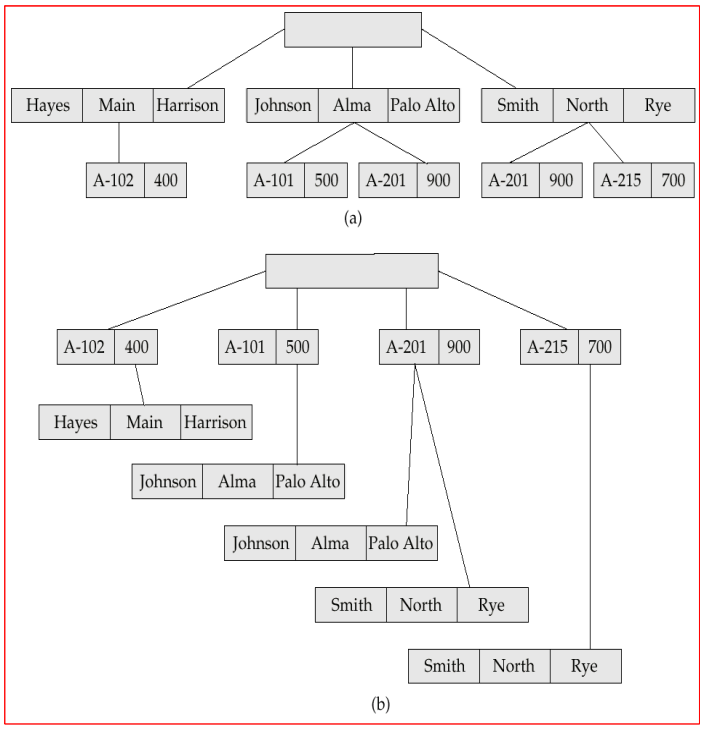
- Only one-to-many and one-to-one relationships can be directly represented in the hierarchical mode.

Transforming Many-To-Many Relationships:

- Must consider the type of queries expected and the degree to which the database schema fits the given E-R diagram.
- In all versions of this transformation, the underlying database tree (or trees) will have replicated records.



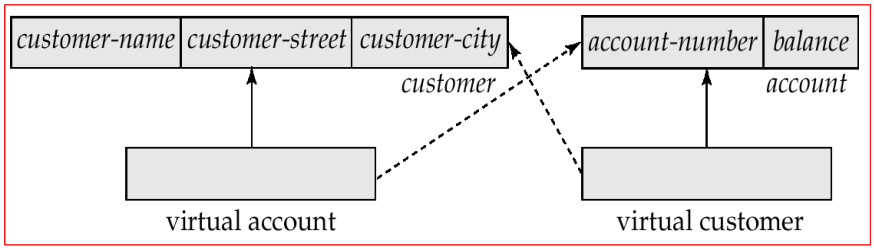
- Create two tree-structure diagrams, *T*1, with the root *customer*, and *T*2, with the root *account.*
- In *T*1, create *depositor*, a many-to-one link from *account* to *customer.*
- In *T*2, create *account-customer*, a many-to-one link from *customer* to *account.*



Virtual Records:

- For many-to-many relationships, record replication is necessary to preserve the tree-structure organization of the database.
- Data inconsistency may result when updating takes place
- Waste of space is unavoidable

- *Virtual record* — contains no data value, only a logical pointer to a particular physical record.
- When a record is to be replicated in several database trees, a single copy of that record is kept in one of the trees and all other records are replaced with a virtual record.
- Let *R* be a record type that is replicated in *T*1, *T*2, . . ., *Tn*. Create a new virtual record type *virtual-R* and replace *R* in each of the *n* – 1 trees with a record of type *virtual-R.*
- Eliminate data replication in the following diagram ; create *virtual-customer* and *virtual-account.*
- Replace *account* with *virtual-account* in the first tree, and replace *customer* with *virtual-customer* in the second tree.
- Add a dashed line from *virtual-customer* to *customer*, and from *virtual-account* to *account*, to specify the association between a virtual record and its corresponding physical record.



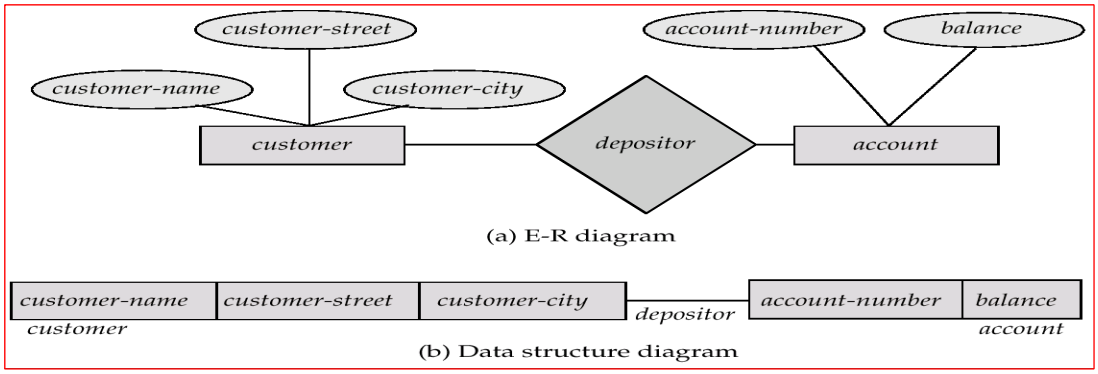
Network Model:

- Data are represented by collections of *records*.
  - similar to an entity in the E-R model
  - Records and their fields are represented as *record type*
- type *customer* = record
  customer-name: string;
  customer-street: string;
  customer-city: string;
  type *account* = record type
  account-number: integer;
  balance: integer;
- end
  end
- Relationships among data are represented by *links*
  - similar to a restricted (binary) form of an E-R relationship
  - restrictions on links depend on whether the relationship is many-to-many, many-to-one, or one-to-one.

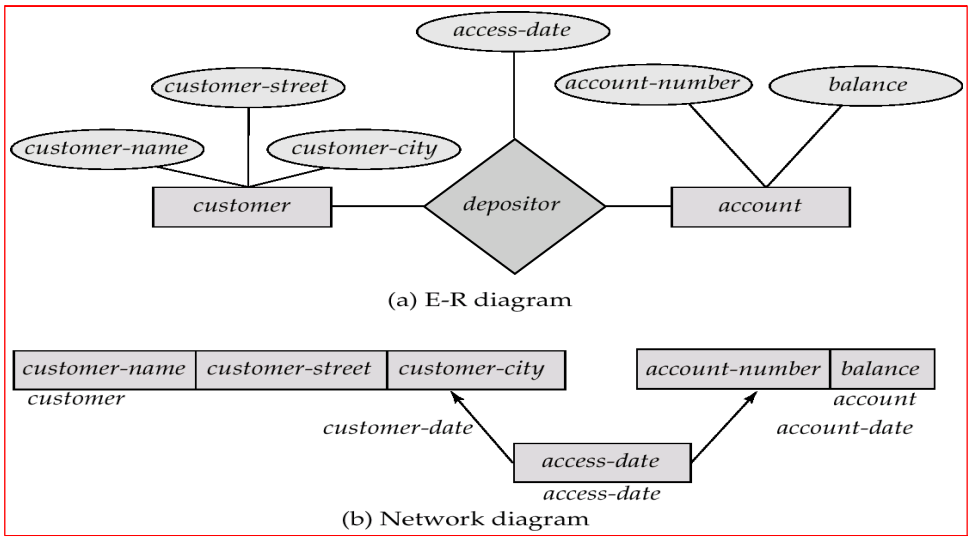
Data-Structure Diagrams:

- Schema representing the design of a network database.
- A data-structure diagram consists of two basic components:
  - Boxes, which correspond to record types.
  - Lines, which correspond to links.
- Specifies the overall logical structure of the database.

For every E-R diagram, there is a corresponding data-structure diagram.

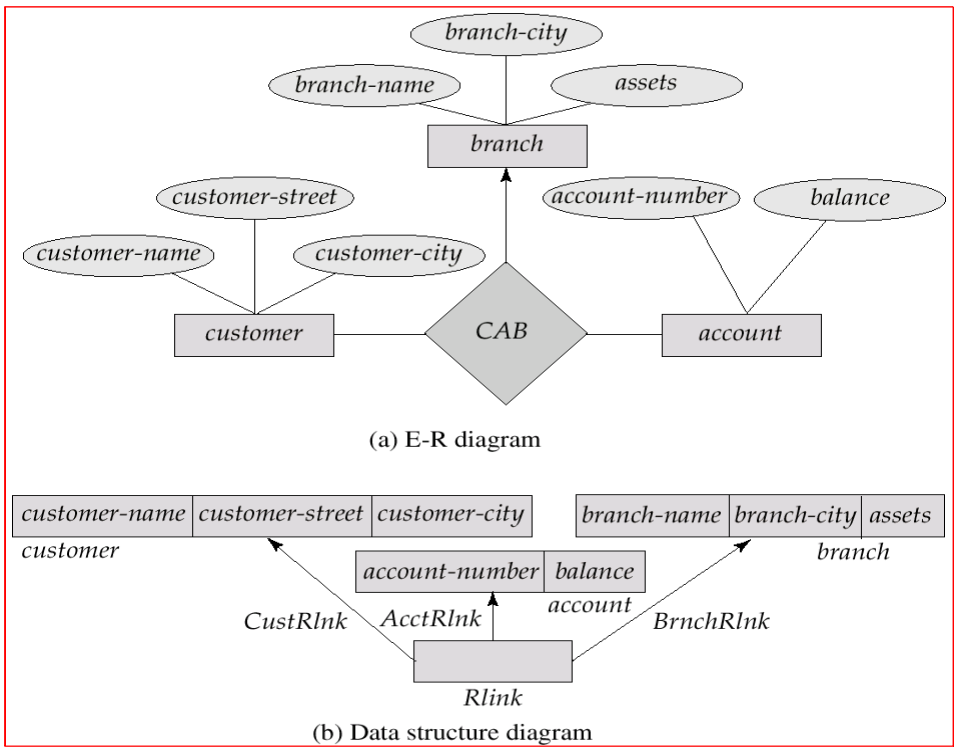


Since a link cannot contain any data value, represent an E-R relationship with attributes with a new record type and links**.**



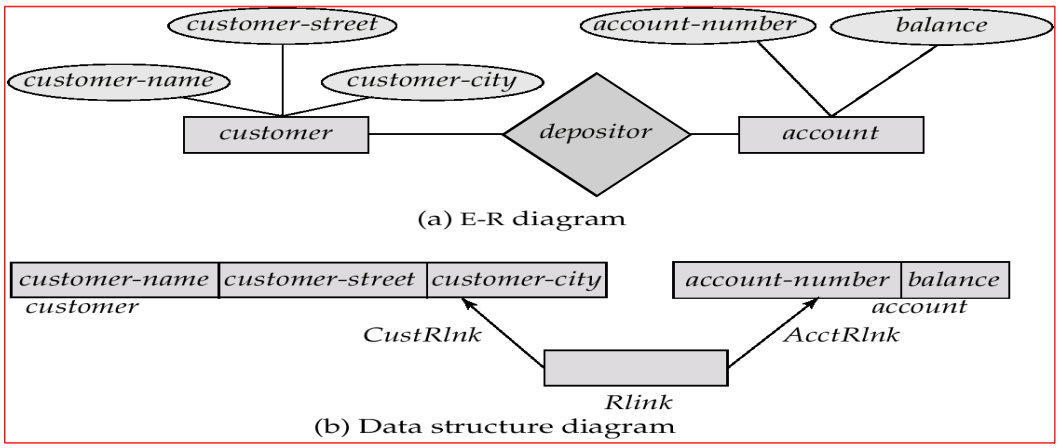
To represent an E-R relationship of degree 3 or higher, connect the participating record types through a new record type that is linked directly to each of the original record types.

1. Replace entity sets *account*, *customer*, and *branch* with record types *account, customer*, and *branch*, respectively.
2. Create a new record type *Rlink* (referred to as a *dummy* record type).
3. Create the following many-to-one links:
    - *CustRlink* from *Rlink* record type to *customer* record type
    - *AcctRlnk* from *Rlink* record type to *account* record type
    - *BrncRlnk* from *Rlink* record type to *branch* record type



The DBTG CODASYL Model:

- All links are treated as many-to-one relationships.
- To model many-to-many relationships, a record type is defined to represent the relationship and two links are used.



DBTG Sets:

- The structure consisting of two record types that are linked together is referred to in the DBTG model as a *DBTG set*
- In each DBTG set, one record type is designated as the *owner*, and the other is designated as the *member*, of the set.
- Each DBTG set can have any number of *set occurrences* (actual instances of linked records).
- Since many-to-many links are disallowed, each set occurrence has precisely one owner, and has zero or more member records.
- No member record of a set can participate in more than one occurrence of the set at any point.
- A member record can participate simultaneously in several set occurrences of *different* DBTG sets.

LECTURE-10: RELATIONAL MODEL

RELATIONAL MODEL

Relational model is simple model in which database is represented as a collection of "relations" where each relation is represented by two-dimensional table.

account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

The relational model was founded by E. F. Codd of the IBM in 1972. The basic concept in the relational model is that of a relation.

Properties:

- It is column homogeneous. In other words, in any given column of a table, all items are of the same kind.
- Each item is a simple number or a character string. That is a table must be in first normal form.
- All rows of a table are distinct.
- The ordering of rows with in a table is immaterial.
- The column of a table are assigned distinct names and the ordering of these columns is immaterial.

Domain, attributes tuples and relational:

Tuple:

Each row in a table represents a record and is called a tuple .A table containing 'n' attributes in a record is called is called n-tuple.

Attributes:

The name of each column in a table is used to interpret its meaning and is called an attribute.Each table is called a relation. In the above table, account_number, branch name, balance are the attributes.

Domain:

A domain is a set of values that can be given to an attributes. So every attribute in a table has a specific domain. Values to these attributes can not be assigned outside their domains.

Relation:

A relation consist of

- **Relational schema**
- **Relation instance**

Relational Schema:

A relational schema specifies the relation's name, its attributes and the domain of each attribute. If

R is the name of a relation and A1, A2,…An is a list of attributes representing R then R(A1,A2,…,An) is called a Relational Schema. Each attribute in this relational schema takes a value from some specific domain called domain(Ai).

Example:

PERSON (PERSON_ID:INTEGER, NAME:STRING, AGE:INTEGER, ADDRESS:STRING)

Total number of attributes in a relation denotes the degree of a relation since the PERSON relation scheme contains four attributes, so this relation is of degree 4.

Relation Instance:

A relational instance denoted as r is a collection of tuples for a given relational schema at a specific point of time.

A relation state r to the relations schema R(A1, A2…, An) also denoted by r(R) is a set of n-tuples

R{t1,t2,…tm}

Where each n-tuple is an ordered list of n values

T=<v1,v2,….vn>

Where each vi belongs to domain (Ai) or contains null values.

The relation schema is also called 'intension' and the relation state is also called 'extension'.

Eg: Relation schema for Student

STUDENT(rollno:string, name:string, city:string, age:integer)

Relation instance:

Student:

Rollno	Name	City	Age
101	Sujit	Bam	23
102	kunal	bbsr	22

Keys:

Super key:

A super key is an attribute or a set of attributes used to identify the records uniquely in a relation.

For example, customer-id, (cname, customer-id), (cname,telno)

Candidate key:

Super keys of a relation can contain extra attributes. Candidate keys are minimal super keys. i.e, such a key contains no extraneous attribute. An attribute is called extraneous if even after removing it from the key, makes the remaining attributes still has the properties of a key(atribute represents entire table).

In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following properties:

- *Uniqueness:* no two distinct tuples in R have the same values for the candidate key
- *Irreducible:* No proper subset of the candidate key has the *uniqueness property that is the candidate key.*
- *A candidate key's values must exist. It can't be null.*
- *The values of a candidate key must be stable. Its value can not change outside the*

control of the system.

Eg: (cname,telno)

Primary key:

The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities with in an entity set. The remaining candidate keys if any are called ***alternate key*.**

LECTURE-11: CONSTRAINTS

RELATIONAL CONSTRAINTS:

There are three types of constraints on relational database that include

- DOMAIN CONSTRAINTS
- KEY CONSTRAINTS
- INTEGRITY CONSTRAINTS

DOMAIN CONSTRAINTS:

It specifies that each attribute in a relation an atomic value from the corresponding domains. The data types associated with commercial RDBMS domains include:

- Standard numeric data types for integer
- Real numbers
- Characters
- Fixed length strings and variable length strings

Thus, domain constraints specifies the condition that we to put on each instance of the relation. So the values that appear in each column must be drawn from the domain associated with that column.

Rollno	Name	City	Age
101	Sujit	Bam	23
102	kunal	bbsr	22

Key Constraints:

This constraints states that the key attribute value in each tuple msut be unique .i.e, no two tuples contain the same value for the key attribute.(null values can allowed)
Emp(empcode,name,address) . here empcode can be unique

Integrity CONSTRAINTS:

There are two types of integrity constraints:

- Entity Integrity Constraints
- Referential Integrity constraints

Entity Integrity Constraints:

It states that no primary key value can be null and unique. This is because the primary key is used to identify individual tuple in the relation. So we will not be able to identify the records uniquely containing null values for the primary key attributes. This constraint is specified on one individual relation.

Referential Integrity Constraints:

It states that the tuple in one relation that refers to another relation must refer to an existing tuple in that relation. This constraints is specified on two relations. If a column is declared as foreign key that must be primary key of another table.

Department (deptcode, dname)
Here the deptcode is the primary key.

Emp (*empcode*, name, city, *deptcode*).
Here the deptcode is foreign key.

CODD'S RULES

Rule 1 : The information Rule.

"All information in a relational data base is represented explicitly at the logical level and in exactly one way - by values in tables."

Everything within the database exists in tables and is accessed via table access routines.

Rule 2 : Guaranteed access Rule.

"Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name."

To access any data-item you specify which column within which table it exists, there is no reading of characters 10 to 20 of a 255 byte string.

Rule 3 : Systematic treatment of null values.

"Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type."

If data does not exist or does not apply then a value of NULL is applied, this is understood by the RDBMS as meaning non-applicable data.

Rule 4 : Dynamic on-line catalog based on the relational model.

"The data base description is represented at the logical level in the same way as-ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data."

The Data Dictionary is held within the RDBMS, thus there is no-need for off-line volumes to tell you the structure of the database.

Rule 5 : Comprehensive data sub-language Rule.

"A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all the following items

- Data Definition
- View Definition
- Data Manipulation (Interactive and by program).
- Integrity Constraints
- Authorization.

Every RDBMS should provide a language to allow the user to query the contents of the RDBMS and also manipulate the contents of the RDBMS.

Rule 6 : .View updating Rule

"All views that are theoretically updateable are also updateable by the system."

Not only can the user modify data, but so can the RDBMS when the user is not logged-in.

Rule 7 : High-level insert, update and delete.

"The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data."

The user should be able to modify several tables by modifying the view to which they act as base tables.

Rule 8 : Physical data independence.

"Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods."

The user should not be aware of where or upon which media data-files are stored

Rule 9 : Logical data independence.

"Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit un-impairment are made to the base tables."

User programs and the user should not be aware of any changes to the structure of the tables (such as the addition of extra columns).

Rule 10 : Integrity independence.

"Integrity constraints specific to a particular relational data base must be definable in the relational data sub-language and storable in the catalog, not in the application programs."

If a column only accepts certain values, then it is the RDBMS which enforces these constraints and not the user program, this means that an invalid value can never be entered into this column, whilst if the constraints were enforced via programs there is always a chance that a buggy program might allow incorrect values into the system.

Rule 11 : Distribution independence.

"A relational DBMS has distribution independence."

The RDBMS may spread across more than one system and across several networks, however to the end-user the tables should appear no different to those that are local.

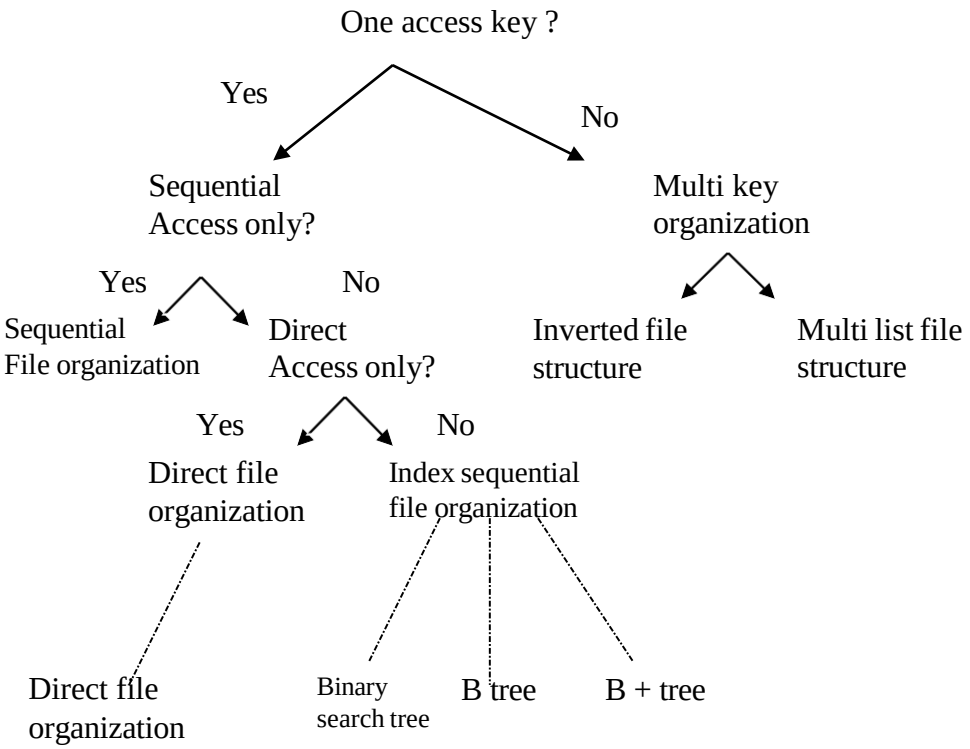
Rule 12 : Non-subversion Rule.

"If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity Rules and constraints expressed in the higher level relational language (multiple-records-at-a-time)."

LECTURE-12: FILE ORGANISATION

FILE ORGANISATION AND ITS TYPES:

A file organization is a technique to organize data in the secondary memory. File organization is a way of arranging the records in a file when the file is stored on the disk. Data files are organized so as to facilitate access to records and to ensure their efficient storage. A DBMS supports several file organization techniques.



(File organization techniques)

Heap files (unordered file)

Basically these files are unordered file. It is the simplest and most basic type. These files consist of randomly ordered records. The records will have no particular order.The operations we can perform on the records are insert , retrieve and delete. The features of the heap file organization are:

- New records can be inserted in any empty space that can accommodate them.
- When old records are deleted, the occupied space becomes empty and available for any new insertion.
- If updated records grow, they may need to be relocated to a new empty space. This needs to keep a list of empty space.

Advantage of heap files:

1. This is a simple file organization method
2. Insertion is somehow efficient
3. Good for bulk-lading data into a table.
4. Best if file scans are common or insertions are frequent

Disadvantages of heap files:

1. Retrieval requires a linear search and is inefficient

2. Deletion can result in unused space/need for reorganization

Sequential file organization:

The most basic way to organize the collection of records in a file is to use sequential organization. Records of the file are stored in sequence by the primary key field values/ They are accessible only in the order stored i.e, in the primary key order. This kind is of file organization works well for tasks which need to access nearly every record in a file. Eg. Payroll..

In a sequentially organized file records are written consecutively when the file is created and must be accessed consecutively when the file later used for input.

A sequential file maintains the records in the logical sequence of its primary key values. Sequential file are inefficient for random access. And files can be stored on devices like magnetic tape that allow sequential access. As records are in sorted order it will use binary search technique to search for a record.

Advantages of sequential file organization:

- It is fast and efficient when dealing with large volumes of data that need to be processed periodically(batch system)

Disadvantages of sequential file organization:

- Requires that all new transactions be sorted into the proper sequence for sequential access processing
- Locating, storing, modifying, deleting or adding records in the file require rearranging the file/
- This method is too slow to handle application requiring immediate updating or responses.

Indexed sequential file organization:

It organized the file like a large dictionary, i.e, records are stored in order of the key, but an index is kept which also permits a type of direct access. The records are stored sequentially by primary key values and there is an index built over the primary key field.

An index is a set of index value, address pairs. Indexing associates a set of objects to a set pf orderable quantities, that are usually smaller in number or their properties. Thus an index is a mechanism for faster search. Although the indices and the data blocks are kept physically, they are logically distinct.

A sequential file that is indexed on its primary key is called an index sequential file. The index allows for random access to records, while the sequential storage of the records of the file provides easy access to the sequential records. An addition feature of this file system is the overflow area. The overflow area provides additional space for record addition without the need to create.

Advantage of ISAM indexes:

1. Because the whole structure is ordered to a large extent, partial (like ty%) and range(between 12 and 29) based retrievals can often benefit from the use of this type of index.
2. ISAM is good for static tables because there are usually fewer index levels than B-tree
3. In general there are fewer disk I/Os required to access data, provided there is no overflow.

Disadvantage of ISAM indexes:

1. ISAM is still not as quick as some in hash file organization
2. Overflow can be a real problem in highly volatile table.

Hashed file organization:

Hashing is the most common form of purely random access to a file or database. It is also used to access columns that do not have a index as an optimization technique. Hash functions calculate the address of the page in which the record is to be stored based on one or more fields in the record. The records in a hash file appear randomly distributed across the available space. It requires some hashing algorithm and the technique. Hashing algorithm converts a primary key value into a record address.

Advantage of hashed file organization:

1. Insertion or search on hash key is fast.
2. Best if equality search is needed on hash key

Disadvantage of hashed file organization:

1. It is a complex file organization method
2. search is slow
3. It suffers from disk space overhead
4. Unbalanced buckets degrade performance
5. Range search is slow

LECTURE-13: INDEX

Types of Indexes:

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- **Search Key** - attribute to set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form
- Index files are typically much smaller than the original file
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order
  - **Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function".

Ordered Indices

Indexing techniques evaluated on basis of:

1. In an **ordered index,** index entries are stored sorted on the search key value. E.g., author catalog in library.
2. **Primary index:** in a sequentially ordered file, the index whose search key specifies the sequential order of the file.
   a. Also called **clustering index**
   b. The search key of a primary index is usually but not necessarily the primary key.
3. **Secondary index**: an index whose search key specifies an order different from the sequential order of the file. Also called non-clustering index**.**
4. Index-sequential file**:** ordered sequential file with a primary index.

There are two types of ordered indices that we can use:

Dense index:

An index record appears for every search-key value in the file. In a dense primary index, the index record contains the search-key value and a pointer to the first data record with that search-key value. The rest of the records with the same search key-value would be stored sequentially after the first record, since, because the index is a primary one, records are sorted on the same search key. Dense index implementations may store a list of pointers to all records with the same search-key value; doing so is not essential for primary indices.

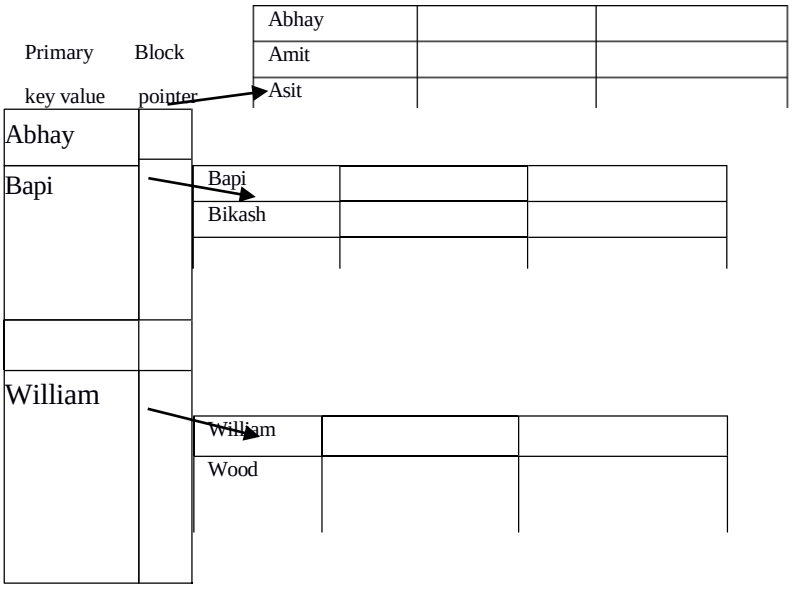
Sparse index:

An index record appears for only some of the search-key values. As is true in dense indices, each index record contains a search-key value and a pointer to the first data record with that search-key value. To locate a record, we find the index entry with the largest search-key value that is less than or equal to the search-key value for which we are looking. We start at the record pointed to by that index entry, and follow the pointers in the file until we find the desired record.

Primary Indexes:

A primary index is an ordered file whose records are of fixed length with two fields. The first field is of the same data types as the ordering key filed of the data file and the second field is a pointer to a disk block- a block address. The ordering key field is called the primary key of the data file. There

is one index entry in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer to other block as its two fields values.
The first record of each block of the data file is known as anchor record of the block.
Primary index is an example of non dense index



(Primary indexes on the ordering key field)

A major problem with primary index as with any ordered file- is insertion and deletion of records. With a primary Index, the problem is compounded because if we attempt to insert a record in its correct position in the data file, we not only have to move records to make space for the new record but also have to change some index entries because moving records will change the anchor records of some blocks.

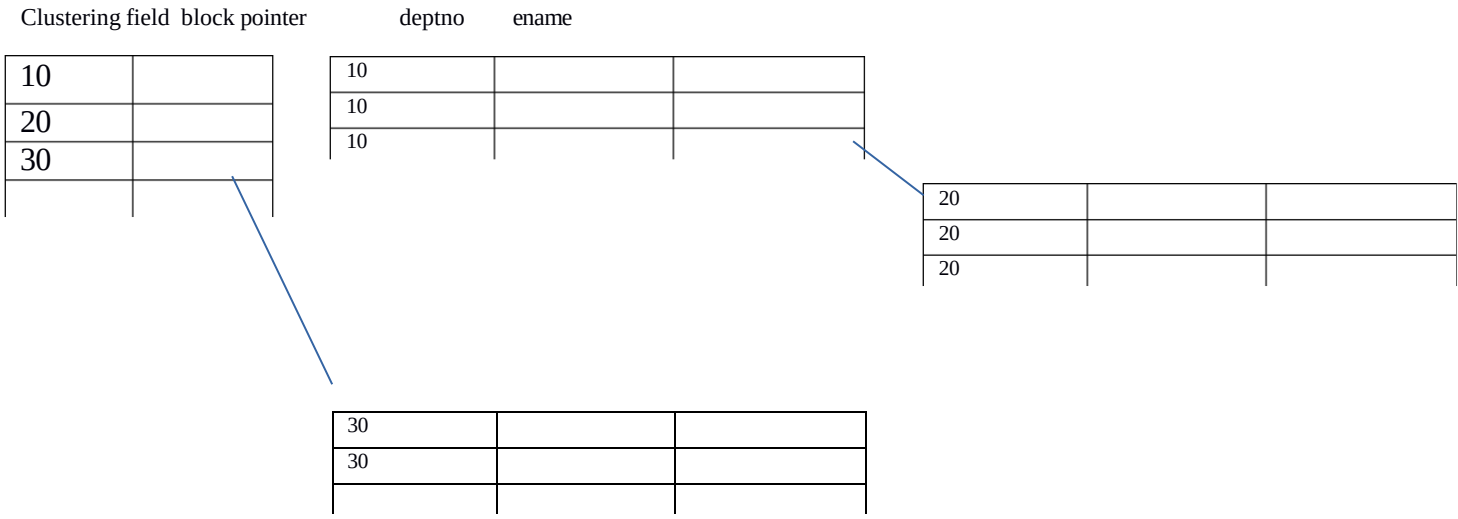
LECTURE-14: Clustering Index

Clustering Indexes:

If the records of a file are physically ordered on a non key field that does not have a distinct value for each record, that filed is called the clustering filed of the file. We can create a different type of index called clustering index to speed up retrieval of records that have the same value for the clustering field. This differs from a primary index, which requires that the ordering field of the data file have a distinct value for each record.

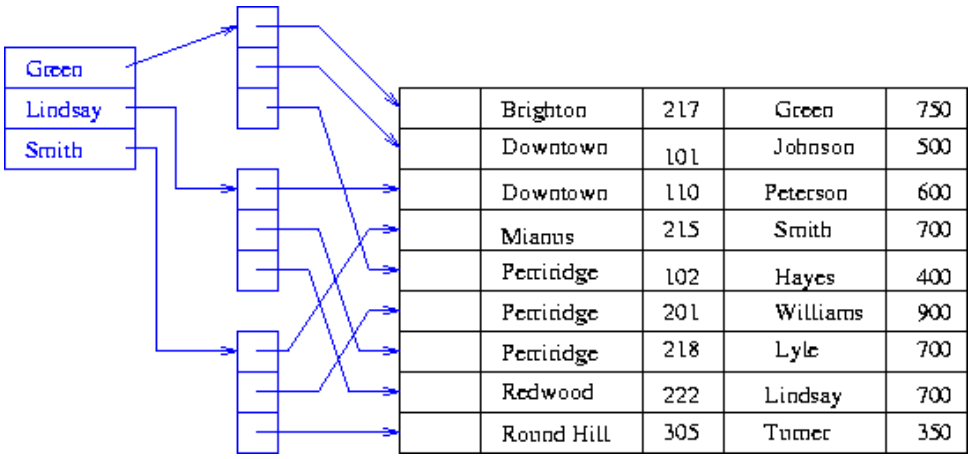
A clustering index is also an ordered file with two fields, the first field is of the same type as the clustering field of the data file and the second field is block pointer. There is one entry in the clustering index for each distinct value of the clustering field, containing that value and a pointer to the first block in the data file that has a record with that value for its clustering field.

A clustering index is another example of non dense index.



Secondary Indexes:

A secondary index also is an ordered file with two fields and as in the other indexes, the second filed is a pointer to a disk block. The first field is of the same type as some non ordering field of the data file. The field on which the secondary index is constructed is called an indexing field of the file. Whether its values are distinct for every record or not.



	Brighton	217	Green	750
	Downtown	101	Johnson	500
	Downtown	110	Peterson	600
	Mianus	215	Smith	700
	Perryridge	102	Hayes	400
	Perryridge	201	Williams	900
	Perryridge	218	Lyle	700
	Redwood	222	Lindsay	700
	Round Hill	305	Turner	350

There can be many secondary indexes and hence indexing fields for the same file.

Primary and Secondary Indices:

Secondary indices have to be dense.

1. Indices offer substantial benefits when searching for records.
2. When a file is modified, every index on the file must be updated, Updating indices imposes overhead on database modification.
3. Sequential scan using primary index is efficient, but a sequential scan using a secondary index is expensive
    a. each record access may fetch a new block from disk

Used in query representation used in parsing.

Query Optimization:

A single query can be executed through different algorithms or re-written in different forms and structures. Hence, the question of query optimization comes into the picture – Which of these forms or pathways is the most optimal? The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.

There are broadly two ways a query can be optimized:

1. Analyze and transform equivalent relational expressions: Try to minimize the tuple and column counts of the intermediate and final query processes (discussed here).
2. Using different algorithms for each operation: These underlying algorithms determine how tuples are accessed from the data structures they are stored in, indexing, hashing, data retrieval and hence influence the number of disk and block accesses (discussed in query processing).

Analyze and transform equivalent relational expressions

Here, we shall talk about generating minimal equivalent expressions. To analyze equivalent expression, listed are a set of equivalence rules. These generate equivalent expressions for a query written in relational algebra. To optimize a query, we must convert the query into its equivalent form as long as an equivalence rule is satisfied.

1. **Conjunctive selection operations can be written as a sequence of individual selections. This is called a sigma-cascade.**

   Explanation: Applying condition intersection is expensive. Instead, filter out tuples satisfying condition (inner selection) and then apply condition (outer selection) to the then resulting fewer tuples. This leaves us with less tuples to process the second time. This can be extended for two or more intersecting selections. Since we are breaking a single condition into a series of selections or cascades, it is called a "cascade".

2. **Selection is commutative.**

   Explanation: condition is commutative in nature. This means, it does not matter whether we apply first or first. In practice, it is better and more optimal to apply that selection first which yields a fewer number of tuples. This saves time on our outer selection.

3. **All following projections can be omitted, only the first projection is required. This is called a pi-cascade.**

   Explanation: A cascade or a series of projections is meaningless. This is because in the end, we are only selecting those columns which are specified in the last, or the outermost projection. Hence, it is better to collapse all the projections into just one i.e. the outermost projection.

4. **Selections on Cartesian Products can be re-written as Theta Joins.**

   - **Equivalence 1**

   Explanation: The cross product operation is known to be very expensive. This is because it matches each tuple of E1 (total m tuples) with each tuple of E2 (total n tuples). This yields m*n entries. If we apply a selection operation after that, we would have to scan through m*n entries to find the suitable tuples which satisfy the condition . Instead of doing all of this, it is more optimal to use the Theta Join, a join specifically designed to select only those entries in the cross product which satisfy the Theta condition, without evaluating the entire cross product first.

   - **Equivalence 2**

   Explanation: Theta Join radically decreases the number of resulting tuples, so if we apply an intersection of both the join conditions i.e. and into the Theta Join itself,

we get fewer scans to do. On the other hand, a condition outside unnecessarily increases the tuples to scan.

5. **Theta Joins are commutative.**

Explanation: Theta Joins are commutative, and the query processing time depends to some extent which table is used as the outer loop and which one is used as the inner loop during the join process (based on the indexing structures and blocks).

6. **Join operations are associative.**

- **Natural Join**

Explanation: Joins are all commutative as well as associative, so one must join those two tables first which yield less number of entries, and then apply the other join.

- **Theta Join**

Explanation: Theta Joins are associative in the above manner, where involves attributes from only E2 and E3.

7. **Selection operation can be distributed.**

- **Equivalence 1**

Explanation: Applying a selection after doing the Theta Join causes all the tuples returned by the Theta Join to be monitored after the join. If this selection contains attributes from only E1, it is better to apply this selection to E1 (hence resulting in a fewer number of tuples) and then join it with E2.

- **Equivalence 2**

Explanation: This can be extended to two selection conditions, and , where Theta1 contains the attributes of only E1 and contains attributes of only E2. Hence, we can individually apply the selection criteria before joining, to drastically reduce the number of tuples joined.

8. **Projection distributes over the Theta Join.**

- **Equivalence 1**

Explanation: The idea discussed for selection can be used for projection as well. Here, if L1 is a projection that involves columns of only E1, and L2 another projection that involves the columns of only E2, then it is better to individually apply the projections on both the tables before joining. This leaves us with a fewer number of columns on either side, hence contributing to an easier join.

- **Equivalence 2**

Explanation: Here, when applying projections L1 and L2 on the join, where L1 contains columns of only E1 and L2 contains columns of only E2, we can introduce another column E3 (which is common between both the tables). Then, we can apply projections L1 and L2 on E1 and E2 respectively, along with the added column L3. L3 enables us to do the join.

9. **Union and Intersection are commutative.**

Explanation: Union and intersection are both distributive; we can enclose any tables in parantheses according to requirement and ease of access.

10. **Union and Intersection are associative.**

Explanation: Union and intersection are both distributive; we can enclose any tables in parantheses according to requirement and ease of access.

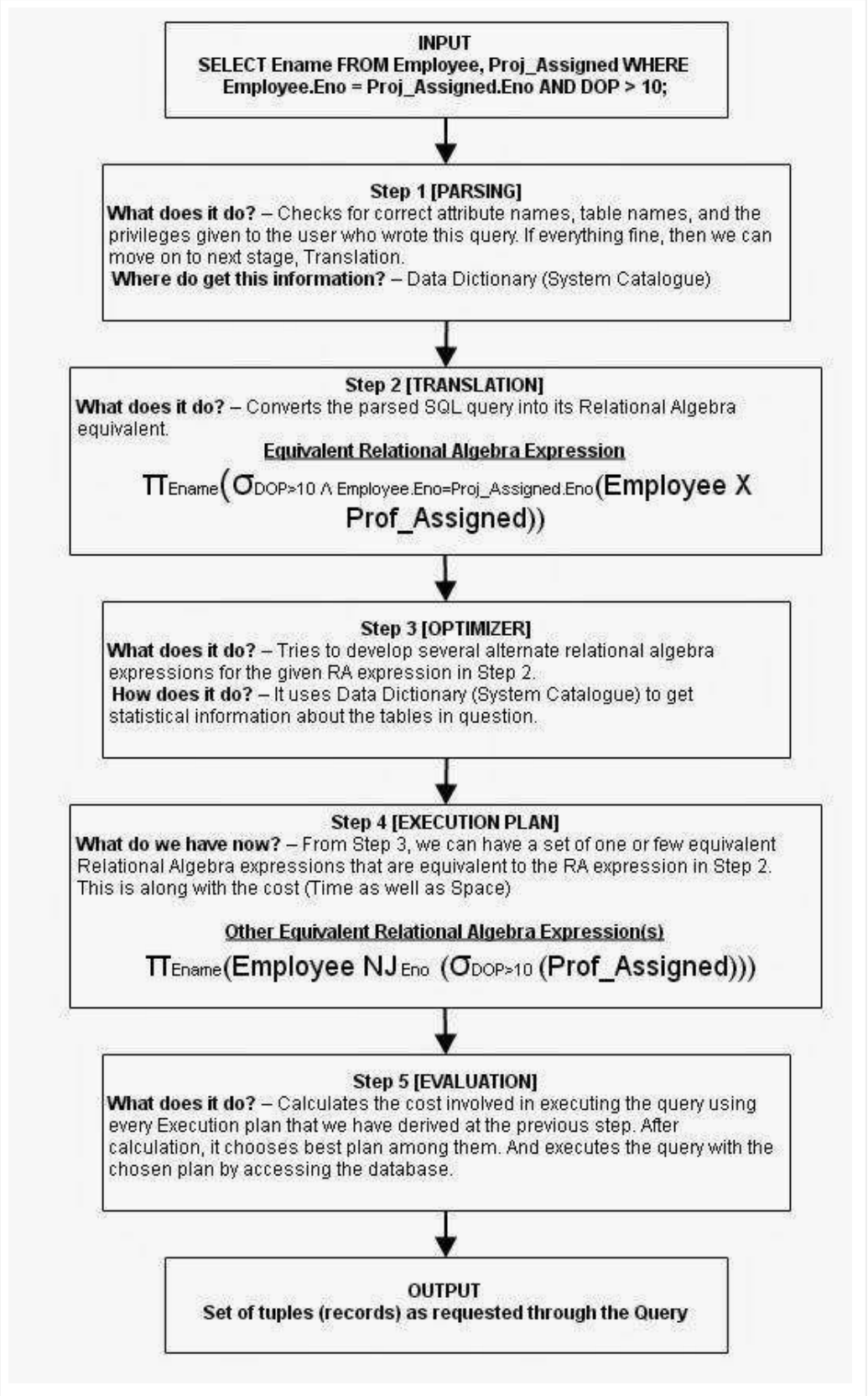
11. **Selection operation distributes over the union, intersection, and difference operations.**

Explanation: In set difference, we know that only those tuples are shown which belong to

table E1 and do not belong to table E2. So, applying a selection condition on the entire set difference is equivalent to applying the selection condition on the individual tables and then applying set difference. This will reduce the number of comparisons in the set difference step.

12. **Projection operation distributes over the union operation.**

Explanation: Applying individual projections before computing the union of E1 and E2 is more optimal than the left expression, i.e. applying projection after the union step.



LECTURE-18: Evaluation of Expressions

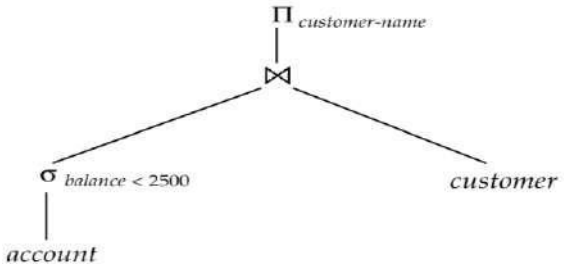
Evaluation of Expressions

- So far: we have seen algorithms for individual operations
- Alternatives for evaluating an entire expression tree
  - **Materialization**: generate results of an expression whose inputs are relations or are already computed, **materialize** (store) it on disk. Repeat.
  - **Pipelining**: pass on tuples to parent operations even as an operation is being executed
- We study above alternatives in more detail

Materialization

- **Materialized evaluation:** evaluate one operation at a time, starting at the lowest-level. Use intermediate results materialized into temporary relations to evaluate next-level operations.
- E.g., in figure below, compute and store

  then compute the store its join with *customer*, and finally compute the projections on *customer-name.*



- Materialized evaluation is always applicable
- Cost of writing results to disk and reading them back can be quite high
  - Our cost formulas for operations ignore cost of writing results to disk, so
    - Overall cost = Sum of costs of individual operations + cost of writing intermediate results to disk
- Double buffering: use two output buffers for each operation, when one is full write it to disk while the other is getting filled
  - Allows overlap of disk writes with computation and reduces execution time

Pipelining

- **Pipelined evaluation :** evaluate several operations simultaneously, passing the results of one operation on to the next.
  - E.g., in previous expression tree, don't store result of instead, pass tuples directly to the join.. Similarly, don't store result of join, pass tuples directly to projection.
- Much cheaper than materialization: no need to store a temporary relation to disk.
- Pipelining may not always be possible – e.g., sort, hash-join.
- For pipelining to be effective, use evaluation algorithms that generate output tuples even as tuples are received for inputs to the operation.
- Pipelines can be executed in two ways: **demand driven** and **producer driven**
- In **demand driven** or **lazy** evaluation
  - system repeatedly requests next tuple from top level operation
  - Each operation requests next tuple from children operations as required, in order to output its next tuple

- In between calls, operation has to maintain "**state**" so it knows what to return next
- Each operation is implemented as an **iterator** implementing the following operations
  - open()
    - E.g. file scan: initialize file scan, store pointer to beginning of file as state
    - E.g.merge join: sort relations and store pointers to beginning of sorted relations as state
  - next()
    - E.g. for file scan: Output next tuple, and advance and store file pointer
    - E.g. for merge join: continue with merge from earlier state till next output tuple is found. Save pointers as iterator state.
  - close()

- In produce-driven or **eager** pipelining
  - Operators produce tuples eagerly and pass them up to their parents
    - Buffer maintained between operators, child puts tuples in buffer, parent removes tuples from buffer
    - if buffer is full, child waits till there is space in the buffer, and then generates more tuples
  - System schedules operations that have space in output buffer and can process more input tuples

Evaluation Algorithms for Pipelining

- Some algorithms are not able to output results even as they get input tuples
  - E.g. merge join, or hash join
  - These result in intermediate results being written to disk and then read back always
- Algorithm variants are possible to generate (at least some) results on the fly, as input tuples are read in
  - E.g. hybrid hash join generates output tuples even as probe relation tuples in the in-memory partition (partition 0) are read in
  - **Pipelined join technique**: Hybrid hash join, modified to buffer partition 0 tuples of both relations in-memory, reading them as they become available, and output results of any matches between partition 0 tuples
    - When a new r_0 tuple is found, match it with existing s_0 tuples, output matches, and save it in r_0
    - Symmetrically for s_0 tuples

Complex Joins

- Join involving three relations: *loan* *depositor* *customer*
- **Strategy 1.** Compute *depositor* *customer;* use result to compute *loan* (*depositor* *customer*)
- **Strategy 2.** Computer *loan* *depositor* first, and then join the result with *customer.*
- **Strategy 3.** Perform the pair of joins at once. Build and index on *loan* for *loan-number,* and on *customer* for *customer-name.*
  - For each tuple *t* in *depositor,* look up the corresponding tuples in *customer* and the corresponding tuples in *loan.*
  - Each tuple of *deposit* is examined exactly once.
- Strategy 3 combines two operations into one special-purpose operation that is more efficient than implementing two joins of two relations.

Module-2:
LECTURE-19

Relational Algebra:

Basic operations:

1. *Selection* (σ) Selects a subset of rows from relation.
2. *Projection* (π) Selects a subset of columns from relation.
3. *Cross-product* (×) Allows us to combine two relations.
4. *Set-difference* ( ) Tuples in relation. 1, but not in relationn. 2.
5. *Union* (U) Tuples in reln. 1 and in reln. 2.
6. *Rename*(ρ) Use new name for the Tables or fields.

Additional operations:

7. Intersection (∩), *Join*(⋈), Division(÷): Not essential, but (very!) useful.

Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

Projection

- 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. (Unary Operation)
- Projection operator has to eliminate duplicates! (as it returns a relation which is a set)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Duplicate values may be representing different real world entity or relationship).

Example: Consider the BOOK table:

Acc-No	Title	Author
100	"DBMS"	"Silbershatz"
200	"DBMS"	"Ramanuj"
300	"COMPILER"	"Silbershatz"
400	"COMPILER"	"Ullman"
500	"OS"	"Sudarshan"
600	"DBMS"	"Silbershatz"

π_{Title}(BOOK) =

Title
"DBMS"
"COMPILER"
"OS"

Selection

- Selects rows that satisfy *selection condition.*
- 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.*)

Example: For the example given above:

$\sigma_{Acc\text{-}no>300}$(BOOK) =

Acc-No	Title	Author
400	"COMPILER"	"Ullman"
500	"OS"	"Sudarshan"

$\sigma_{Title="DBMS"}$(BOOK)=

Acc-No	Title	Author
100	"DBMS"	"Silbershatz"
200	"DBMS"	"Ramanuj"
600	"DBMS"	"Silbershatz"

$\pi_{Acc\text{-}no}$ ($\sigma_{Title="DBMS"}$ (BOOK))=

Acc-No
100
200
600

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - Corresponding' fields have the same type.
- What is the *schema* of result?

Consider:

Borrower

Cust-name	Loan-no
Ram	L-13
Shyam	L-30
Suleman	L-42

Depositor

Cust-name	Acc-no
Suleman	A-100
Radheshyam	A-300
Ram	A-401

List of customers who are either borrower or depositor at bank= $\pi_{Cust\text{-}name}$ (Borrower) U $\pi_{Cust\text{-}name}$ (Depositor)=

Cust-name
Ram
Shyam
Suleman
Radeshyam

Customers who are both borrowers and depositors = $\pi_{Cust\text{-}name}$ (Borrower) ∩ $\pi_{Cust\text{-}name}$ (Depositor)=

Cust-name
Ram
Suleman

Customers who are borrowers but not depositors = $\pi_{Cust\text{-}name}$ (Borrower) □ $\pi_{Cust\text{-}name}$ (Depositor)=

Cust-name
Shyam

Cartesian-Product or Cross-Product (S1 × R1)

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
- Consider the borrower and loan tables as follows:

Borrower:

Cust-name	**Loan-no**
Ram	L-13
Shyam	L-30
Suleman	L-42

Loan:

Loan-no	**Amount**
L-13	1000
L-30	20000
L-42	40000

Cross product of Borrower and Loan, Borrower × Loan =

Borrower.Cust-name	**Borrower.Loan-no**	**Loan.Loan-no**	**Loan.Amount**
Ram	L-13	L-13	1000
Ram	L-13	L-30	20000
Ram	L-13	L-42	40000
Shyam	L-30	L-13	1000
Shyam	L-30	L-30	20000
Shyam	L-30	L-42	40000
Suleman	L-42	L-13	1000
Suleman	L-42	L-30	20000
Suleman	L-42	L-42	40000

The rename operation can be used to rename the fields to avoid confusion when two field names are same in two participating tables:

For example the statement, $\rho_{\text{Loan-borrower(Cust-name,Loan-No-1, Loan-No-2,Amount)}}$(Borrower × Loan) results into- A new Table named Loan-borrower is created where it has four fields which are renamed as Cust-name, Loan-No-1, Loan-No-2 and Amount and the rows contains the same data as the cross product of Borrower and Loan.

Loan-borrower:

Cust-name	**Loan-No-1**	**Loan-No-2**	**Amount**
Ram	L-13	L-13	1000
Ram	L-13	L-30	20000
Ram	L-13	L-42	40000
Shyam	L-30	L-13	1000
Shyam	L-30	L-30	20000
Shyam	L-30	L-42	40000
Suleman	L-42	L-13	1000
Suleman	L-42	L-30	20000
Suleman	L-42	L-42	40000

Rename Operation:

It can be used in two ways :

- $\rho_x(E)$ return the result of expression E in the table named *x*.
- $\rho_{x(A_1,A_2,\ldots,A_n)}(E)$ return the result of expression E in the table named *x* with the attributes renamed to $A_1, A_2, \ldots, A_n$.
- It's benefit can be understood by the solution of the query " Find the largest account balance in the bank"

It can be solved by following steps:

- Find out the relation of those balances which are not largest.
- Consider Cartesion product of Account with itself i.e. Account × Account
- Compare the balances of first Account table with balances of second Account table in the product.
- For that we should rename one of the account table by some other name to avoid the confusion

It can be done by following operation

$\Pi_{Account.balance}\ (\sigma_{Account.balance < d.balance}(Account \times \rho_d(Account))$

- So the above relation contains the balances which are not largest.
- Subtract this relation from the relation containing all the balances i.e . $\Pi_{balance}\ (Account)$.

So the final statement for solving above query is

$\Pi_{balance}\ (Account) - \Pi_{Account.balance}\ (\sigma_{Account.balance < d.balance}(Account \times \rho_d(Account))$

LECTURE-20

Additional Operations

Natural Join ($S_1 \bowtie R_1$)

- Forms Cartesian product of its two arguments, performs selection forcing equality on those attributes that appear in both relations
- For example consider Borrower and Loan relations, the natural join between them $\text{Borrower} \bowtie \text{Loan}$ will automatically perform the selection on the table returned by Borrower × Loan which force equality on the attribute that appear in both Borrower and Loan i.e. Loan-no and also will have only one of the column named Loan-No.
- That means $\text{Borrower} \bowtie \text{Loan} = \sigma_{Borrower.Loan\text{-}no = Loan.Loan\text{-}no}$ (Borrower × Loan).
- The table returned from this will be as follows:

Eliminate rows that does not satisfy the selection criteria "$\sigma_{Borrower.Loan\text{-}no = Loan.Loan\text{-}no}$" from Borrower × Loan =

Borrower.Cust-name	**Borrower.Loan-no**	**Loan.Loan-no**	**Loan.Amount**
Ram	L-13	L-13	1000
~~Ram~~	~~L-13~~	~~L-30~~	~~20000~~
~~Ram~~	~~L-13~~	~~L-42~~	~~40000~~
~~Shyam~~	~~L-30~~	~~L-13~~	~~1000~~
Shyam	L-30	L-30	20000
~~Shyam~~	~~L-30~~	~~L-42~~	~~40000~~
~~Suleman~~	~~L-42~~	~~L-13~~	~~1000~~
~~Suleman~~	~~L-42~~	~~L-30~~	~~20000~~
Suleman	L-42	L-42	40000

And will remove one of the column named Loan-no.

- i.e. $\text{Borrower} \bowtie \text{Loan}$ =

Cust-name	**Loan-no**	**Amount**
Ram	L-13	1000
Shyam	L-30	20000
Suleman	L-42	40000

Division Operation:

- denoted by ÷ is used for queries that include the phrase "for all".
- For example "Find customers who has an account in all branches in branch city Agra". This query can be solved by following statement.

$\Pi_{Customer\text{-}name.\ branch\text{-}name}(\text{Depositor} \bowtie Account) \div \Pi_{branch\text{-}name}(\sigma_{Branch\text{-}city="Agra"}(Branch))$

- The division operations can be specified by using only basic operations as follows:

Let r(R) and s(S) be given relations for schema R and S with $S \subseteq R$

$r \div s = \Pi_{R\text{-}S}(r) - \Pi_{R\text{-}S}((\Pi_{R\text{-}S}(r) \times s) - \Pi_{R\text{-}S,S}(r))$

LECTURE-21

Tuple Relational Calculus

Relational algebra is an example of procedural language while tuple relational calculus is a nonprocedural query language.
A query is specified as:
{t | P(t)}, i.e it is the set of all tuples t such that predicate P is true for t.

The formula P(t) is formed using atoms which uses the relations, tuples of relations and fields of tuples and following symbols
$\in$(belongs to), $<, >, \leq, \geq, \neq, =$, (comparison operators)
These atoms can then be used to form formulas with following symbols
$\forall$ (universal qualifier generally called "for all")
$\exists$ (existential qualifier generally called "there exists")
$\wedge$ (and), $\vee$ (or), $\neg$(not)
For example : here are some queries and a way to express them using tuple calculus:

- Find the branch-name, loan-number and amount for loans over Rs 1200.
  $\{t \mid t \in Loan \wedge t[amount] > 1200\}$.

- Find the loan number for each loan of an amount greater that Rs1200.
  $\{t \mid \exists s \in Loan(t[loan\text{-}number] = s[loan\text{-}number] \wedge s[amount] > 1200\}$

- Find the names of all the customers who have a loan from the Sadar branch.
  $\{t \mid \exists s \in Borrower (t[customer\text{-}name] = s[customer\text{-}name] \wedge$
  $\exists u \in Loan (u[loan\text{-}number] = s[loan\text{-}number$
  $\wedge u[branch\text{-}name] = "Sadar"))\}$

- Find all customers who have a loan , an account, or both at the bank
  $\{t \mid \exists s \in Borrower (t[customer\text{-}name] = s[customer\text{-}name])$
  $\vee \exists u \in Depositor (t[customer\text{-}name] = u[customer\text{-}name])\}$

- Find only those customers who have both an account and a loan.
  $\{t \mid \exists s \in Borrower (t[customer\text{-}name] = s[customer\text{-}name])$
  $\wedge \exists u \in Depositor (t[customer\text{-}name] = u[customer\text{-}name])\}$

- Find all customers who have an account but do not have loan.
  $\{t \mid \exists u \in Depositor (t[customer\text{-}name] = u[customer\text{-}name]) \wedge$
  $\neg \exists s \in Borrower (t[customer\text{-}name] = s[customer\text{-}name])\}$

- Find all customers who have an account at all branches located in Agra
  $\{t \mid \forall w \in Branch(w[branch\text{-}city] = "Agra" \Rightarrow$
  $\exists s \in Depositor (t[customer\text{-}name] = s[customer\text{-}name]$
  $\wedge \exists u \in Account (u[account\text{-}number] = s[account\text{-}number]$
  $\wedge u[branch\text{-}name] = w[branch\text{-}name]))\}$

Domain Relational Calculus

1. Domain relational calculus is another non procedural language for expressing database queries.
2. A query is specified as:
   $\{<x_1,x_2,\ldots,x_n> \mid P(x_1,x_2,\ldots,x_n)\}$ where $x_1,x_2,\ldots,x_n$ represents domain variables. P represent a predicate formula as in tuple calculus

- Since the domain variables are referred in place of tuples the formula doesn't refer the fields of tuples rather they refer the domain variables.
- For example the queries in domain calculus are mentioned as follows:
  - Find the branch-name, loan-number and amount for loans over Rs 1200.
    $\{<b, l, a> \mid <b, l, a> \in Loan \wedge a > 1200\}$
  - Find the loan number for each loan of an amount greater that Rs1200.
    $\{< l > \mid \exists\, b,a(\ <b, l, a> \in Loan \wedge a > 1200\}$
  - Find the names of all the customers who have a loan from the Sadar branch and find the loan amount
    $\{< c, a > \mid \exists\, l(< c, l > \in Borrower$
    $\exists\, b(\ < b, l, a > \in Loan \wedge b="Sadar"))\}$
  - Find names of all customers who have a loan , an account, or both at the Sadar Branch
    $\{<c> \mid \exists\, l(< c, l > \in Borrower\ \wedge \exists\, b, a(<b, l, a> \in Loan \wedge b ="Sadar"))$
    $\vee\ \exists\, a(<c, a> \in Depositor \wedge \exists\, b, n(<b, a, n> \in Account \wedge b ="Sadar"))\}$
  - Find only those customers who have both an account and a loan.
    $\{<c> \mid \exists\, l(<c, l> \in Borrower\ ) \wedge \exists\, a(<c, a> \in Depositor\ )\}$
  - Find all customers who have an account but do not have loan.
    $\{t \mid \exists\, a(<c, a> \in Depositor\ ) \wedge \neg \exists\, l(<c, l> \in Borrower\ )\}$
  - Find all customers who have an account at all branches located in Agra
    $\{<c> \mid\ \forall\, x, y, z(<x, y, z> \in Branch) \wedge y = "Agra" \Rightarrow$

Outer Join.

Outer join operation is an extension of join operation to deal with missing information

- Suppose that we have following relational schemas:
  Employee(employee-name, street, city)
  Fulltime-works(employee-name, branch-name, salary)
  A snapshot of these relations is as follows:

Employee:

employee-name	**street**	**city**
Ram	M G Road	Agra
Shyam	New Mandi Road	Mathura
Suleman	Bhagat Singh Road	Aligarh

Fulltime-works

employee-name	**branch-name**	**salary**
Ram	Sadar	30000
Shyam	Sanjay Place	20000
Rehman	Dayalbagh	40000

Suppose we want complete information of the full time employees.

- The natural join ($Employee \bowtie Fulltime\text{-}works$)will result into the loss of information for Suleman and Rehman because they don't have record in both the tables (left and right relation). The outer join will solve the problem.
- Three forms of outer join:
  - **Left outer join($\supset\bowtie$):**the tuples which doesn't match while doing natural join from left relation are also added in the result putting null values in missing field of right relation.
  - **Right outer join($\bowtie\subset$):**the tuples which doesn't match while natural join from right relation are also added in the result putting null values in missing field of left relation.
  - **Full outer join($\supset\bowtie\subset$):** include both of the left and right outer joins i.e. adds the tuples which did not match either in left relation or right relation and put null in place of missing values.
- The result for three forms of outer join are as follows:

Left join: $Employee \supset\bowtie Fulltime\text{-}works_{=}$

employee-name	**street**	**City**	**branch-name**	**salary**
Ram	M G Road	Agra	Sadar	30000
Shyam	New Mandi Road	Mathura	Sanjay Place	20000
Suleman	Bhagat Singh Road	Aligarh	Null	Null

Right join: $Employee \bowtie\subset Fulltime\text{-}works_{=}$

employee-name	**street**	**city**	**branch-name**	**salary**
Ram	M G Road	Agra	Sadar	30000
Shyam	New Mandi Road	Mathura	Sanjay Place	20000
Rehman	null	null	Dayalbagh	40000

Full join: $Employee \supset\bowtie\subset Fulltime\text{-}works_{=}$

employee-name	**street**	**city**	**branch-name**	**salary**
Ram	M G Road	Agra	Sadar	30000
Shyam	New Mandi Road	Mathura	Sanjay Place	20000
Suleman	Bhagat Singh Road	Aligarh	null	null
Rehman	null	null	Dayalbagh	40000

LECTURE-22

Structured Query Language (SQL)

Introduction

- Commercial database systems use more user friendly language to specify the queries.
- SQL is the most influential commercially marketed product language.
- Other commercially used languages are QBE, Quel, and Datalog.

Basic Structure

- The basic structure of an SQL consists of three clauses: **select, from** and **where.**
- **select:** it corresponds to the projection operation of relational algebra. Used to list the attributes desired in the result.
- **from**: corresponds to the Cartesian product operation of relational algebra. Used to list the relations to be scanned in the evaluation of the expression
- **where**: corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the **from** clause.
- A typical SQL query has the form:

  select $A_1, A_2, \ldots, A_n$

  from $r_1, r_2, \ldots, r_m$

  where P
  - A_i represents an attribute
  - r_j represents a relation
  - P is a predicate
  - It is equivalent to following relational algebra expression:
  - $\Pi_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$

[Note:** The words marked in dark in this text work as keywords in SQL language. For example "select", "from" and "where" in the above paragraph are shown in bold font to indicate that they are keywords**]

Select Clause

Let us see some simple queries and use of **select** clause to express them in SQL.

- Find the names of all branches in the Loan relation

  select branch-name

  from Loan
- By default the **select** clause includes duplicate values. If we want to force the elimination of duplicates the **distinct** keyword is used as follows:

  select distinct branch-name

  from Loan
- The **all** key word can be used to specify *explicitly* that duplicates are not removed. Even if we not use **all** it means the same so we don't require **all** to use in **select** clause.

  select all branch-name

  from Loan
- The asterisk "*" can be used to denote "all attributes". The following SQL statement will select and all the attributes of Loan.

  select *

  from Loan
- The arithmetic expressions involving operators, +, -, *, and / are also allowed in **select** clause. The following statement will return the amount multiplied by 100 for the rows in Loan table.

select branch-name, loan-number, amount * 10 **from** Loan.

Where Clause

- Find all loan numbers for loans made at "Sadar" branch with loan amounts greater than Rs 1200.

  select loan-number
  from Loan
  where branch-name= "Sadar" **and** amount > 1200
- **where** clause uses uses logival connectives **and**, **or**, and **not**
- operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and < >.
- **between** can be used to simplify the comparisons

  select loan-number
  from Loan
  where amount between 90000 and 100000

From Clause

- The **from** clause by itself defines a Cartesian product of the relations in the clause.
- When an attribute is present in more than one relation they can be referred as *relation-name.attribute-name* to avoid the ambiguity.
- For all customers who have loan from the bank, find their names and loan numbers

  select distinct customer-name, Borrower.loan-number
  from Borrower, Loan
  where Borrower.loan-number = Loan.loan-number

The Rename Operation

- Used for renaming both relations both relations and attributes in SQL
- Use **as** clause: old-name **as** new-name
- Find the names and loan numbers of the customers who have a loan at the "Sadar" branch.

  select distinct customer-name, borrower.loan-number **as** loan-id
  from Borrower, Loan
  where Borrower.loan-number = Loan.loan-number **and**
  branch-name = "Sadar"

  we can now refer the loan-number instead by the name loan-id.
- For all customers who have a loan from the bank, find their names and loan-numbers.

  select distinct customer-name, T.loan-number
  from Borrower **as** T, Loan **as** S
  where T.loan-number = S.loan-number
- Find the names of all branches that have assets greater than at least one branch located in "Mathura".

  select distinct T.branch-name
  from branch **as** T, branch **as** S
  where T.assets > S.assets **and** S.branch-city = "Mathura"

String Operation

- Two special characters are used for pattern matching in strings:
  - Percent (%) : The % character matches any substring
  - Underscore(_): The _ character matches any character
- "%Mandi": will match with the strings ending with "Mandi" viz. "Raja Ki mandi", "Peepal Mandi"
- "_ _ _" matches any string of three characters.
- Find the names of all customers whose street address includes the substring "Main"

select customer-name
from Customer
where customer-street like "%Main%"

Set Operations

- **union, intersect and except** operations are set operations available in SQL.
- Relations participating in any of the set operation must be compatible; i.e. they must have the same set of attributes.
- Union Operation:
  - Find all customers having a loan, an account, or both at the bank
    (**select** customer-name **from** Depositor)
    union
    (**select** customer-name **from** Borrower)
    It will automatically eliminate duplicates.
  - If we want to retain duplicates **union all** can be used
    (**select** customer-name **from** Depositor)
    union all
    (**select** customer-name **from** Borrower)
- Intersect Operation
  - Find all customers who have both an account and a loan at the bank
    (**select** customer-name **from** Depositor)
    intersect
    (**select** customer-name **from** Borrower)
  - If we want to retail all the duplicates
    (**select** customer-name **from** Depositor)
    intersect all
    (**select** customer-name **from** Borrower)
- Except Opeartion
  - Find all customers who have an account but no loan at the bank
    (**select** customer-name **from** Depositor)
    except
    (**select** customer-name **from** Borrower)
  - If we want to retain the duplicates:
    (**select** customer-name **from** Depositor)
    except all
    (**select** customer-name **from** Borrower)

Aggregate Functions

- Aggregate functions are those functions which take a collection of values as input and return a single value.
- SQL offers 5 built in aggregate functions-
  - Average: **avg**
  - Minimum:**min**
  - Maximum:**max**
  - Total: **sum**
  - Count:**count**
- The input to **sum** and **avg** must be a collection of numbers but others may have collections of non-numeric data types as input as well
- Find the average account balance at the Sadar branch
  select avg(balance)

from Account
where branch-name= “Sadar”

The result will be a table which contains single cell (one row and one column) having numerical value corresponding to average balance of all account at sadar branch.

- **group by** clause is used to form groups, tuples with the same value on all attributes in the **group by** clause are placed in one group.
- Find the average account balance at each branch

  select branch-name, **avg**(balance)
  from Account
  group by branch-name
- By default the aggregate functions include the duplicates.
- **distinct** keyword is used to eliminate duplicates in an aggregate functions:
- Find the number of depositors for each branch

  select branch-name, **count**(**distinct** customer-name)
  from Depositor, Account
  where Depositor.account-number = Account.account-number
  group by branch-name
- **having** clause is used to state condition that applies to groups rather than tuples.
- Find the average account balance at each branch where average account balance is more than Rs. 1200

  select branch-name, **avg**(balance)
  from Account
  group by branch-name
  having avg(balance) > 1200
- Count the number of tuples in Customer table

  select count(*)
  from Customer
- SQL doesn’t allow **distinct** with **count**(*)
- When **where** and **having** are both present in a statement **where** is applied before **having**.

LECTURE-23

Nested Sub queries

A subquery is a **select-from-where** expression that is nested within another query.

Set Membership

The **in** and **not in** connectives are used for this type of subquery.

"Find all customers who have both a loan and an account at the bank", this query can be written using nested subquery form as follows

select distinct customer-name
from Borrower
where customer-name **in**(**select** customer-name
from Depositor)

- Select the names of customers who have a loan at the bank, and whose names are neither "Smith" nor "Jones"

select distinct customer-name
from Borrower
where customer-name **not in**("Smith", "Jones")

Set Comparison

Find the names of all branches that have assets greater than those of at least one branch located in Mathura

select branch-name
from Branch
where asstets > **some** (**select** assets
from Branch
where branch-city = "Mathura")

1. Apart from > **some** others comparison could be < **some** , <= **some** , >= **some** , = **some** , < > **some.**
2. Find the names of all branches that have assets greater than that of each branch located in Mathura

select branch-name
from Branch
where asstets > **all** (**select** assets
from Branch
where branch-city = "Mathura")

- Apart from > **all** others comparison could be < **all** , <= **all** , >= **all** , = **all** , < >**all.**

Views

In SQL **create view** command is used to define a view as follows:

create view v **as** <query expression>

where <query expression> is any legal query expression and v is the view name.

- The view consisting of branch names and the names of customers who have either an account or a loan at the branch. This can be defined as follows:

create view All-customer **as**
(**select** branch-name, customer-name
from Depositor, Account
where Depositor.account-number=account.account-number)
union

(**select** branch-name, customer-name
from Borrower, Loan
where Borrower.loan-number = Loan.loan-number)

- The attributes names may be specified explicitly within a set of round bracket after the name of view.
- The view names may be used as relations in subsequent queries. Using the view Allcustomer

Find all customers of Sadar branch
select customer-name
from All-customer
where branch-name= "Sadar"

- A **create-view** clause creates a view definition in the database which stays until a command - **drop view** *view-name* - is executed.

Modification of Database

Deletion

- In SQL we can delete only whole tuple and not the values on any particular

attributes. The command is as follows:

delete from r **where** P.
where P is a predicate and r is a relation.

- **delete** command operates on only one relation at a time. Examples are as follows:
- Delete all tuples from the Loan relation

delete from Loan

- Delete all of the Smith's account record

delete from Depositor
where customer-name = "Smith"

- Delete all loans with loan amounts between Rs 1300 and Rs 1500.
  delete from Loan
  where amount **between** 1300 **and** 1500
- Delete the records of all accounts with balances below the average at the bank

delete from Account
where balance < (**select avg**(balance)
from Account)

Insertion

In SQL we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. Examples are as follows:
Insert an account of account number A-9732 at the Sadar branch having balance of Rs 1200
insert into Account
values("Sadar", "A-9732", 1200)
the values are specified in the order in which the corresponding attributes are listed in the relation schema.
SQL allows the attributes to be specified as part of the **insert** statement
insert into Account(account-number, branch-name, balance)
values("A-9732", "Sadar", 1200)
insert into Account(branch-name, account-number, balance)

values(“Sadar”, “A-9732”, 1200)

Provide for all loan customers of the Sadar branch a new Rs 200 saving account for each loan account they have. Where loan-number serve as the account number for these accounts.
insert into Account
select branch-name, loan-number, 200
from Loan
where branch-name = “Sadar”

Updates

Used to change a value in a tuple without changing all values in the tuple.
Suppose that annual interest payments are being made, and all balances are to be increased by 5 percent.
update Account
set balance = balance * 1.05
Suppose that accounts with balances over Rs10000 receive 6 percent interest, whereas all others receive 5 percent.
update Account
set balance = balance * 1.06
where balance > 10000
update Account
set balance = balance * 1.05
where balance <= 10000

Data Definition Language

Data Types in SQL

char(n): fixed length character string, length n.
varchar(n): variable length character string, maximum length n.
int: an integer.
smallint: a small integer.
numeric(p,d): fixed point number, p digits(plus a sign), and d of the p digits are to right of the decimal point.
real, double precision: floating point and double precision numbers.
float(n): a floating point number, precision at least n digits.
date: calendar date; four digits for year, two for month and two for day of month.
time: time of day n hours minutes and seconds.

Domains can be defined as
create domain person-name **char**(20).
the domain name person-name can be used to define the type of an attribute just like built-in domain.

Schema Definition in SQL
create table command is used to define relations.
create table r (A1D1, A2D2,… , AnDn,
<integrity constraint1>,
… ,
<integrity constraintk>)

where r is relation name, each Ai is the name of attribute, Di is the domain type of

values of Ai. Several types of integrity constraints are available to define in SQL.

Integrity Constraints which are allowed in SQL are

primary key(Aj1, Aj2,… , Ajm)
and
check(P) where P is the predicate.

drop table command is used to remove relations from database.
alter table command is used to add attributes to an existing relation
alter table r **add** A D
it will add attribute A of domain type D in relation r.
alter table r **drop** A
it will remove the attribute A of relation r.

LECTURE-24

Integrity Constraints

- Integrity Constraints guard against accidental damage to the database.
- Integrity constraints are predicates pertaining to the database.
- **Domain Constraints:**
- Predicates defined on the domains are Domain constraints.
- Simplest Domain constraints are defined by defining standard data types of the attributes like Integer, Double, Float, etc.
- We can define domains by **create domain** clause also we can define the constraints on such domains as follows:

  create domain hourly-wage **numeric**(5,2)
  constraint wage-value-test check(value >= 4.00)
- So we can use hourly-wage as data type for any attribute where DBMS will automatically allow only values greater than or equal to 4.00.
- Other examples for defining Domain constraints are as follows:

  create domain account-number **char**(10)
  constraint account-number-null-test **check(value not null)**
  create domain account-type **char**(10)
  constraint account-type-test
  check (value in ("Checking", "Saving"))

By using the later domain of two above the DBMS will allow only values for any attribute having type as account-type i.e. Checking and Saving.

- **Referential Integrity:**
- **Foreign Key**: If two table R and S are related to each other, K1 and K2 are primary keys of the two relations also K1 is one of the attribute in S. Suppose we want that every row in S must have a corresponding row in R, then we define the K1 in S as **foreign key**. Example in our original database of library we had a table for relation BORROWEDBY, containing two fields Card No. and Acc. No. . Every row of BORROWEDBY relation must have corresponding row in USER Table having same Card No. and a row in BOOK table having same Acc. No.. Then we will define the Card No. and Acc. No. in BORROWEDBY relation as foreign keys.
- In other way we can say that every row of BORROWEDBY relation must refer to some row in BOOK and also in USER tables.
- Such referential requirement in one table to another table is called Referential Integrity.

LECTURE-25

Query by Example (QBE)

Query by Example (QBE) is a method of query creation that allows the user to search for documents based on an example in the form of a selected text string or in the form of a document name or a list of documents. Because the QBE system formulates the actual query, QBE is easier to learn than formal query languages, such as the standard Structured Query Language (SQL), while still enabling powerful searches.

Selections in QBE

QBE uses skeleton tables to represent table name and fieldnames like:

Table name	Field1	Field2	…..

For selection, P operator along with variable name/constant name is used to display one or more fields.

Example 1:

Consider the relation: student (name, roll, marks)

The following query can be represented as:

SQL: select name from student where marks>50;

student	name	roll	marks
	P.X		>50

Here X is a constant; alternatively we can use _X as a variable.

Example 2:

For the relation given above

The following query can be represented as:

SQL: select * from student where marks>50 and marks <80;

student	name	roll	marks
P.			_X

CONDITION
_X>50 ∧ _X<80

We can use condition box to represent complex conditions.

Example 3:

For the relation given above

The following query can be represented as:

SQL: select name, roll from student where marks<50 or marks >80;

student	name	roll	marks
	P.A	P.B	<50
	P.A	P.B	>80

OR operation retrieves results in multiple rows.

Example 4:

(Joins in QBE)

Consider the following tables:

Student (name, roll, marks)

Grades (roll, grade)

The following query can be represented as:

SQL: select s.name, g.grade from Student s, Grades g where s.roll=g.roll and s.marks>50;
Uses two skeleton tables:

Student	name	roll	marks
	P.A	_X	>50

And

Grades	roll	grade
	_X	P.B

Insertions in QBE:

Uses operator I. on the table.
Example: Consider the following query on Student table
SQL: insert into student values ('abc',10,60);

Student	name	roll	marks
I.	abc	10	60

Multiple insertions can be represented by separate rows in skeleton table.

Deletions in QBE:

Uses operator D. on the table.
Example: Consider the following query on Student table
SQL: delete from student where marks=0;

Student	name	roll	marks
D.			0

Multiple deletions can be represented by separate rows in skeleton table.
Deletions without any condition can truncate the entire table.

Updation in QBE:

Uses operator U. on the table.
Example: Consider the following query on Student table
SQL: update student set mark=50 where roll=40;

Student	name	roll	marks
U.			50
		40	

LECTURE-26

RELATIONAL DATABASE DEGIN

Database design is a process in which you create a logical data model for a database, which store data of a company. It is performed after initial database study phase in the database life cycle. You use normalization technique to create the logical data model for a database and eliminate data redundancy.

Normalization also allows you to organize data efficiently in a database and reduce anomalies during data operation. Various normal forms, such as first, second and third can be applied to create a logical data model for a database. The second and third normal forms are based on partial dependency and transitivity dependency. Partial dependency occurs when a row of table is uniquely identified by one column that is a part of a primary key. A transitivity dependency occurs when a non key column is uniquely identified by values in another non-key column of a table.

Database Design Process:

We can identify six main phases of the database design process:

1. Requirement collection and analysis
2. Conceptual database design
3. Choice of a DBMS
4. Data model mapping(logical database design)
5. Physical database design
6. Database system implementation and tuning

1. Requirement Collection and Analysis

Before we can effectively design a data base we must know and analyze the expectation of the users and the intended uses of the database in as much as detail.

2. Conceptual Database Design

The goal for this phase is to produce a conceptual schema for the database that is independent of a specific DBMS.

- We often use a high level data model such ER-model during this phase
- We specify as many of known database application on transactions as possible using a notation that is independent of any specific dbms.
- Often the dbms choice is already made for the organization the intent of conceptual design still to keep , it as free as possible from implementation consideration.

3. **Choice of a DBMS**

The choice of dbms is governed by a no. of factors some technical other economic and still other concerned with the politics of the organization.

The economics and organizational factors that offer the choice of the dbms are:

Software cost, maintenance cost, hardware cost, database creation and conversion cost, personnel cost, training cost, operating cost.

4. **Data model mapping (logical database design)**

During this phase, we map the conceptual schema from the high level data model used on

phase 2 into a data model of the choice dbms.

5. **Physical databse design**
   During this phase we design the specification for the database in terms of physical storage structure ,record placement and indexes.
6. **Database system implementation and tuning**
   During this phase, the database and application programs are implemented, tested and eventually deployed for service.

Informal Guidelines for Relation Design

Want to keep the semantics of the relation attributes clear. The information in a tuple should represent exactly one fact or an entity. The hidden or buried entities are what we want to discover and eliminate.

- Design a relation schema so that it is easy to explain its meaning.
- Do not combine attributes from multiple entity types and relationship types into a single relation. Use a view if you want to present a simpler layout to the end user.
- A relation schema should correspond to on entity type or relationship type.
- Minimize redundant information in tuples, thus reducing update anomalies
- If anomalies are present, try to decompose the relation into two or more to represent the separate facts, or document the anomalies well for management in the applications programs.

Minimize the use of **null** values. Nulls have multiple interpretations:

- The attribute does not apply to this tuple
- The attribute value is unknown
- The attribute value is absent
- The attribute value might represent an actual value

If nulls are likely (non-applicable) then consider decomposition of the relation into two or more relations that hold only the non-null valued tuples.

- Do not permit the creation of spurious tuples

Too much decomposition of relations into smaller ones may also lose information or generate erroneous information

- Be sure that relations can be logically joined using natural join and the result doesn't generate relationships that don't exist

Functional Dependencies

FD's are constraints on well-formed relations and represent a formalism on the infrastructure of relation.

Definition: A *functional dependency* (FD) on a relation schema **R** is a constraint $\boldsymbol{X \rightarrow Y}$**,** where X and Y are subsets of attributes of **R.**

Definition: an FD is a relationship between an attribute "Y" and a determinant (1 or more other attributes) "X" such that for a given value of a determinant the value of the attribute is uniquely defined.

- X is a determinant
- X determines Y
- Y is functionally dependent on X
- X → Y
- X →Y is trivial if Y ⊆ X

Definition: An FD ***X*** → ***Y*** is *satisfied* in an instance **r** of **R** if for every pair of tuples, *t* and s: if *t* and *s* agree on all attributes in *X* then they must agree on all attributes in *Y*

A key constraint is a special kind of functional dependency: all attributes of relation occur on the right-hand side of the FD:

- *SSN → SSN, Name, Address*

Example Functional Dependencies

Let R be
NewStudent(***stuId, lastName, major, credits, status, socSecNo***)

FDs in R include

- ***{stuId}→{lastName}***, but not the reverse
- ***{stuId} →{lastName, major, credits, status, socSecNo, stuId}***
- ***{socSecNo} →{stuId, lastName, major, credits, status, socSecNo}***
- ***{credits}→{status},*** but not ***{status}→{credits}***

ZipCode→AddressCity

- 16652 is Huntingdon's ZIP

ArtistName→BirthYear

- Picasso was born in 1881

Autobrand→Manufacturer, Engine type

- Pontiac is built by General Motors with gasoline engine

Author, Title→PublDate

- Shakespeare's Hamlet was published in 1600

Trivial Functional Dependency

The FD X→Y is ***trivial*** if set {Y} is a subset of set {X}

Examples: If A and B are attributes of R,

- {A}→{A}
- {A,B} →{A}
- {A,B} →{B}
- {A,B} →{A,B}

are all trivial FDs and will not contribute to the evaluation of normalization.

FD Axioms

Understanding: Functional Dependencies are recognized by analysis of the real world; no automation or algorithm. Finding or recognizing them are the database designer's task.

FD manipulations:

- **Soundness** -- no incorrect FD's are generated
- **Completeness** -- all FD's can be generated

Axiom Name	Axiom	Example
Reflexivity	if a is set of attributes, b ⊆ a, then a →b	*SSN,Name → SSN*
Augmentation	if a→ b holds and c is a set of attributes, then ca→cb	*SSN → Name* then *SSN,Phone → Name, Phone*
Transitivity	if a →b holds and b→c holds, then a→ c holds	*SSN →Zip* and *Zip → City* then *SSN →City*
Union or Additivity *	if a → b and a → c holds then a→ bc holds	*SSN→Name* and *SSN→Zip* then *SSN→Name,Zip*
Decomposition or Projectivity*	if a → bc holds then a → b and a → c holds	*SSN→Name,Zip* then *SSN→Name* and *SSN→Zip*
Pseudotransitivity*	if a → b and cb → d hold then ac → d holds	*Address → Project* and *Project,Date →Amount* then *Address,Date → Amount*
(NOTE)	*ab→ c does NOT imply a → c and b → c*	

*Armstrong's Axioms (basic axioms)

LECTURE-27

CLOSURE OF A SET OF FUNCTIONAL DEPEDENCIES

Given a relational schema R, a functional dependencies f on R is logically implied by a set of functional dependencies F on R if every relation instance r(R) that satisfies F also satisfies f.

The closure of F, denoted by F^+, is the set of all functional dependencies logically implied by F.
The closure of F can be found by using a collection of rules called **Armstrong axioms.**
Reflexivity rule: If A *is a set of attributes and B is subset or equal to A, then A*→B holds.
Augmentation rule: If A→B holds and C is a set of attributes, then CA→CB holds
Transitivity rule: If A→B holds and B→C holds, then A→C holds.
Union rule: If A→B holds and A→C then A→BC holds
Decomposition rule: If A→BC holds, then A→B holds and A→C holds.
Pseudo transitivity rule: If A→B holds and BC→D holds, then AC→D holds.

Suppose we are given a relation schema R=(A,B,C,G,H,I) and the set of function dependencies {A→B,A→C,CG→H,CG→I,B→H}

We list several members of F^+ here:

1. A→H, since A→B and B→H hold, we apply the transitivity rule.
2. CG→HI. Since CG→H and CG→I , the union rule implies that CG→HI
3. AG→I, since A→C and CG→I, the pseudo transitivity rule implies that AG→I holds

Algorithm of compute F+ :

To compute the closure of a set of functional dependencies F:

F+ = *F*

repeat

for each functional dependency *f* in *F*+

apply reflexivity and augmentation rules on *f*

add the resulting functional dependencies to *F*+

for each pair of functional dependencies *f*1and *f*2 in *F*+

if *f*1 and *f*2 can be combined using transitivity

then add the resulting functional dependency to *F*+

until *F*+ does not change any further

large.

LECTURE-28

LOSS LESS DECOMPOSITION

A decomposition of a relation scheme R<S,F> into the relation schemes Ri(1<=i<=n) is said to be a lossless join decomposition or simply lossless if for every relation R that satisfies the FDs in F, the natural join of the projections or R gives the original relation R, i.e,

$R=\square_{R1}(R)\bowtie\square_{R2}(R)\bowtie\ldots\ldots..\bowtie\square_{Rn}(R)$

If R is subset of $\square_{R1}(R)\bowtie\square_{R2}(R)\bowtie\ldots\ldots..\bowtie\square_{Rn}(R)$

Then the decomposition is called lossy.

DEPEDENCY PRSERVATION:

Given a relation scheme R<S,F> where F is the associated set of functional dependencies on the attributes in S,R is decomposed into the relation schemes R1,R2,…Rn with the fds F1,F2…Fn, then this decomposition of R is dependency preserving if the closure of F' (where F'=F1 U F2 U … Fn)

Example:

Let R(A,B,C) AND F={A→B}. Then the decomposition of R into R1(A,B) and R2(A,C) is lossless because the FD { A→B} is contained in R1 and the common attribute A is a key of R1.

Example:

Let R(A,B,C) AND F={A→B}. Then the decomposition of R into R1(A,B) and R2(B,C) is not lossless because the common attribute B does not functionally determine either A or C. i.e, it is not a key of R1 or R 2.

Example:

Let R(A,B,C,D) and F={A→B, A→C, C→D,}. Then the decomposition of R into R1(A,B,C) with the FD F1={ A→B , A→C }and R2(C,D) with FD F2={ C→D} . In this decomposition all the original FDs can be logically derived from F1 and F2, hence the decomposition is dependency preserving also . the common attribute C forms a key of R2. The decomposition is lossless.

Example:

Let R(A,B,C,D) and F={A→B, A→C, A→D,}. Then the decomposition of R into R1(A,B,D) with the FD F1={ A→B , A→D }and R2(B,C) with FD F2={ } is lossy because the common attribute B is not a candidate key of either R1 and R2 .

In addition , the fds A→C is not implied by any fds R1 or R2. Thus the decomposition is not dependency preserving.

Full functional dependency:

Given a relational scheme R and an FD X→Y ,Y is fully functional dependent on X if there is no Z, where Z is a proper subset of X such that Z→Y. The dependency X→Y is left reduced, there being no extraneous attributes attributes in the left hand side of the dependency.

Partial dependency:

Given a relation dependencies F defined on the attributes of R and K as a candidate key ,if X is a proper subset of K and if F|= X→A, then A is said to be partial dependent on K

Prime attribute and non prime attribute:

A attribute A in a relation scheme R is a **prime attribute** or simply **prime** if A is part of any candidate key of the relation. If A is not a part of any candidate key of R, A is called a nonprime attribute or simply **non prime** .

Trivial functional dependency:
A FD X→Y is said to be a trivial functional dependency if Y is subset of X.

LECTURE-29

Normalization

While designing a database out of an entity–relationship model, the main problem existing in that "raw" database is redundancy. Redundancy is storing the same data item in more one place. A redundancy creates several problems like the following:

1. Extra storage space: storing the same data in many places takes large amount of disk space.
2. Entering same data more than once during data insertion.
3. Deleting data from more than one place during deletion.
4. Modifying data in more than one place.
5. Anomalies may occur in the database if insertion, deletion, modification etc are no done properly. It creates inconsistency and unreliability in the database.

To solve this problem, the "raw" database needs to be normalized. This is a step by step process of removing different kinds of redundancy and anomaly at each step. At each step a specific rule is followed to remove specific kind of impurity in order to give the database a slim and clean look.

Un-Normalized Form (UNF)

If a table contains non-atomic values at each row, it is said to be in UNF. An **atomic value** is something that can not be further decomposed. A **non-atomic value**, as the name suggests, can be further decomposed and simplified. Consider the following table:

Emp-Id	Emp-Name	Month	Sales	Bank-Id	Bank-Name
E01	AA	Jan	1000	B01	SBI
		Feb	1200		
		Mar	850		
E02	BB	Jan	2200	B02	UTI
		Feb	2500		
E03	CC	Jan	1700	B01	SBI
		Feb	1800		
		Mar	1850		
		Apr	1725		

In the sample table above, there are multiple occurrences of rows under each key Emp-Id. Although considered to be the primary key, Emp-Id cannot give us the unique identification facility for any single row. Further, each primary key points to a variable length record (3 for E01, 2 for E02 and 4 for E03).

First Normal Form (1NF)

A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values. The above table in UNF can be processed to create the following table in 1NF.

Emp-Id	Emp-Name	Month	Sales	Bank-Id	Bank-Name
E01	AA	Jan	1000	B01	SBI
E01	AA	Feb	1200	B01	SBI
E01	AA	Mar	850	B01	SBI
E02	BB	Jan	2200	B02	UTI
E02	BB	Feb	2500	B02	UTI
E03	CC	Jan	1700	B01	SBI
E03	CC	Feb	1800	B01	SBI
E03	CC	Mar	1850	B01	SBI
E03	CC	Apr	1725	B01	SBI

As you can see now, each row contains unique combination of values. Unlike in UNF, this relation contains only atomic values, i.e. the rows can not be further decomposed, so the relation is now in 1NF.

Second Normal Form (2NF)

A relation is said to be in 2NF f if it is already in 1NF and each and every attribute fully depends on the primary key of the relation. Speaking inversely, if a table has some attributes which is not dependant on the primary key of that table, then it is not in 2NF.

Let us explain. Emp-Id is the primary key of the above relation. Emp-Name, Month, Sales and Bank-Name all depend upon Emp-Id. But the attribute Bank-Name depends on Bank-Id, which is not the primary key of the table. So the table is in 1NF, but not in 2NF. If this position can be removed into another related relation, it would come to 2NF.

Emp-Id	Emp-Name	Month	Sales	Bank-Id
E01	AA	JAN	1000	B01
E01	AA	FEB	1200	B01
E01	AA	MAR	850	B01
E02	BB	JAN	2200	B02
E02	BB	FEB	2500	B02
E03	CC	JAN	1700	B01
E03	CC	FEB	1800	B01
E03	CC	MAR	1850	B01
E03	CC	APR	1726	B01

Bank-Id	Bank-Name
B01	SBI
B02	UTI

After removing the portion into another relation we store lesser amount of data in two relations without any loss information. There is also a significant reduction in redundancy.

Third Normal Form (3NF)

A relation is said to be in 3NF, if it is already in 2NF and there exists no **transitive dependency** in that relation. Speaking inversely, if a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF.

What is a transitive dependency? Within a relation if we see
A → B [B depends on A]
And
B → C [C depends on B]
Then we may derive
A → C[C depends on A]

Such derived dependencies hold well in most of the situations. For example if we have
Roll → Marks
And
Marks → Grade
Then we may safely derive
Roll → Grade.

This third dependency was not originally specified but we have derived it.

The derived dependency is called a transitive dependency when such dependency becomes improbable. For example we have been given
Roll → City
And
City → STDCode

If we try to derive Roll → STDCode it becomes a transitive dependency, because obviously the STDCode of a city cannot depend on the roll number issued by a school or college. In such a case the relation should be broken into two, each containing one of these two dependencies:
Roll → City
And
City → STD code

LECTURE-30

Boyce-Code Normal Form (BCNF)

A relationship is said to be in BCNF if it is already in 3NF and the left hand side of every dependency is a candidate key. A relation which is in 3NF is almost always in BCNF. These could be same situation when a 3NF relation may not be in BCNF the following conditions are found true.

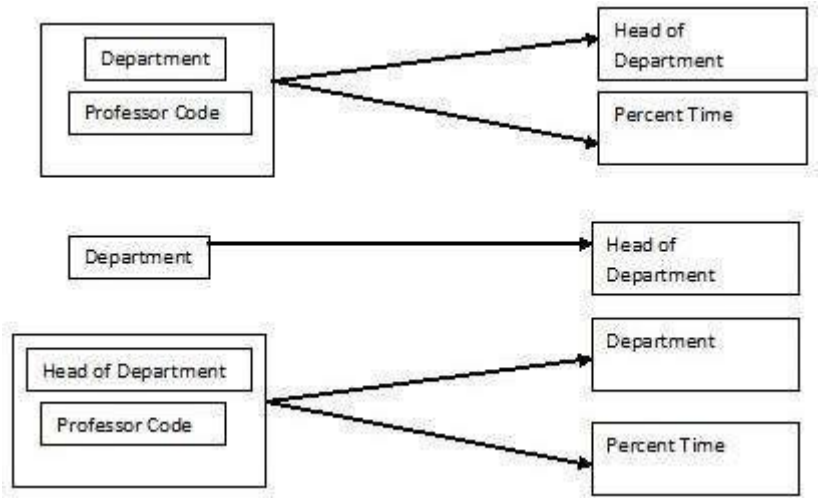
1. The candidate keys are composite.
2. There are more than one candidate keys in the relation.
3. There are some common attributes in the relation.

Professor Code	Department	Head of Dept.	Percent Time
P1	Physics	Ghosh	50
P1	Mathematics	Krishnan	50
P2	Chemistry	Rao	25
P2	Physics	Ghosh	75
P3	Mathematics	Krishnan	100

Consider, as an example, the above relation. It is assumed that:

1. A professor can work in more than one department
2. The percentage of the time he spends in each department is given.
3. Each department has only one Head of Department.

The relation diagram for the above relation is given as the following:



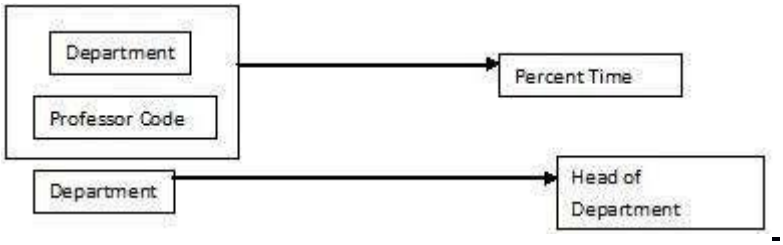
The given relation is in 3NF. Observe, however, that the names of Dept. and Head of Dept. are duplicated. Further, if Professor P2 resigns, rows 3 and 4 are deleted. We lose the information that Rao is the Head of Department of Chemistry.

The normalization of the relation is done by creating a new relation for Dept. and Head of Dept. and deleting Head of Dept. form the given relation. The normalized relations are shown in the following.

Professor Code	Department	Percent Time
P1	Physics	50
P1	Mathematics	50
P2	Chemistry	25
P2	Physics	75
P3	Mathematics	100

Department	Head of Dept.
Physics	Ghosh
Mathematics	Krishnan
Chemistry	Rao

See the dependency diagrams for these new relations.



Fourth Normal Form (4NF)

When attributes in a relation have multi-valued dependency, further Normalization to 4NF and 5NF are required. Let us first find out what multi-valued dependency is.

A **multi-valued dependency** is a typical kind of dependency in which each and every attribute within a relation depends upon the other, yet none of them is a unique primary key.

We will illustrate this with an example. Consider a vendor supplying many items to many projects in an organization. The following are the assumptions:

1. A vendor is capable of supplying many items.
2. A project uses many items.
3. A vendor supplies to many projects.
4. An item may be supplied by many vendors.

A multi valued dependency exists here because all the attributes depend upon the other and yet none of them is a primary key having unique value.

Vendor Code	Item Code	Project No.
V1	I1	P1
V1	I2	P1
V1	I1	P3
V1	I2	P3
V2	I2	P1
V2	I3	P1

V3	I1	P2
V3	I1	P3

The given relation has a number of problems. For example:

1. If vendor V1 has to supply to project P2, but the item is not yet decided, then a row with a blank for item code has to be introduced.
2. The information about item I1 is stored twice for vendor V3.

Observe that the relation given is in 3NF and also in BCNF. It still has the problem mentioned above. The problem is reduced by expressing this relation as two relations in the Fourth Normal Form (4NF). A relation is in 4NF if it has no more than one independent multi valued dependency or one independent multi valued dependency with a functional dependency.

The table can be expressed as the two 4NF relations given as following. The fact that vendors are capable of supplying certain items and that they are assigned to supply for some projects in independently specified in the 4NF relation.

Vendor-Supply

Vendor Code	Item Code
V1	I1
V1	I2
V2	I2
V2	I3
V3	I1

Vendor-Project

Vendor Code	Project No.
V1	P1
V1	P3
V2	P1
V3	P2

Fifth Normal Form (5NF)

These relations still have a problem. While defining the 4NF we mentioned that all the attributes depend upon each other. While creating the two tables in the 4NF, although we have preserved the dependencies between Vendor Code and Item code in the first table and Vendor Code and Item code in the second table, we have lost the relationship between Item Code and Project No. If there were a primary key then this loss of dependency would not have occurred. In order to revive this relationship we must add a new table like the following. Please note that during the entire process of normalization, this is the only step where a new table is created by joining two attributes, rather than splitting them into separate tables.

Project No.	Item Code
P1	11
P1	12
P2	11
P3	11
P3	13

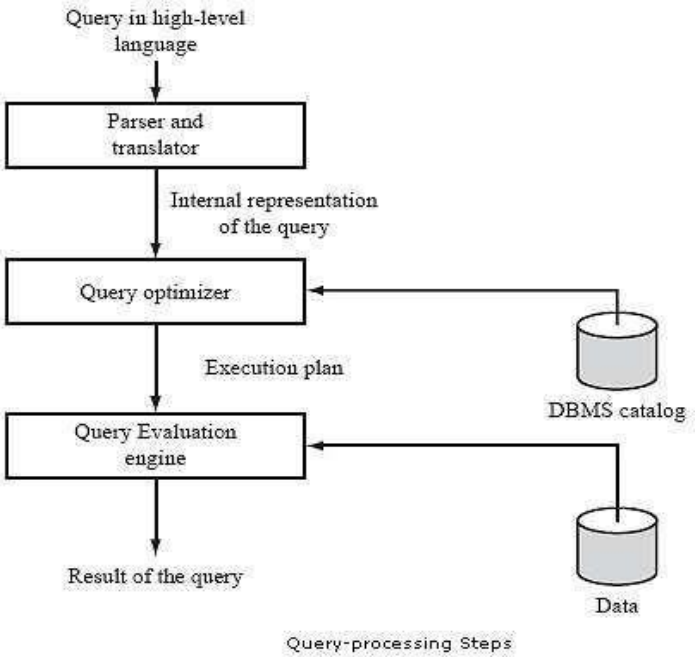
Let us finally summarize the normalization steps we have discussed so far.

Input Relation	Transformation	Output Relation
All Relations	Eliminate variable length record. Remove multi-attribute lines in table.	1NF
1NF Relation	Remove dependency of non-key attributes on part of a multi-attribute key.	2NF
2NF	Remove dependency of non-key attributes on other non-key attributes.	3NF
3NF	Remove dependency of an attribute of a multi attribute key on an attribute of another (overlapping) multi-attribute key.	BCNF
BCNF	Remove more than one independent multi-valued dependency from relation by splitting relation.	4NF
4NF	Add one relation relating attributes with multi-valued dependency.	

LECTURE-31

QUERY PROCESSING

Query processing includes translation of high-level queries into low-level expressions that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result. It is a three-step process that consists of parsing and translation, optimization and execution of the query submitted by the user.



Query-processing Steps

A query is processed in four general steps:

1. Scanning and Parsing
2. Query Optimization or planning the execution strategy
3. Query Code Generator (interpreted or compiled)
4. Execution in the runtime database processor

1. Scanning and Parsing

When a query is first submitted (via an applications program), it must be scanned and parsed to determine if the query consists of appropriate syntax.
Scanning is the process of converting the query text into a tokenized representation.
The tokenized representation is more compact and is suitable for processing by the parser.
This representation may be in a tree form.
The **Parser** checks the tokenized representation for correct syntax.
In this stage, checks are made to determine if columns and tables identified in the query exist in the database and if the query has been formed correctly with the appropriate keywords and structure.
If the query passes the parsing checks, then it is passed on to the Query Optimizer.

2. Query Optimization or Planning the Execution Strategy

For any given query, there may be a number of different ways to execute it.
Each operation in the query (SELECT, JOIN, etc.) can be implemented using one or more different *Access Routines.*

For example, an access routine that employs an index to retrieve some rows would be more efficient that an access routine that performs a full table scan.
The goal of the **query optimizer** is to find a *reasonably efficient* strategy for executing the query (not quite what the name implies) using the access routines.
Optimization typically takes one of two forms: *Heuristic Optimization* or *Cost Based Optimization*
In **Heuristic Optimization**, the query execution is refined based on *heuristic rules* for reordering the individual operations.
With **Cost Based Optimization**, the overall cost of executing the query is systematically reduced by estimating the costs of executing several different execution plans.

3. Query Code Generator (interpreted or compiled)

Once the query optimizer has determined the execution plan (the specific ordering of access routines), the code generator writes out the actual access routines to be executed.
With an interactive session, the query code is interpreted and passed directly to the runtime database processor for execution.
It is also possible to *compile* the access routines and store them for later execution.

4. Execution in the runtime database processor

At this point, the query has been scanned, parsed, planned and (possibly) compiled.
The runtime database processor then executes the access routines against the database.
The results are returned to the application that made the query in the first place.
Any runtime errors are also returned.

Lecture-32

Query Optimization

To enable the system to achieve (or improve) acceptable performance by choosing a better (if not the best) strategy during the process of a query. One of the great strengths to the relational database.

Automatic Optimization vs. Human Programmer

1. A good automatic optimizer will have a wealth of information available to it that human programmers typically do not have.
2. An automatic optimizer can easily reprocess the original relational request when the organization of the database is changed. For a human programmer, reorganization would involve rewriting the program.
3. The optimizer is a program, and therefore is capable of considering literally hundreds of different implementation strategies for a given request, which is much more than a human programmer can.
4. The optimizer is available to a wide range of users, in an efficient and cost-effective manner.

The Optimization Process

1. Cast the query into some internal representation, such as a query tree structure.
2. Convert the internal representation to canonical form.

*A subset (say C) of a set of queries (say Q) is said to be a set of canonical forms for Q if and only if every query Q is equivalent to just one query in C.

During this step, some optimization is already achieved by transforming the internal representation to a better canonical form.

Possible improvements

a. Doing the restrictions (selects) before the join.
b. Reduce the amount of comparisons by converting a restriction condition to an equivalent condition in **conjunctive normal form**- that is, a condition consisting of a set of restrictions that are ANDed together, where each restriction in turn consists of a set of simple comparisons connected only by OR's.
c. A sequence of restrictions (selects) before the join.
d. In a sequence of projections, all but the last can be ignored.
e. A restriction of projection is equivalent to a projection of a restriction.
f. Others

3. Choose candidate low-level procedures by evaluate the transformed query.

*Access path selection: Consider the query expression as a series of basic operations (join, restriction, etc.), then the optimizer choose from a set of pre-defined, low-level implementation procedures. These procedures may involve the user of primary key, foreign key or indexes and other information about the database.

4. Generate query plans and choose the cheapest by constructing a set of candidate query plans first, then choose the best plan. To pick the best plan can be achieved by assigning cost to each given plan. The costs is computed according to the number of disk I/O's involved.

Module-3:
Lecture-33

Transaction

A transaction is the smallest unit of operation done on a database.

It can be basically of two types:

→Read Transaction

→Write Transaction

3.2. ACID Properties of transaction:

Atomicity: (all or nothing)

A transaction is said to be atomic if a transaction always executes all its actions in one step or not executes any actions at all It means either all or none of the transactions operations are performed.

Consistency: (No violation of integrity constraints)

A transaction must preserve the *consistency* of a database after the execution. The DBMS assumes that this property holds for each transaction. Ensuring this property of a transaction is the responsibility of the user.

Isolation: (concurrent changes invisibles)

The transactions must behave as if they are executed in isolation. It means that if several transactions are executed concurrently the results must be same as if they were executed serially in some order. The data used during the execution of a transaction cannot be used by a second transaction until the first one is completed.

Durability: (committed update persist)

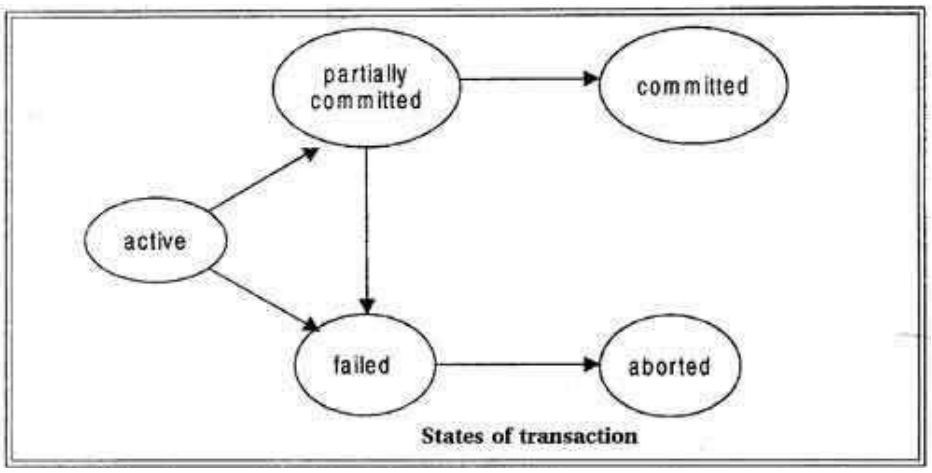
The effect of completed or committed transactions should persist even after a crash. It means once a transaction commits, the system must guarantee that the result of its operations will never be lost, in spite of subsequent failures.

3.3. States of a transaction:

A transaction must be in one of the following states:

- **Active**: the initial state, the transaction stays in this state while it is executing.
- **Partially committed**: after the final statement has been executed.
- **Failed**: when the normal execution can no longer proceed.
- **Aborted**: after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
- **Committed**: after successful completion.

The state diagram corresponding to a transaction is shown in Figure.



We say that a transaction has committed only if it has entered the committed state. Similarly, we say that a transaction has aborted only if it has entered the aborted state. A transaction is said to have terminated if has either committed or aborted.

A transaction starts in the active state. When it finishes its final statement, it enters the partially committed state. At this point, the transaction has completed its execution, but it is still possible that it may have to be aborted, since the actual output may still be temporarily hiding in main memory and thus a hardware failure may preclude its successful completion

The database system then writes out enough information to disk that, even in the event of a failure, the updates performed by the transaction can be recreated when the system restarts after the failure. When the last of this information is written out, the transaction enters the committed state.

Problems due to concurrency:

1) Lost update problem

2) Dirty Read problem

3) Incorrect summary problem

1. The lost update problem: A second transaction writes a second value of a data-item (datum) on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.
2. The dirty read problem: Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have been read by any transaction ("dirty read"). The reading transactions end with incorrect results.
3. The incorrect summary problem: While one transaction takes a summary over the values of all the instances of a repeated data-item, a second transaction updates some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

Concurrency Control Techniques:

1) Locking

2) Timestamp ordering

3) Multi-version concurrency control

4) Optimistic concurrency control

Locking (2-Phase Locking)

2PL protocol locks are applied and removed in two phases:

1. Expanding phase: locks are acquired and no locks are released.
2. Shrinking phase: locks are released and no locks are acquired.

Locks are of two types:

1. Binary Lock
2. Share/Exclusive (Read/Write) Lock

→Binary Lock

A binary lock can have 2 States or values

- Locked (or 1) and
- Unlocked (or 0)

We represent the current state(or value) of the lock associated with data item X as LOCK(X).

Operations used with Binary Locking

1. **lock_item :** A transaction request access to an item by first issuing a lock_item(X) operation.
    - **If LOCK(X) =1 or L(X) :** the transaction is forced to wait.
    - **If LOCK(X) = 0 or U(x) :** it is set to 1(the transaction **locks** the item) and the transaction is a load to access item X.
2. **unlock_item :** After using the data item the transaction issues an operation **unlock(X),** which sets the operation **LOCK(X) to 0** i.e. unlocks the data item so that X may be accessed by another transactions.

Transaction Rules for Binary Locks

Every transaction must obey the following rules :

→A transaction T must issue the lock(X) operation before any read(X) or write(X) operations in T.
→A transaction T must issue the unlock(X) operation after all read(X) and write(X) operations in T.
→If a transaction T already holds the lock on item X, then T will not issue a lock(X) operation.

→If a transaction does not holds the lock on item X, then T will not issue an unlock(X) operation.

```
lock_item (X):
  B: if LOCK (X)=0  (* item is unlocked *)
     then LOCK (X)←1  (* lock the item *)
     else begin
       wait  (until lock (X)=0  and
         the lock manager wakes up the transaction);
       go to B
       end;

unlock_item (X):
  LOCK (X)←0;  (* unlock the item *)
  if any transactions are waiting
   then wakeup one of the waiting transactions;
```

→Shared/Read and Exclusive/Write Lock

The binary lock is too restrictive for data items because at most one transaction can hold on a given item whether the transaction is reading or writing. To improve it we have shared and exclusive locks in which more than one transaction can access the same item for reading purposes.i.e. the read operations on the same item by different transactions are not conflicting.

In this types of lock, system supports two kinds of lock :

- Exclusive(or Write) Locks and
- Shared(or Read) Locks.

Shared Locks

If a transaction Ti has locked the data item A in shared mode, then a request from another transaction Tj on A for :

- **Write operation on A :** Denied. Transaction Tj has to wait until transaction Ti unlock A.
- **Read operation on A :** Allowed.

Exclusive Locks

If a transaction Ti has locked a data item a in exclusive mode then request from some another transaction Tj for

- **Read operation on A :** Denied
- **Write operation on A :** Denied

Operations Used with Shared and Exclusive Locks

1. Read_lock(A) or s(A)
2. Write_lock(A) or X(A)
3. Unlock(X) or U(A)

Implementation of Shared and Exclusive Locks

Shared and exclusive locks are implemented using 4 fields :

1. Data_item_name
2. LOCK
3. Number of Records and
4. Locking_transaction(s)

Again to save space, items that are not in the lock table are considered to be unlocked. The system maintains only those records for the items that are currently locked in the lock table.

Value of LOCK(A) : Read Locked or Write Locked

- **If LOCK(A) = write-locked –** The value of locking transaction is a single transaction that holds the exclusive(write) Lock on A.
- **If LOCK(A) = read-locked –** The value of locking transaction is a list of one or more transactions that hold the Shared(read) on A.

Transaction Rules for Shared and Exclusive Locks

Every transaction must obey the following rules :

1. A transaction T must issue the operation **s(A) or read_lock(A) or x(A) or write_lock(A)** before any read(A) operation is performed in T.
2. A transaction T must issue the operation x(A) or write_lock(A) before any write(A) operation is performed in T.
3. After completion of all read(A) and write(A) operations in T, a transaction T must issue an unlock(A) operation.
4. If a transaction already holds a read (shared) lock or a write (exclusive) lock on item A, then T will not issue an unlock(A) operation.
5. A transaction that already holds a lock on item A, is allowed to convert the lock from one locked state to another under certain conditions.
    - **Upgrading the Lock by Issuing a write_lock(A) Operation or Conversion of read_lock() to write_lock() :**
        - **Case 1 – When Conversion Not Possible :** A transaction T will not issue a write_lock(A) operation if it already holds a read (shared) lock or write (exclusive) lock on item A.
        - **Case 2 – When Conversion Possible :** If T is the only transaction holding a read lock on A at the time it issues the write_lock(A) operation, the lock can be upgraded;
    - **Downgrading the Lock by Issuing a read_lock(A) or Conversion of write_lock() to read_lock() :**
      A transaction T downgrade from the write lock to a read lock by acquiring the write_lock(A) or x(A), then the read_lock(A) or s(A) and then releasing the write_lock(A) or x(A).

```
read_lock (X):
  B: if LOCK (X)='unlocked"
     then begin LOCK (X)←  "read-locked";
                no_of_reads(X)←  1
          end
     else if LOCK(X)='read-locked"
          then no_of_reads(X)←  no_of_reads(X) + 1
          else begin wait (until LOCK (X)='unlocked' and
                     the lock manager wakes up the transaction);
               go to B
               end;

write_lock (X):
  B: if LOCK (X)='unlocked"
     then LOCK (X)←  "write-locked"
     else begin
          wait (until LOCK(X)="unlocked" and
            the lock manager wakes up the transaction);
          go to B
          end;

unlock_item (X):
  if LOCK (X)='write-locked'
     then begin LOCK (X)←  "unlocked;"
          wakeup one of the waiting transactions, if any
          end
     else if LOCK(X)='read-locked"
          then begin
               no_of_reads(X)←  no_of_reads(X) – 1;
               if no_of_reads(X)=0
                 then begin LOCK (X)="unlocked";
                      wakeup one of the waiting transactions, if any
                      end
                 end;
```

CONTENTS

S.No	Chapter Name	Page No
1	BASIC CONCPETS OF DBMS	1-6
2	DATA MODELS	7-17
3	RELATIONAL DATABASE	18-29
4	NORMALIZATION IN RELATIONAL SYSTEM	30-39
5	STRUCTURED QUERY LANGUAGE	40-44
6	TRANSACTION PROCESSING CONCEPTS	45-48
7	CONCURRENCY CONTROL CONCEPTS	49-55
8	SECURITY AND INTEGRITY	56-62

- **Telecommunications**-> For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards and storing information about the communication network.
- **Finance**-> For storing information about holdings, sales and purchases of financial instruments such as stocks and bonds.
- **Manufacturing**-> For management of supply chain and for tracking production of items in factories, inventory of items in warehouses and orders for items.
- **Human resources**-> For information about employees, salaries, payroll taxes& benefits and for generation of payments.

ADVANTAGE OF DBMS/ PURPOSE OF DATABASE SYSTEMS:-

1. **Reduction of redundancy & inconsistency:** Unlike file processing system in database approach data can be stored at a single place or Centralized control of data by the Database administrator. It avoids unnecessary duplication of data which saves space and does not permit inconsistency.
2. **Shared data:-** A database allows the sharing of data under its control by any number of application programs or users.
3. **Data independence**: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to protect application code from such details.
4. **Integrity can be improved:** Data integrity refers to validity and consistency. Data integrity means data should be accurate & consistent. This is done by providing some checks & constraints. These are the consistency rules that the database is not permitted to violate. For Example, the age of an employee can be between 18 to 58 years only. While entering the data for the age of an employee, the database should check this.
5. **Supports Concurrent access**: A DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Example: Two people reading a balance and updating it at the same time and gives accurate result.
6. **Improved Backup & Recovery:** DBMS provides facilities for recovering the hardware & software failures. A backup & recovery subsystem is responsible for this. In case a program fails, it restores the database to a state in which it was before the execution of program.
7. **Improved Security:** DBMS can enforce access controls that govern what data is visible to different classes of users. For Example, salary of the employees should not be visible anyone other than the department dealing in this.

Disadvantages of DBMS

Although there are many advantages of DBMS, the DBMS may also have some minor disadvantages. These are:

1. **Cost of Hardware & Software**: A processor with high speed of data processing and memory of large size is required to run the DBMS software. It means that you have to upgrade the hardware used for file-based system. Similarly, DBMS software is also very costly.

2. **Cost of Data Conversion:** When a computer file-based system is replaced with a database system, the data stored into data file must be converted to database file. It is very difficult and costly method to convert data of data files into database.
3. **Cost of Staff Training:** Most DBMSs are often complex systems so the training for users to use the DBMS is required. Training is required at all levels, including programming, application development, and database administration. The organization has to be paid a lot of amount for the training of staff to run the DBMS.
4. **Appointing Technical Staff:** The trained technical persons such as database administrator, application programmers, data entry operators etc are required to handle the DBMS. You have to pay handsome salaries to these persons. Therefore, the" system cost increases".
5. **Database Damage:** In most of the organizations, all data is integrated into a single database. If database is damaged due to electric failure or database is corrupted on the storage media, then your valuable data may be lost forever.

DBA

Database Administrator (DBA): This is the chief administrator, who oversees and manages the database system (including the data and software). Duties include authorizing users to access the database, coordinating/monitoring its use, acquiring hardware/software for upgrades, etc. In large organizations, the DBA might have a support staff.

ROLE OF A DBA

Defining the Schema:

The DBA defines the schema which contains the structure of the data in the application. The DBA determines what data needs to be present in the system how this data has to be represented and organized.

Defining Security & Integrity Checks:

The DBA finds about the access restrictions to be defined and defines security checks accordingly. Data Integrity checks are also defined by the DBA.

Defining Backup / Recovery Procedures:

The DBA also defines procedures for backup and recovery. Defining backup procedures includes specifying what data is to backed up, the periodicity of taking backups and also the medium and storage place for the backup data.

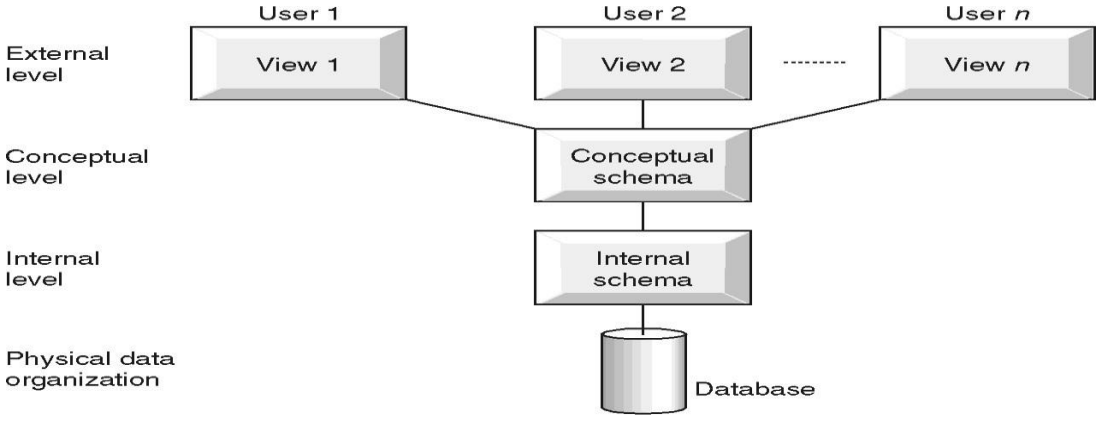
Monitoring Performance:

The DBA has to continuously monitor the performance of the queries and take measures to optimize all the queries in the application.

THREE LEVEL ARCHITECTURE OF DBMS / LOGICAL DBMS ARCHITECTURE / LEVEL OF ABSTRACTION

- A database system is a collection of interrelated data and set of programs that allows users to access and modify these data.

- The major purpose database system is to provide users with an abstract view of data (i.e. the system hides certain details of data how data are stored and maintained).
- The logical architecture describes how data in the database is perceived by users. It is not concerned with how data is handled and processed by DBMS, but only with how it looks.
- The system is called ANSI/SPARC MODEL and system is divided in to three levels of abstraction: the internal or physical level, conceptual or logical level, external or view level.



1. **Physical level / Internal level:**

- This level of abstraction is concerned with physical storage of information.
- It provides the internal view of the actual physical storage of data by telling how the data is stored.
- This level defines various stored data types.
- Here the complex low level data structures are described in details.

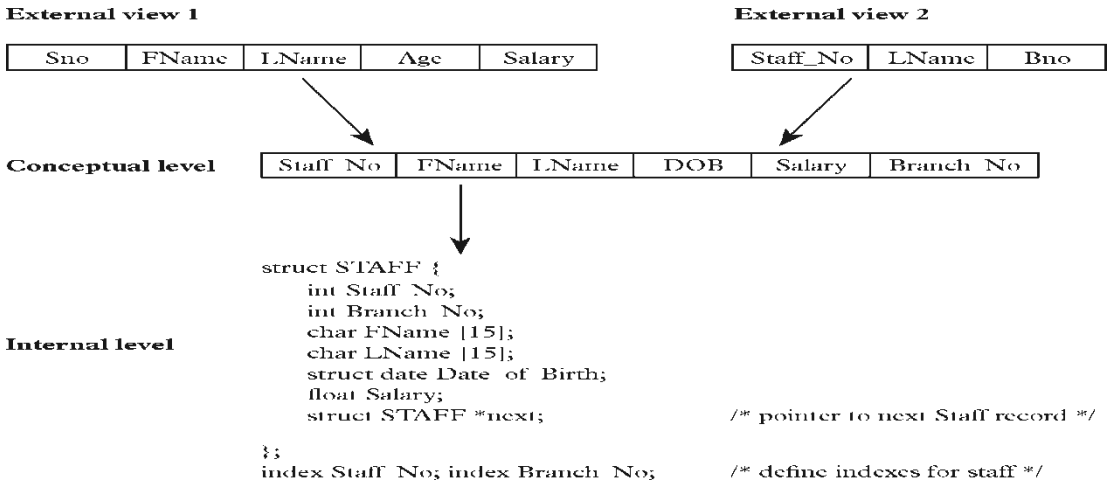
2. **Conceptual/ Logical level:**

- The next higher level of abstraction is conceptual level which describes what data are stored and relationship among them.
- Here the entire data base describes in terms of a small number of relatively simple structures.
- This level of abstraction is used by the database administrator, who must decide what information is kept in the database.
- Only one logical view is present.

3. **External/view level:**

- This is the highest level of abstraction which describes only part of entire data base.(i.e. Describes that part of database that is relevant to a particular user.)
- Many users of database system will not be concerned in all information. The system may provide different views for different users.
- This level is closest to the user.
- The view level of abstraction exists to simplify their interaction with the system.

Differences between Three Levels of ANSI-SPARC Architecture:



Schemas:

- The overall design of the database is called database schema or database scheme.
- In other words it is the logical structure of the database.
- It is analogous to type information (i.e. data type) of a variable in a program.
- Database system has several schemas, partitioned according to level of abstraction. They are:

1. Physical schema: The physical schema describes the database design at the physical level, which is the lowest level of abstraction describing how the data are actually stored.
2. Logical schema: The logical schema describes the database design at the logical level, which describes what data are stored in the database and what relationship exists
3. subschemas: The several schemas present at the view level to describe different views of the database.

Instances:

- The collection of information stored in a database at a particular point in time is called an instance of the database.
- It is analogous to the value of a variable.

DATA DICTIONARY

- The relational database system needs to maintain data about relation. This information is called data dictionary.
- In other word, a data dictionary is a set of metadata (i.e. data about data) which contains the definition and representation of data elements.

- The data dictionary is created when database is created.
- Then after, whenever the database is in operation data dictionary is updated automatically by DBMS.
- The data dictionary is a reference for all database users.

The contents of dictionary are:

- Names of relations, time of its creation and when it was last accessed.
- The names and data types of its attributes of each relation.
- Domains of the attributes.
- Where the data is stored.
- Names of views defined on the database and definitions of those views.
- Constraints applied to tables.
- Names of authorized user and the rights and privileges they have granted.

DATABASE USER

- There are different types of database system users differentiated by the way they expect to interact with the system.

Naive Users:-

- These are end users who interact with the system by invoking permanent application program that have been written previously. They need not be aware about the application program they only use it.
- Ex:- ATM users

Application Programmer:-

- They are the computer professional who develop the application program. The application programs could be written in a general purpose programming language such as PASCAL, COBOL, C, C++.

Sophisticated User:-

- These users interact with the system without writing the program. They form the request by writing queries in database query language.
- Analysts who submit queries to explore data in the database fall in this category.

Specialized User:-

- Users who are responsible to write specialized database applications that do not fit into the conventional data processing system. For ex. Computer Aided Design (CAD), Artificial Intelligence System etc.

Chapter-2
DATA MODELS

DATA INDEPENDENCE: The ability to modify a schema definition in one level without affecting a schema definition in the next higher level is called data independence.

OR

Data and programs are independent of each other, so change is one has no or minimum effect on the other. Data and its structure are stored in the database where as application programs manipulating this data are stored separately, the change in one does not necessarily affect the other.

These are of 2 types:

1. **Logical data independence**: The ability to modify the conceptual schema without changing the external schema or application programs to be rewritten.

Example: The addition or removal of new entities attributes or relationships to the conceptual schema should be possible without having to change existing external schemas or having to rewrite existing application programs.

2. **Physical data independence:** The ability to modify the internal schema without having to change the conceptual schemas.

Modifications at this level are usually to improve performance.

Changes in the physical schema may include:

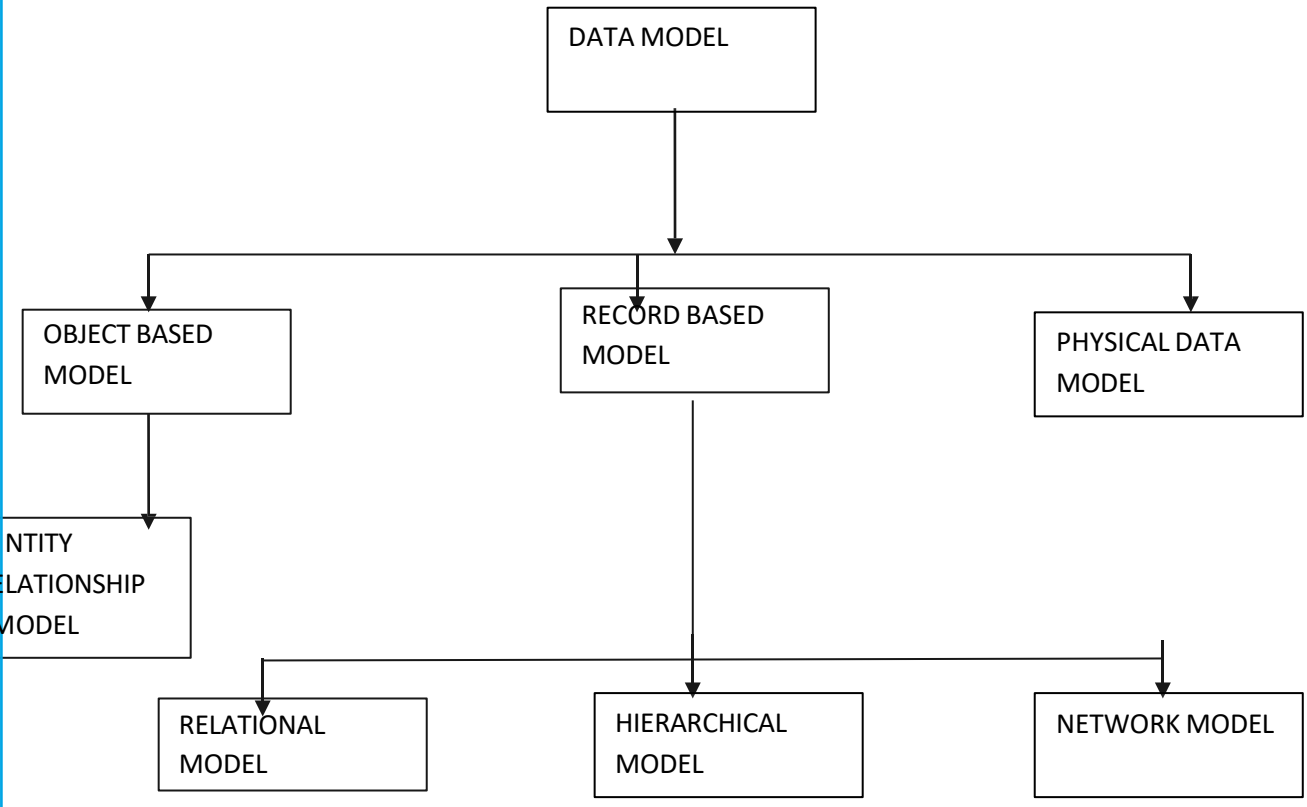
- Using new storage device
- Using different data structures
- Switching from one access method to another, modifying indexes etc.

Data model

Data models are the collection of conceptual tools which describes data, their relationship, their schematic and consistency constraints.

In other word we can say that a database model defines the logical data structure & the data relationships.

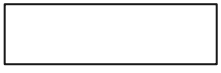
A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system.



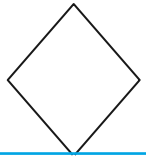
OBJECT BASED MODEL:

This data model uses objects as the key data representation components.
For ex: Entity-Relationship model
ENTITY-RELATIONSHIP MODEL

- It is a collection of real world objects called entities and their relationships.
- It is mainly represented in graphical form using E-R diagrams.
- This model is very useful in database design.
- E-R diagram which consists of following components.

 : - Represents Entity

:-Represents Attributes

 : Represents Relationship among entity set

_____:-Represents link between entity set and entity set to relationship set.

Entity:

An entity is an object of concern to represent the things in real world and is distinguishable from other by certain property.e.g. car, table,book etc.

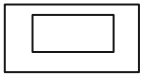
*An entity need not be physical entity, it can also be represents a concept in real world.e.g. project, loan etc.

*It represents a class of things, not any instance.

e.g. (student) entity has instances of "Ramesh and Mohan".

Entity Set/Entity Type:

A collection of a similar kind of entities is called an entity set or entity type.
It is of two types.
1. Weak entity set: Weak entity set does not contain any key attribute (i.e. primary key).

It is denoted by 

2. Strong entity set: The entity type containing a key attribute are called strong entity or regular entity set.

It is denoted by

e.g.- The 'STUDENT' entity has a key attitude ROLL NO which uniquely identifies it.

Attribute:

An attribute is a property used to describe the specific feature of entity. So to describe an entity entirely, a set of attributes is used.

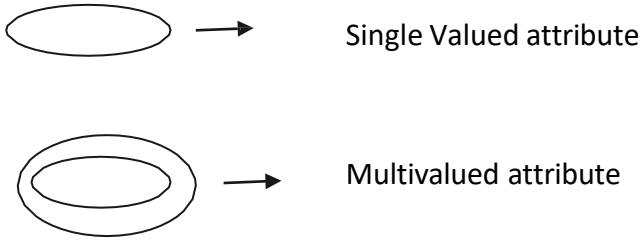
Types Of Attribute:

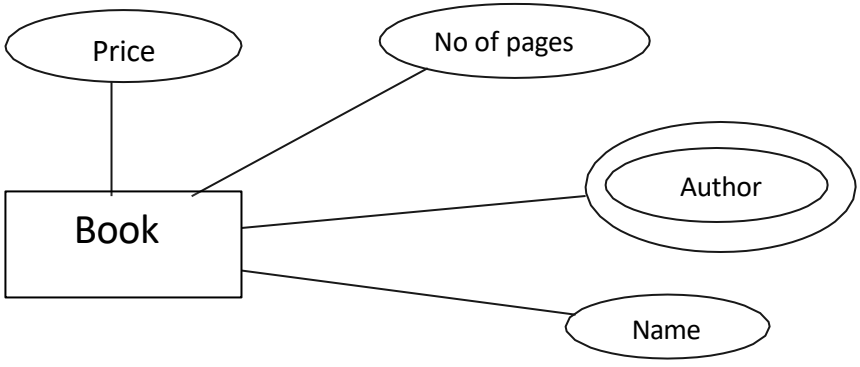
Attribute are of following type:
1. Single valued and multi valued
2. Simple and composite
3. Store and derived
4. Null value

Single valued and multi valued attribute:

The attributes having a single value for a particular entity is called single valued.
Ex: age of a person is a single-values attribute.

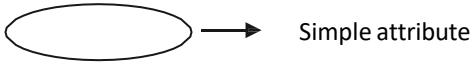
Attributes that have more than one value for a particular entity is called a multivalued attribute .e.g. book entity have attributes name, author, no of pages. Here author is the multivalued attribute as a single book has more than one author & others are single valued.



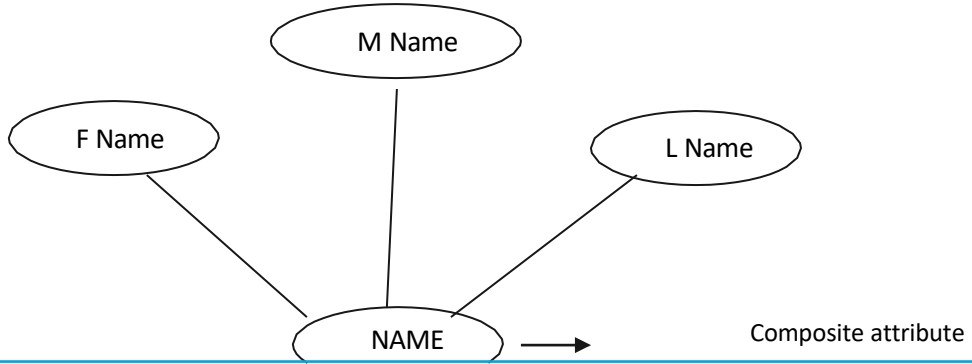


Simple and composite Attribute

The attribute that cannot be further sub-divided into smaller parts and represents the basic meaning is called simple attribute.
e.g.'age'.'Roll no' etc.



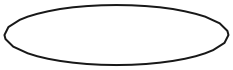
*Composite attributes are attributes that can be further divided into smaller units and each individual unit contains a specific meaning. .e.g. The 'NAME 'attribute of an employee entity can be subdivided into 'First name', middle name', 'last name'.



STORED AND DERIVED ATTEIBUTE

Attributes that are directly stored in the database are called stored attribute.

e.g. 'Date of birth' of a person.



Stored attribute

An attribute whose value is derived from the stored attribute is known as derived attribute.

Example: age, and it's value is derived from the stored attribute Date of Birth.

Derived attribute

NULL VALUED ATTRIBUTE

*The attribute which has no value.

RELATIONSHIP

A relationship can be defined as a connection or set of association, or a rule for communication among entities.

e.g. in college data base, the association between student and course entity. i.e." student opts course "is an example of relationship.

RELATIONSHIP CARDINALITY/MAPPING CONSTRAINT

*Cardinality specifies the no of instances of an entity associated with another entity, participating in a relationship.

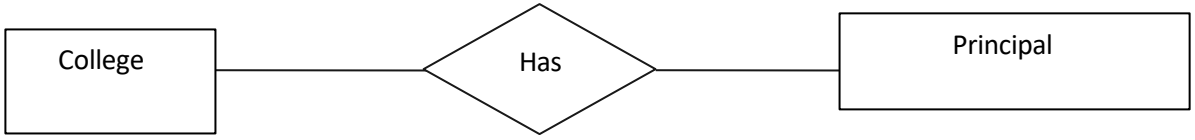
*Based on the cardinality binary relationship can be further classified into the following categories.

a>one----to---one
b>one---to---many
c>many---to---one
d>many---to---many

one----to---one

Only one instance of entity 'A' associated with at most one instant of entity B.

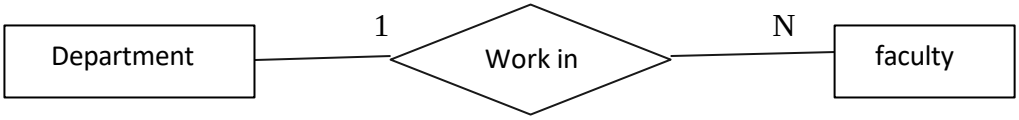
e.g.: Relationship between college and principal.



One college can have at most one principal and one principal can be assigned to only one college.

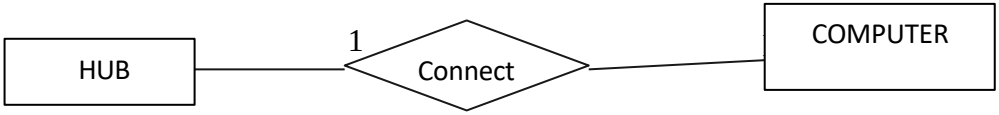
one---to---many

e.g. Relation between department and faculty.



One department can appoint any no. of faculty but a faculty is assigned to one department.

e.g. Relationship between HUB & COMPUTER.

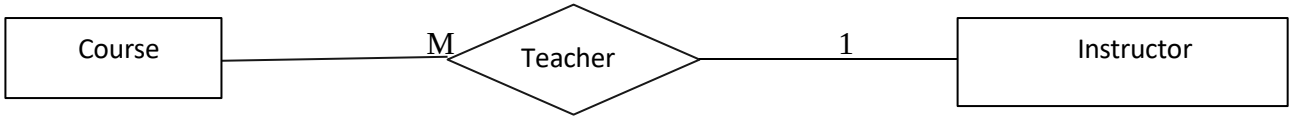


One HUB can connect to any no of COMPUTER but a computer can connect to any one HUB.

many---to---one

Any one of instance of entity A are associated with only one instance of entity B.

E.g. Relationship between course and instructor.

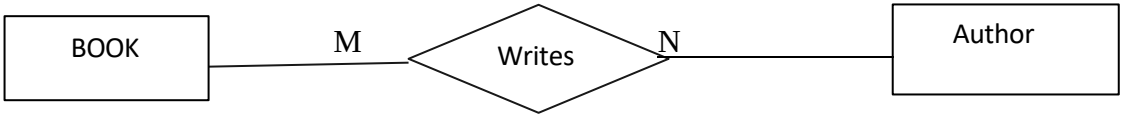


An instructor can teach various courses but a course can be taught only by one instructor.

many---to---many

Instances of entities A & B are associated with any number of instances from each other.

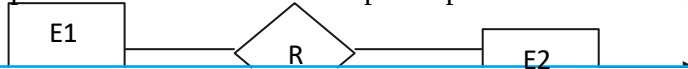
e.g. Relationship between book & author.



One author can write many books and one book can be written by more than one authors.

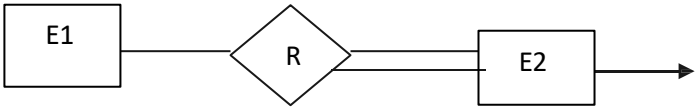
One author can write many books and one book can be written by more than one authors.

Partial participation: The participation of an entity set E in a relationship set R is said to be partial if some entities in E participates in relationships in R.



Partial participation of E2 in R

Total participation: The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R.

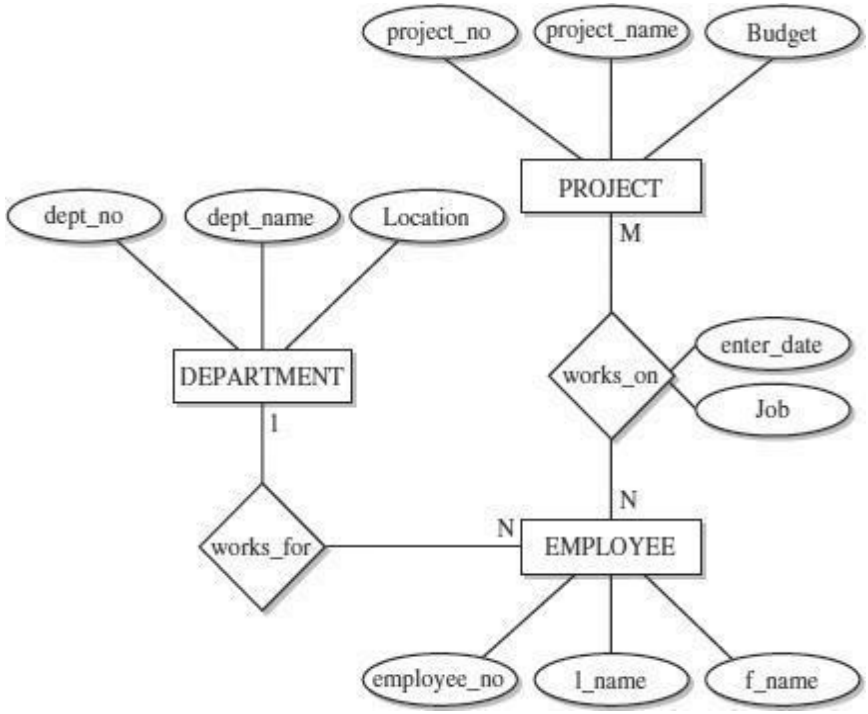


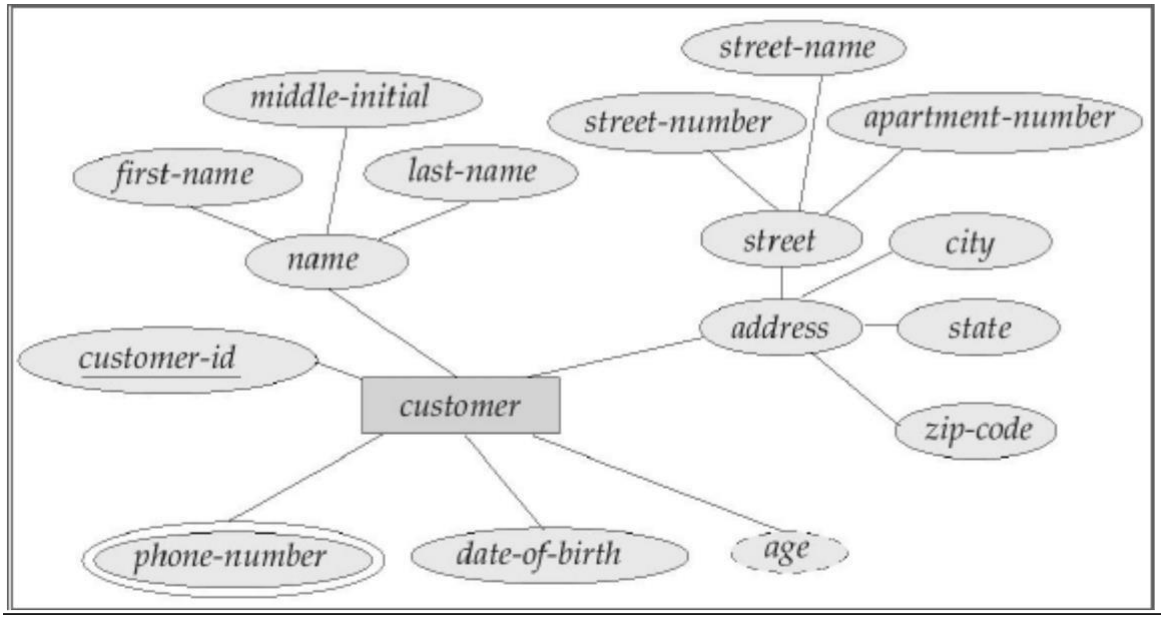
Total participation of E2 in R

E---R DIAGRAM (Entity-Relationship Diagram)

The overall logical structure of database can be expressed using E-R model graphically with the help of an E-R diagram.

Example:





RECORD BASED MODEL

A record based data model is used to specify the overall logical structure of the database.

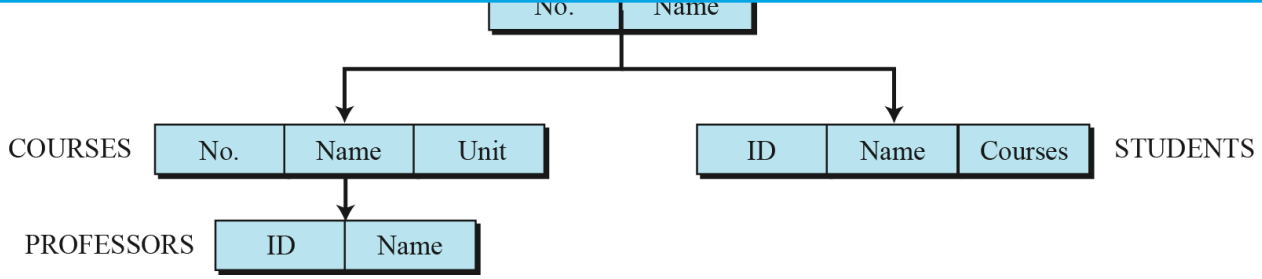
Each record type defines a fixed no. of fields having a fixed length.
This model describes data at the conceptual level or view level and provides higher level of description of the implementation.
This data model uses records as key data representation components.

There are many record based data model

1) Relational Model
2) Hierarchical Model
3) Network Model

HIERARCHICAL MODEL

- The hierarchical data model organizes data in a tree structure .
- Each entity has only one parent but can have several children. At the top of the hierarchy, there is one entity, which is called the root.
- Data and relationships among them is represented by records and pointers.
- This model efficiently describes many real-world relationships like index of a book, recipes etc.



ADVANTAGES

- Adding and deleting records is easy.
- Fast data retrieval through higher level records.
- Efficient for 1 to many relationship fixed over time.

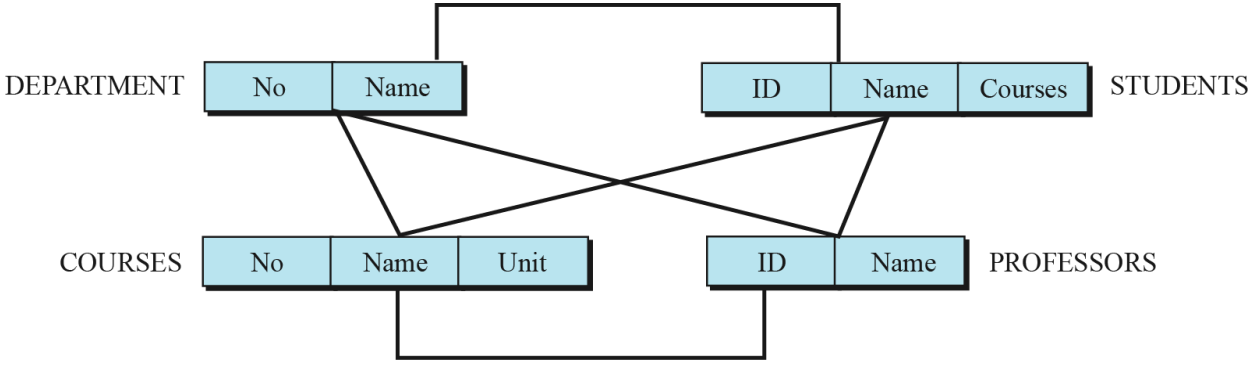
DISADVANTAGES

Management difficulties caused by parent segment deletion in that all child segments must also be deleted.

Pointers requires large amount of computer storage.

NETWORK MODEL

In the network model, the entities are organized in a graph, in which some entities can be accessed through several paths.



The data are represented by collection of records and relationship among the data represented by links.

The network model is an improvement of hierarchical model; here multiple parent child relationships are used.

In network model the relationship as well as the navigation through the data base is predefined at the data base creation time.

ADVANTAGES

Data independence.

It can handle more relationships than hierarchical e.g. a child can have multiple parents.

DISADVANTAGES

Detailed structural knowledge is required.

Lack of structural independence.

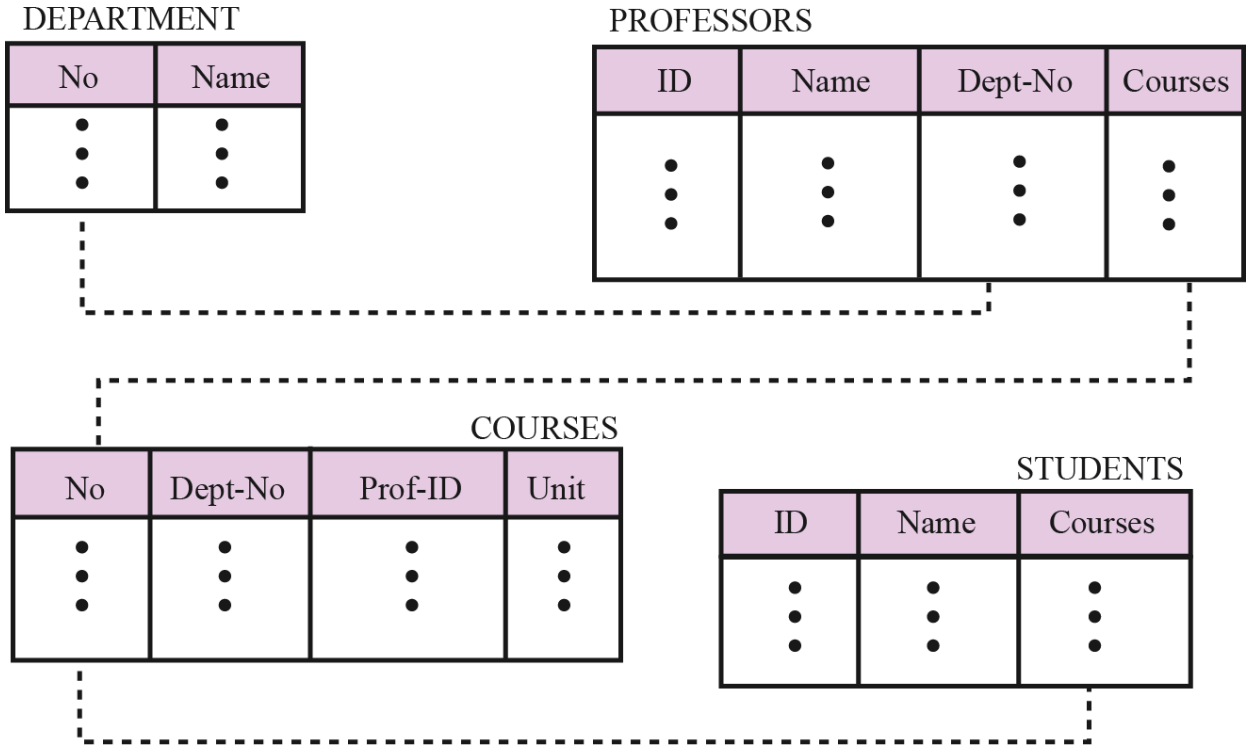
RELATIONAL MODEL

In the relational model, data is organized in two-dimensional tables called relations. The tables or relations are related to each other.

Constraints are stored in a meta-data table.

This is a very simple model and most widely used data base model.

The relational model is based on the relational algebra.



Relation: Each table in a database is known as relation.

Attributes: Each column in a relation is called an attribute. The attributes are the column headings in the table .

Domain: The set of all possible value of an attribute is known as domain of the attribute.

Name.: Each relation in a relational database should have a name that is unique among other relations.

Tuples. Each row in a relation is called a tuple.

ADVANTAGES

Avoids data duplication.

Avoids inconsistent records.

Easier to change data and data format.

Easier to maintain security.

Data can be added and removed easily.

DISASVANTAGES

New relations can require considerable processing.
Sequential access is slow.
Method of storage on disk impacts processing time.

KEY

A key is an attribute or a set of attribute in a relation that uniquely identifies a tuple in a relation.

Types of keys:

1. **Super Key:** A super key is an attribute or any set of attributes that uniquely identifies a row in a relation.
2. **Composite Key:** A composite key is a key that contains more than one attribute.
3. **Candidate Key**: A candidate key is an attribute or set of attribute that uniquely identifies a row in a relation. In other words minimal super key is called candidate key.
4. **Primary Key**: A primary key is a candidate key which can uniquely identify a record or tuple. Each table can have only one primary key. The primary key should be selected in the manner such that it is unique and not null. .
5. **Alternate Key**: The alternate keys of any table are those candidate keys which are not selected as the primary key.

   AK(Alternate Key)=CK(Candidate Key)-PK(Primary Key)
6. **Overlapping Key**: The keys having common attributes are called overlapping key.
7. **Foreign Key**: Foreign key is a referential key which must be a primary key of another table. This way we can link two tables for retrieving the data jointly. we only insert those values which are present in the base table. If we delete the base table automatically the the foreign key and primary key relationship is broken.

Chapter-3
RELATIONAL DATABASE

Relational Terminology

RELATION
Roughly a table in a relational database is called a relation.
A relation consist of

1. Relational scheme
2. Relational instances.

Attribute
The name of each column in a database is used to interpret its meaning is called attribute.
Domain
Each attribute is defined over a set of values known as its domain.
Tuple
A tuple is a row in a relation\table.
Degree/Arity
The no. of attributes in the relation scheme is called its degree or arity.
Cardinality
No. of tuples present in a relation scheme is called cardinality of a relation.
Union Compatible
Two relation P[P] and Q[Q] are said to be union compatible, if both P & Q are of same degree and the domains of the corresponding attributes are equal.
i.e. if $P=\{P_1,P_2 \ldots\ldots\ldots\ldots P_n\}$ & $Q= \{Q_1,Q_2\ldots\ldots\ldots Q_n\}$ then DOM(Pi) = DOM(Qi) for $i=\{1,2,3,4\ldots\ldots n\}$,
where DOM(Pi) represents the domain of attribute Pi.

RELATIONAL ALGEBRA

Relational algebra is a collection of operation used to manipulate the data in relational model
NOTE: Result of each relation operation is also a relation.
→The operation can be classified in two categories.
1. Basic set operation
   a) Union
   b) Intersection
   c) Set difference
   d) Cartesian product
2. Relational operation
   a) Selection
   b) Projection
   c) Join
   c) Division.

BASIC SET OPERATION

→These are binary operations (i.e. each is applied to two relations)
→These two relations should be union compatible expect in case of Cartesian product.

Example

R1

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

$R3 = R1 \cap R2$ is

A	B
A1	B1
A2	B2
A4	B4

UNION

→If R1 & R_2 are two union compatible relations then the union of R_1 & R_2 ($R_3 = R_1 \cup R_2$) is the relation containing tuples that are either in R_1 & R_2 or in both of them.
→The duplicated tuples such that
$R3 = \{t \mid R1 \in t \vee R2 \in t\}$ and $\max(|R1|, |R2|) \leq |R3| \leq (|R1| + |R2|)$

→The cardinality of the resultant relation depends on duplication of tuple in R_1 & R_2.
If the tuples in R_1 & R_2 are disjoints, then $|R3| = |R1| + |R2|$

NOTE

- Union is a commutative operation i.e. $R_1 \cup R_2 = R_2 \cup R_1$
- Union is an associative operation i.e. $R_1 \cup (R_2 \cup R_3) = (R_1 \cup R_2) \cup R_3$

INTERSECTION

- If R1 & R2 are two union compatible relation the R=R1 …R2 is the relation that includes all tuples that are in both the relations.

- In other words R3 will have tuples such that
  $R3 = \{t \mid R1 \in t \wedge R2 \in t\}$.and $0 \leq |R3| \leq \min(|R1|, |R2|)$

Intersection is really unnecessary .it can be simplified R1..R2=R1(R1-R2)

Example

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

$R3 = R1 \cup R2$ is

Q

A	B
A1	B1
A2	B2
A3	B3
A4	B4
A7	B7

Note: 1) Intersection is a commutative operation, i.e.,
$R1 \cap R2 = R2 \cap R1$.
2) Intersection is an associative operation, i.e.,
$R1 \cap (R2 \cap R3) = (R1 \cap R2) \cap R3$

SET DIFFERENCE

If R1 & R2 are two union compatible relations, The results of R=R1-R2 is the relation that includes only those tuples that are in R1 but not in R2 .

In other words we can say it removes that common tuples from the first relation.

R3 will have tuples such that

$R3 = \{t \mid R1 \in t \wedge t \notin R2\}$. and $0 \leq |R| \leq |R1|$

Note: -1) Difference operation is not commutative,

i.e., $R1 - R2 \neq R2 - R1$

2) Difference operation is not associative,

i. e.,$R1 - (R2 - R3) \neq (R1 - R2) - R3$

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R1-R2 =

A	B
A3	B3

R2-R1=

A	B
A7	B7

CARTESIAN PRODUCT

- The Cartesian product of 2 relation is concatenations of tuples belonging to two relations.
- A new resultant relation scheme is created consisting of all possible combinations of tuples that in the both relations.
- R3 = R1 × R2 where a tuple is given by:
  R3 = {t1 || t2 | R1 ∋ t1 ∧ R2 ∋ t2}.
- R3 is obtained by concatenation each tuple in relation R1 with each tuple in relation R2 here || represents the concatenation operation.
  Degree (R3)=(Degree(R1)+Degree(R2)
- The cardinality of resultant relation is given by |R|=|P|*|Q|

R1

C
C1
C2

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

$R3 = R1 \times R2$ is

A	B	C
A1	B1	C1
A1	B1	C2
A2	B2	C1
A2	B2	C2
A3	B3	C1
A3	B3	C2
A4	B4	C1
A4	B4	C2

Relational Operations

Select Operation

- The select operation is used to select some specific records from the database based on some criteria.
- This is a unary operation mathematically denoted as σ

Syntax:

σ **<Selection condition> (Relation)**

The Boolean expression is specified in <Select condition> is made of a number of clauses of the form:

<attribute name><comparison operator><constant value> or

<attribute name><comparison operator><attribute name>

Comparison operators in the set { ≤, ≥, ≠, =, <, <} apply to the attributes whose domains **are *ordered value* like integer.**

Example :

Consider the relation PERSON. If you want to display details of persons having age less than or equal to 30 than the select operation will be used as follows:

σ **AGE <=30 (PERSON)**

The resultant relation will be as follows:

PERSON_ID	NAME	AGE	ADDRESS
2	Sharad Gupta	30	Pocket 2, Mayur Vihar.

Note:

1) Select operation is commutative; i.e.,

σ <condition1> (σ <condition2> (R)) = σ <condition2> (σ <condition1> (R))

Hence, Sequence of select can be applied in any order

2) More than one condition can be applied using Boolean operators AND & OR etc.

PROJECT Operation

The project operation is used to select the records with specified attributes while discarding the others based on some specific criteria**.**

This is denoted as Π**.**

Π List of attribute for project (Relation)

Example :

Consider the relation PERSON. If you want to display only the names of persons then the project operation will be used as follows:

Π Name (PERSON)

The resultant relation will be as follows:

NAME
Sanjay Prasad
Sharad Gupta
Vibhu Datt

Selection+ Projection

For example:

Find those students who live in Bhopal from student relation.

Πsname(σsaddress=" Bhopal" (student));

JOIN:-

The join operator joins two or more relations to form another relation.

The join operator joints two relations on the basis of some comparison operator in meaningful way.

Syntax:- R1 ⋈ R2

x Θy

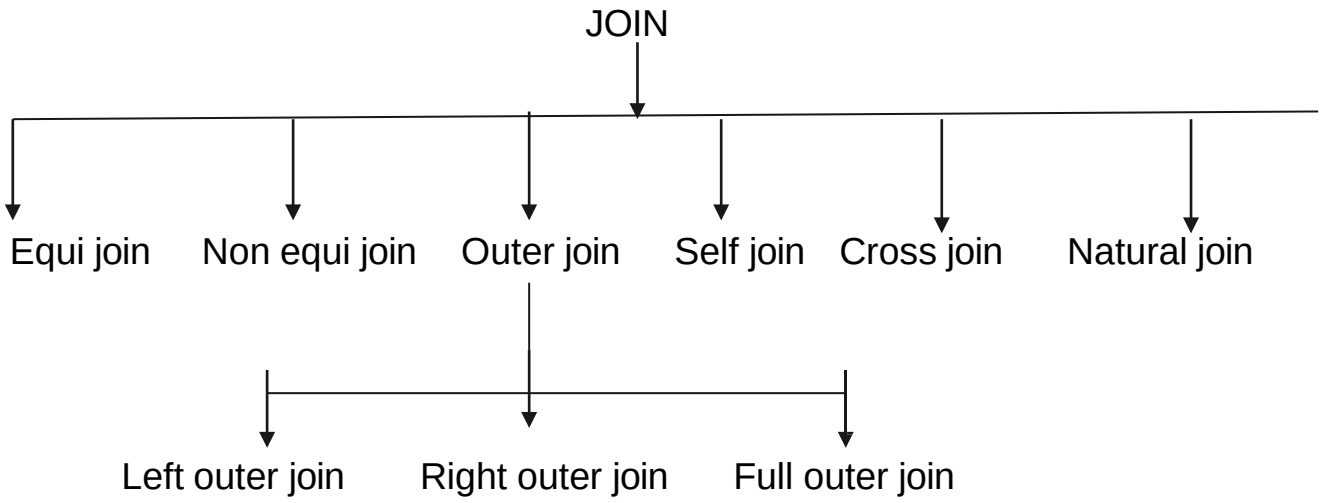
Where R1, R2→ Two relations

X → Attributes of R1

Y → Attributes of R2

⋈ →join operator

Θ → Comparison operator



1. Equi join:-

- When there is a common attribute in two relations can be joined by equating the common column value.
- When the comparison operator is an equal operator then the join is called equi join.

STUDENT

Name	Roll No	Class
Sita	1	3rd CS
Rama	4	5th CS
Gita	9	3rd ETC
Swayam	10	5th CS
Rabi	14	3rd Elect

HOSTEL

Name	Hostel	Room No
Sita	L-H	1
Rabi	B-H	4
Gita	L-H	2
Ganga	L-H	2

If we join the relation student with relation hostel then the resulting will as follows.

Student ⋈ Hostel

Student.Name = Hostel.Name

Name	Roll no	Class	Hostel	Room no
Sita	1	3rd CS	L-H	1
Gita	9	3rd ETC	L-H	2
Rabi	14	3rd Elect	B-H	4

2. Non-Equi join:-

A non-equi join is the join where comparison operator is other than equal operator. The non equi join takes place where there is no common attribute is present in both relations.

Student

Roll no	Name	Mark
1	XXX	55
2	YYY	35
3	ZZZ	69
4	PPP	25

Grade

Lower	Upper	Division
60	100	1st
50	60	2nd
30	50	3rd
0	30	fail

If we want to retrieve the relation , what is the grade of each student then we have to join the two relation as follows

Student ⋈ Grade

(Marks between lower and upper)

Roll no	Name	Marks	Division
1	XXX	55	2nd
2	YYY	35	3rd
3	ZZZ	69	1st
4	PPP	25	Fail

3. Outer Join:-

Sometime we do not want to lose any information in the join operation and where this can be possible is called outer join. The use of Outer Join is that it even joins those tuples that do not have matching values in common columns are also included in the result table. Outer join places null values in columns where there is not a match between them.

Outer join is of three types.

A) Left outer join,

B) Right outer join,

c) Full outer join

Student

Name	Roll no	Class
Sita	1	3rd CS
Rama	4	5th CS
Gita	9	3rd ETC
Swayam	22	5th CS
Rabi	10	3rd Elect

Hostel

Name	Hostel	Room no
Sita	LH	1
Gita	LH	2
Rabi	BH	4
Ganga	LH	2

A) Left outer join:-

- In this type of outer join the resulting relation consists of all tuples of the left relations and the tuples of right relation which satisfy the condition.
- If we join the relation student with hostel through left outer join then the result will be as follows:

Student ⟕ Hostel

Student.Name = Hostel.Name

Name	Roll no	Class	Hostel	Room no
Sita	1	3rd CS	LH	1
Gita	9	5th CS	LH	2
Rama	4	3rd ETC	NULL	NULL
Swayam	22	5th CS	NULL	NULL

Rabi	10	3rd Elect	BH	4

B) Right outer join:-

In this type of outer join the resulting relation consists of all tuples of the right relations and the tuples of left relation which satisfy the condition.

If we join the relation student with hostel through left outer join then the result will be as follows:

Student ⟕ Hostel

Student.Name = Hostel.Name

Name	Roll no	Class	Hostel	Room no
Sita	1	3rd CS	LH	1
Gita	9	3rd ETC	LH	2
Rabi	10	3rd ELECT	BH	4
Ganga	NULL	NULL	LH	2

C) Full outer join:-

Here all the tuples of both relations are present in the resulting relation. If we join the relation student with hostel through full outer join then the result will be as follows:

Student ⟗ Hostel

Student.Name = Hostel.Name

Name	Roll no	Class	Hostel	Room no
Sita	1	3rd CS	LH	1
Rama	4	5th CS	NULL	NULL
Gita	9	3rd ETC	LH	2
Swayam	22	5th CS	NULL	NULL
Rabi	10	3rd Elect	BH	4
Ganga	NULL	NULL	LH	2

4. Cross Join -:

This type of join is rarely used as it does not have a join condition, so every row of table 1 is joined to every row of table 2. For example, if both tables contain 100 rows the result will be 10,000 rows.

The size of a Cartesian product is the number of the rows in first table multiplied by the number of rows in the second table. This is sometimes known as a Cartesian product and can be specified in either one of the following ways:

SELECT * FROM table1 CROSS JOIN table2
SELECT * FROM table1, table2

5. **Natural Join**: It is the same like Equi join except one of the duplicate columns is eliminated in the result table. The natural join is the most commonly used form of join operation.

Courses		
CID	**Course**	**Dept**
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HoD	
Dept	**Head**
CS	Alex
ME	Maya
EE	Mira

Courses ⋈ HoD			
Dept	**CID**	**Course**	**Head**
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

6. **Self-Join**: It is a join operation where a table is joined with itself. Consider the following sample partial data of EMP table:

EMPNO	ENAME	MGRID	…..
1	Nirmal	4	
2	Kailash	4	
3	Veena	1	
4	Boss	NULL	
…..	…..	…	

The DIVISION operation:

To perform the division operation R1 ÷ R2, R2 should be a proper subset of R1. In the following example R1 contains attributes SNo and PNo and R2 contains only attribute PNo so R2 is a proper subset of R1. If we perform R1 R2 than the resultant relation will contain those values of SNo from R1 that are related to all values of PNo present in R2.

If R(x)=R1(z) ÷ R2(y)

The relation R(x) is having all the tuples t(x) in R1(z) that appears in R1 in combination with every tuple from R2(y).

Where Z= X U Y

R1

SNo	PNo
S1	P1
S1	P3
S1	P2
S2	P1
S1	P4
S2	P2
S4	P1
S4	P4

R2

PNo
P1
P4

The result of R1 ÷ R2 will be

SNo
S1
S4

Note:Degree of relation: Degree (R ÷ S)=Degree of R – Degree of S.

Chapter-4

NORMALIZATION IN RELATIONAL SYSTEM

Database anomalies.

- The real world is model in database through number of relations, since the real world changes over time, the tuples changes throughout.
- Those tuples may be added, deleted or updated and this frequent updation leads to some anomalies (problem). These anomalies arise due to the bad design.

The anomalies are:-

Updated anomalies -

The multiple copies of some data may leads to updates anomalies or inconsistency.
When an update is made and only some of the multiple copies are updated.

Insertion anomalies:-

Sometimes the insertion is depends upon another if the dependent relation values are inserted of inserting to the relation on which it is dependent, then it leads to bad database design.

Deletion anomalies:-

If the relations are dependent with each other then deletion of tuples in one relation should be updated and stored in other relation otherwise it leads to deletion anomalies.

Functional dependency

A functional dependency is a particular relationship between two attributes.

Given a relation R, a set of attributes X in R is said to functionally determine another attribute Y, also in R, (written $X \rightarrow Y$) if and only if each X value is associated with at most one Y value.

Let R be a relation and x & y are two non empty set of attribute in R then a relation instance has a functional; dependency $x \rightarrow y$, if the following conditions holds for every pair of following conditions holds for every pair of tuples of t1 &t2 is R.

If $t_1[x]=t_2[x]$ then $t_1[y]=t_2[y]$

$x \rightarrow y$ denote: - x functionally determine y or y functionally dependent on x

- Functional dependency does not imply mathematical dependence that the value of the attributes may be computed from value of another attribute.
- Functional dependent of y on x means that there can be one value of Y for each value of x.

- An attribute may be functionally dependent on two or more attributes rather than a single attribute or vice versa.
- Functional dependencies are consequences of interrelationship among attributes of an entity represented by a relation or due to relationship between the entities i.e. also represented by a relation.

EXAMPLE:-

Emp-id→Ename

Emp-id→Dept, addr

Lets x= {roll no}

y= {name, branch}

x→y

NOTE

The values of all attributes are functionally dependent on primary key of a relation.

INFERENCE RULE OF FDS

For a govern relationship there are many FDS.

let F->set of FDS that are given for a relation R.

There are many other FDS that can be derived from the given FDS and is called as infered.FDS is denoted by F^+.

The set of all dependencies that include F as well as dependencies that can be inferred from F is called closure of F is denoted as F^+.

EXAMPLE:-

F= {Emp -id →ename, addr, dept-no}

Let x= Emp –id, y= dept-no

Dept-no-> {dept-name, mgr no}

x→y

y→z

Then

x→z inferred dependency

F^+={emp-id →dept –name}

to inferred the FDS from F, there are certain rules which are describes below.

RULES(AMSTRONGS AXIOMS)

1)**REFLEXIVE**

IF (X⊃Y) then automatically x derives y or x→y

2)**AUGMENTATION**

IF X→Y then xz →yz

3) **TRANSITIVE**

If(x→y)&(y→z) Then (x→z)

If x determine y & y determine z then automatically x determines z.

4)**DECOMPOSITION**
IF (X→YZ) then (x→y) & (x→z)
if x determines y & z combinedly then automatically x determines y &x determines z

5)**UNION**

If (x→y) (x→z) Then(x→yz)

If x determines y &z independently then x can be determines y & z combinedly.

6) PSEUDOTRANSITIVE

If x→y & yz→p then xz→p

Let X=Rollno, y=Name , z= Addr. , p= ph.no

Rollno-→name, name,addr.→Ph.no

Then address, rollno→Ph.no

NOTE

The first three rule (reflexive, augmentation, transitive)are fundamental axioms and are sufficient to generate all possible FDS.

Example

Let R(A,B,C,D,E,F) & the sets of FDS are given below :

F→[{A→BC},{B→E},{CD→EF}]

Ans:

(1) *A→BC*

⇒A→B & A→C (Decomposition)

(2) *A→B & B→E*

⇒A→E (Transitive)

(3) *CD→EF*

⇒E & CD→F (Decomposition)

(4) *A→BC & A→E*

⇒ A→BCE(Union law)

(5) *A→C & A→E*

⇒A→CE(Union law)

F+=[{A→BC},{B→E},{CD→EF},{A→B},{A→C},{A→E},{CD→E},{CD→F},{A→BCE},{A→CE}]

EXAMPLE

Let R = (A,B,C,D) & F={A→B,A→C,BC→D}. Find F+.

Answer

(1) A→B & A→C

A→BC(Union)

(2) A→BC & BC→D

A→D (Transitive rule)

(3) A→B & A→D

A→BD (Union)

(4) A→AC & A→D

A→CD (Union)

(5) A→BC & A→D

A→BCD (union)

F+=[{A→B},{A→C},{BC→D},{A→BC},{A→BC},{A→D},{A→BD},{A→CD},{A→BCD}]

PARTIAL DEPENDANCY:

- A dependency in which one or more non-key attributes are functionally dependant on part but not all of the primary key.

OR

- If primary key is a composite key, some of the attributes will depend on the primary key then those attribute are called partially depends upon the primary key.

EXAMPLE

Stud{s-Id, Name, DOB, Addr, Course, Date of completion}

The FDS are:-

S-id:- Name,DoB,ADDR,Course

S-id:- date of completion

The primary key of the relation is the composite key i.e **s-id,course**, therefore the non-key attributes name, dob, address are functional dependant on only s-id, but not on course. Hence, these are partial dependant.

NORMALIZATION

- **Normalization** is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like insertion. deletion and modification.
- The poor database design gives rise to many anomalies. This anomalies can be avoided by a process called Decomposition process or **"Normalization".**
- The guidelines for constructing relation scheme having desirable property and avoiding anomalies are called normal forms.
- Normalization is a process of designing the scheme in an appropriate normal from that can allow to retrieve information easily.

Advantages of normalization

- Improve database design.
- Ensures minimum redundancy of data.
- Removes anomalies for database activity.
- Eliminate the problems of flow data querying and using memory space usage.

There are many normal forms based on there of functional dependency:-

1. First normal form (1NF)
2. Second normal form (2NF)
3. Third normal form (3NF)
4. Boyee codd normal form (BCNF)
5. Fourth normal form (4NF)
6. Fifth normal form (5NF)

FIRST NORMAL FORM

A relation schema is in first normal form if the values in the relation instances are atomic for every attribute in relation.

OR

A table will be in first normal form when each cell in the table contains precisely one value that means multivalued attribute is not allowed.

EX: -

We might have to store data item, colors, price and tax. As every item can have more than one color so, the arrangement might look like this table.

item	colors	price	tax
T-shirt	red, blue	12.00	0.60
polo	red, yellow	12.00	0.60
sweatshirt	blue, black	25.00	1.25

The Table is not in first normal form because Multiple items present in color field.
To make the table in 1NF we should have the data like this.

item	colors	price	tax
T-shirt	red,	12.00	0.60
T-shirt	blue	12.00	0.60
polo	red,	12.00	0.60
polo	yellow	12.00	0.60
sweatshirt	blue,	25.00	1.25
sweatshirt	black	25.00	1.25

SECOND NORMAL FORM (2NF)

A relation schema is in second normal form
If it is in first normal form.
If non-key attribute is not functionally dependant on part of the key attribute (i.e. **no partial dependency**) i.e. if the non key attribute is fully functional dependent of key attribute.

Example :

item	colors	price	tax
T-shirt	red,	12.00	0.60
T-shirt	blue	12.00	0.60
polo	red,	12.00	0.60
polo	yellow	12.00	0.60
sweatshirt	blue,	25.00	1.25
sweatshirt	black	25.00	1.25

Now the table is in 1NF because each field contain single attribute. Here we have taken {item, color} as primary key. However the table is not in 2NF because the non key attribute price and tax depend only on item, but not on color which is a part of the primary key. This violates the rule for 2NF as the rule says no non-key attribute is functionally dependant on part of the key attribute.

To make the table complies with 2NF we can break it in two tables like this:

item	color
T-shirt	red,
T-shirt	blue
polo	red,
polo	yellow
sweatshirt	blue,
sweatshirt	black

Item	price	tax
T-shirt	12.00	0.60
sweatshirt	25.00	1.25
polo	12.00	0.60

Now the table is in 2NF.

3NF (THIRD NORMAL FORM)

A relation R is said to be 3NF.

- If it is in 2NF.
- All non-key attribute are functionally depends on the key attribute(primary key)in other words we can say on non-key attribute of relation R is **not transitively depend** on the key attribute.

Item	price	tax
T-shirt	12.00	0.60
sweatshirt	25.00	1.25
polo	12.00	0.60

In this table **item** is the Primary key. The non key attribute **tax** depends on **price** not on **item.**
Here item→price and price → tax. So item →tax. That means transitive dependeny exists here.
Therefore the table is not in 3NF.
To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency.

Item	price
T-shirt	12.00
sweatshirt	25.00
polo	12.00

price	tax
12.00	0.60
25.00	1.25

Now the tables are in 3NF.

BOYCE CODD NORMAL FORM (BCNF)

- It is slightly stronger version of 3NF.

A table is in BCNF if and only if, it is in 3NF and for every functional X->Y, X should be the super key of the table..

- The BCNF is a special case of 3NF and BCNF can only be violated if a 3NF complaint table contain more than one candidate key.

CHARACTERISTICS

- 3NF (special case)
- Every determinant is a candidate key.

Example: Suppose there is a company wherein employees work in **more than one department**.

They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:

emp_id -> emp_nationality

emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
Stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	Stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:

emp_id -> emp_nationality

emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:

For first table: emp_id

For second table: emp_dept

For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

Chapter-5
STRUCTURED QUERY LANGUAGE

QUERY LANGUAGE:-

- A query language is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language.
- Query languages can be categorized as either procedural or non-procedural. In a procedural language the user instructs the system to perform a sequence of operations on the database to compute the desired result. In a non-procedural language the user describes the desired information without giving a specific procedure for obtaining that information.

SQL:

- Structured Query Language (SQL) is a standard query language.
- It is commonly used with all relational databases for data definition and manipulation.
- SQL is called a non-procedural language as it just specifies what is to be dome rather than how it is to be done.
- since SQL is a higher-level query language, it is closer to a language like English.
- Therefore, it is very user friendly.
- The American National Standard Institute (ANSI) has designed standard versions of SQL.

Some of the important features of SQL are:

- It is a non procedural language.
- It is an English-like language.
- It can process a single record as well as sets of records at a time.
- It is different from a third generation language (C& COBOL). All SQL statements define what is to be done rather than how it is to be done.
- SQL is a data sub-language consisting of three built-in languages
- Data definition language (DDL), Data manipulation language (DML) and Data Control language (DCL).
- It insulates the user from the underlying structure and algorithm.
- SQL has facilities for defining database views, security, integrity constraints, transaction controls, etc.

Data definition language (DDL)

- The Data definition language (DDL) defines a set of commands used in the creation and modification of schema objects such as tables, indexes, views etc.
- These commands provide the ability to create, alter and drop these objects.
- These commands are related to the management and administrations of the databases.
- Before and after each DDL statement, the current transactions are implicitly committed, that is changes made by these commands are permanently stored in the databases.

CREATE TABLE Command

This command creates a table in the data base.

Syntax:

create table tablename (colname datatype(size), colname datatype (size),……, colname datatype (size));

Example:-

Create table student(rollno varchar2(20), name varchar2(30), address varchar2(50),semester varchar2(10));

To view the structure of a table:-

Syntax:-

Sql> desc table name;

ALTER TABLE Command:

This command is used for modification of existing structure of the table in the following situation:

- When a new column is to be added to the table structure.
- When the existing column definition has to be changed, i.e., changing the width of the data type or the data type itself.
- When integrity constraints have to be included or dropped.
- When a constraint has to be enabled or disabled.

Modifying the structure of a table:-

To add a new column:-

Syntax:-

Sql> alter table table name add (column1 (data type (size)), (column2 (data type (size))) ;

Example:-

Alter table student add (dob date);

To drop a column from a table:-

Syntax:-

Sql> alter table table name drop column name;

Example:-

Alter table student drop dob;

To modify columns from a table:-

Syntax:-

Sql> alter table table name modify (column name (new data type (new size)));

Example:-

Alter table student modify(name varchar2(50));

DROP TABLE Command:

When an existing object is not required for further use, it is always better to eliminate it from the database. To delete the existing object from the database the following command is used.

Syntax:

DROP TABLE<table name>;

Example :

DROP TABLE emp;

Example:-
Create table student(rollno varchar2(20), name varchar2(30), address varchar2(50),semester varchar2(10));

To view the structure of a table:-

Syntax:-
Sql> desc table name;

DATA MANIPULATION LANGUAGE (DML)

Data manipulation language (DML) defines a set of commands that are used to query and modify data within existing schema objects.
In this case commit is not implicit that is changes are not permanent till explicitly committed.
DML statements consist of SELECT, INSERT, UPDATE and DELETE statements.

SELECT Statement

This statement is used for retrieving information from the databases. It can be coupled with many clauses.

Syntax:

SELECT <column_list> FROM <table_name>;

Example:
SELECT Rollno, name FROM Student;

INSERT INTO command:

Values can be inserted for all columns or for the selected columns
Values can be given through sub query.
If data is not available for all the columns, then the column list must be included following the table name.
While inserting a single row of data into a table, the insert operation first creates a new row (empty in the database table) and then loads the value passed into the column specified.

Syntax:-

Insert into table name (col1, col2, col3,……., coln) values (expr1, expr2, expr3, ..., exprn);

Example:

Insert into student (‘f1201207005’,’radhika’,’bbsr’,’3rd cse’);

Example : Insert the employee numbers, an increment amount of Rs.500/- and the increment date-today (which is being entered through function SYSDATE) of all the managers into a table INCR (increment due table) from the employee file.

INSERT INTO INCR
SELECT EMPNO, 500, SYSDATE FROM EMP
WHERE JOB = ‘MANAGER’;

UPDATE Command:

The update command is used to change or modify data value of a table.
Syntax :
UPDATE <table name> Set <column name> = <value>Where <condition>;
Example:

UPDATE student SET addr=”etc” Where rollno=10;

DELETE Command

This command is responsible for deleting certain values of a relation those satisfies the conditions.

To remove all the rows from a table the syntax is as follows

Syntax:-

Sql> delete from table name;

Example:-

delete from student;

To remove a set of rows from a table the syntax is as follows

Syntax:-

Sql> delete from table name where <condition>;

Example:-

delete from student where name=’radhika’;

Example : Delete the records of employees who have got no increment.

DELETE FROM EMP

WHERE EMPNO NOT IN (SELECT EMPNO FROM INCR);

DATA CONTROL LANGUAGE (DCL)

The data control basically refers to commands that allow system and data privileges to be passed to various users. These commands are normally available to database administrator. Let us look into some data control language commands:

Create a new user:

CREATE USER < user name > IDENTIFIED BY < Password>

Example:

CREATE USER MCA12 IDENTIFIED BY W123

1. **Grant:** This command grant users various privileges to table .It is used to provide database access permission to users. It is used to allow specified user to perform specified task . Privileges can be the combination of select, update, delete, alter. It is of two types

(1) System level permission (2) Object level permission.

System level permission

Syntax

GRANT CREATE SESSION TO USER;

Example:

GRANT CREATE SESSION TO CSE;

Object level permission

Syntax

GRANT <previllege_list> ON <table_name> to user;

Example:

GRANT select On student to CSE;

2. **Revoke:** It is used to cancel the permission granted.
Syntax :

REVOKE <previllege_list> ON <table_name> FROM user

Example:
REVOKE ALL ON EMP FROM CSE;
(All permissions will be cancelled)
We can also revoke only some of the permissions.

Drop: A user-id can be deleted by using drop command.
Example:DROP USER CSE;

Chapter-6
TRANSACTION PROCESSING CONCEPTS

TRANSACTION

- Transaction is a programming unit of work whose execution may change the database.
- It consists of a set of operation including reading, writing & modifying the database objects.
- Thus a transaction is a program until whose execution preserves the consistency of the database is in a consistent state before a transaction is executed, and then the database is also in a consistent state after execution.
- A **read operation** brings a database objects into main memory from disk where the database is residing & then the value is copied to a program variable defined within the transaction.
- A **write operation** identifies a copy of the data base object in the main memory, updates its value & then writes this objects to the disk.

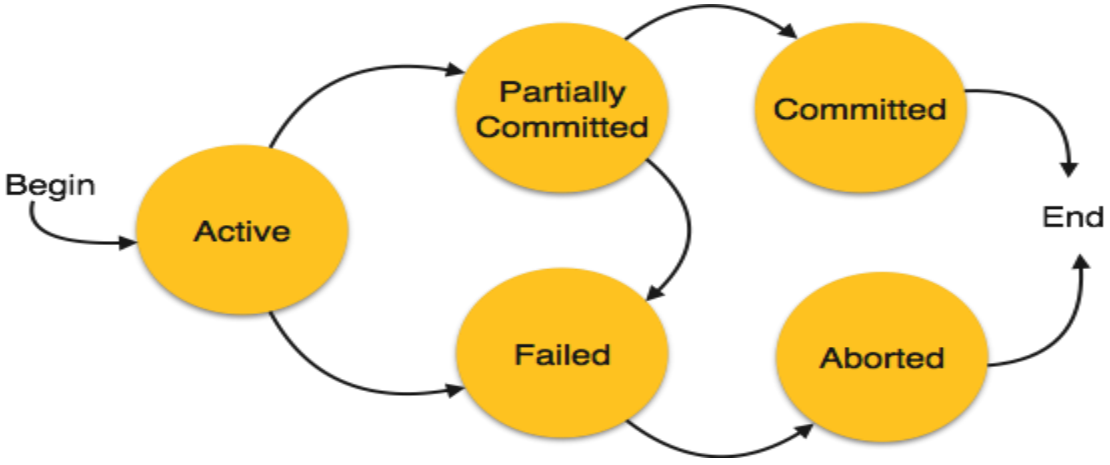
Example:

Ti is a transaction that transfers Rs 50 from account A to account B. This transaction can be defined as:

T1 Schedule
Read (A,a)
a: =a-50;
Write (A,a)
Read(B,b)
b: = b+50;
Write (B,b);

States of Transactions:

A transaction in a database can be in one of the following states −



- **Active** − In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** − When a transaction executes its final operation, it is said to be in a partially committed state.
- **Failed** − A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- **Aborted** − If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts −

  i) Re-start the transaction

  ii) Kill the transaction

- **Committed** − If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

PROPERTIES OF TRANSACTION (ACID properties)

There are 4 properties of transactions.

1. **ATOMICITY** :-A transaction is said to be atomic when all the operation of the transaction are either executed at a time or none of them is executed.

2. **CONSISTENCY :**

- Each transaction must preserve the consistency of data base.
- That is state before the transaction beings its execution & the database remains in a consistent state after the execution of transaction. .

3. **ISOLATION :**

- A transaction executes in isolation without any interference from any other transaction. It should not make visible to other transaction until it is successfully completed.
- Transaction should be visible till they are committed.

4. **DURABILITY:**

- Once a transaction is successfully completed its effects should never be lost even if the system crashes before all changes are reflected in the disk.Then effect should never be lost even in subsequent system failure.

SCHEDULE

- A **schedule** is a process of grouping the transactions into one and executing them in a predefined order.
- A schedule is required in a database because when some transactions execute in parallel, they may affect the result of the transaction.
- Hence a schedule is created to execute the transactions.

Types of schedule:

1. **Serial schedule**: when the actions of different transactions are not interleaved (i.e. transactions are executed from start to finish)then the schedule is called as serial schedule.
It always ensures a consistent state.

2. **Non-Serial schedule**: A schedule in which the operations from a set of concurrent transactions are interleaved is called as a non-serial schedule.
A non-serial schedule is acceptable if the final state produced by a non-serial schedule is equivalent to some state produced by a serial schedule.

3. **Equivalent schedule:** Two schedules are said to be equivalent schedules if they produce the same result on any database state.
Equivalent schedules always produce identical results.

4. **Serializable schedule:** A non-serial schedule that is equivalent to some serial execution of transactions is known as serializable schedule.

5. **Complete schedule**: A schedule that contains either an abort or commit for each transaction whose actions are listed in it is called as a complete schedule. A complete schedule must contain all the actions of every transactions that appears in it.

RECOVERABILITY

There are 2 main types of damage happened to the data base.

Type 1:When there is catastrophic error to the database. Ex:-when the hard disk crashes.

- The only way to recreate the data is if we have a copy of it stored as backup.
- During back up the entire database and the database transaction logs are copied to an alternative storage medium.

Type 2: there are several reasons for a transactions to fail which may lead to errors in database. This is called as non-catastrophic errors.

There are 2 mechanisms to do such error recovery.

- Deferred update method
- Immediate update method

Deferred update method

- In this method, the physical data on a database is not changed until the entire transactions is completed successfully.
- The successful completion of the transaction is called COMMIT point.
- During commit all update are stored in a file called as database log file.
- After committing the contents of the log are copied to the database.
- If the transactions fails before reaching the commit point there is no need to UNDO any data in database.
- Therefore, this method is called **NO UNDO/REDO METHOD.**

Immediate update method

- The dbms which use this method changes can be made on the database while a transaction is still being processed.
- However, before making any change in database the change is recorded in the log file.
- Only fter recording the change in log file the database is modified at each step.
- In this acse, if the tranasactions fails at some intermediate point then all updates perform before the fail point must be rolled back i.e. we need to UNDO this change.
- Next, to complete the transactions we must REDO the entire transactions.
- Therefore this method is called **UNDO/REDO method**.

Chapter-7
CONCURRENCY CONTROL CONCEPTS

If only one transaction to be executed at a time in serial order ,then performance will be quite poor.

- Then to improve the performance we can execute the transaction concurrently .But due to concurrent execution the lost update problem ,incorrect summary anomaly and dirty reads occurs.
- To recover from these problems we have to control the concurrency.
- If concurrency is controlled properly then the transaction through put can be maximized while avoiding the corruption of the data base.
- The concurrency can be controlled by the technique called **locking**.

LOCKING:

- A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it.
- Locking is a mechanism to ensure data integrity while allowing maximum concurrent access to data. It is used to implement concurrency control when multiple users access table to manipulate its data at the same time.

The simplest form of lock can be a binary lock having only 2 possible state only -
1. LOCK
2. UNLOCK

Depending upon the rules we have found, we can classify the locks into two types.
Shared Lock(S): A transaction may acquire shared lock on a data item in order to read its content. The lock is shared in the sense that any other transaction can acquire the shared lock on that same data item for reading purpose.

Exclusive Lock(X): A transaction may acquire exclusive lock on a data item in order to both read/write into it. The lock is excusive in the sense that no other transaction can acquire any kind of lock (either shared or exclusive) on that same data item.

The relationship between Shared and Exclusive Lock can be represented by the following table which is known as **LockMatrix.**

	Shared	Exclusive
Shared	TRUE	FALSE
Exclusive	FALSE	FALSE

Two Phase Locking Protocol:-

The Two Phase Locking Protocol defines the rules of how to acquire the locks on a data item and how to release the locks.
The Two Phase Locking Protocol assumes that a transaction can only be in one of two phases.

Phase 1: The lock acquisition phase/Growing phase:

In this phase the transaction can only acquire locks, but can not release any locks till all the locks acquired by the transaction are obtained.

The transaction enters into growing phase as soon as it acquires first lock.

Once all locks have been acquired the transaction is in its **locked point**.

Phase 2: Lock Release Phase/Shrinking Phase:

After a lock point has been reached the transaction enters into shrinking phase.

The existing locks can be released in any order but no new lock can be acquired after a lock has been released.

The transaction enters into shrinking phase as soon as it release the first lock.

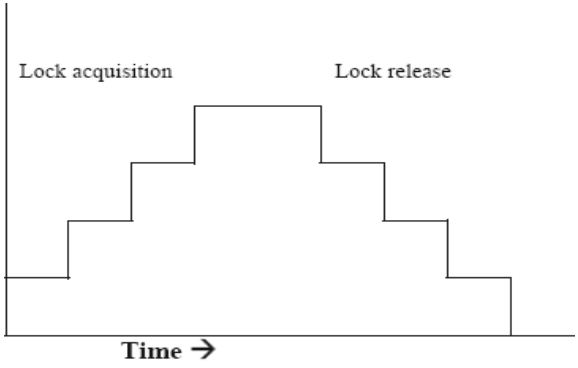
There are two types of 2PL:

(1) The Basic 2PL
(2) Strict 2PL

The Basic 2PL:

The basic 2PL allows release of lock at any time after all the locks have been acquired.

Disadvantage: Once a lock has been released on a data item it can be modified by another transaction before the first transaction comits or aborts.

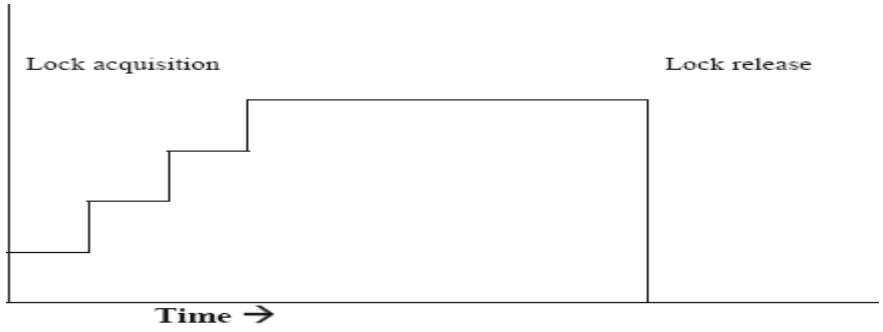


Basic Two Phase Locking

Strict 2PL:

To avoid such a situation we use strict 2PL. Transaction T does not release any of its exclusive lock (X) until that transaction comits or aborts.

In this way no other transaction can access the item that is written by T unless the transaction T comits.



Strict Two Phase Locking

But due to locking we can face two problems

1. DEAD LOCK
2. LIVE LOCK

DEAD LOCK:

A deadlock situation occurs when two transaction wait indefinitely for each other to unlock data.

Deadlock occurs when 2 transaction T1 &T2 exist in the following mode.

T1= Access data item X and Y

T2= Access data item Y and X

If T1 has not unlocked data item Y, T2 cannot begin. If T2 has not unlocked data item X ,T1 can not continue. Consequently T1 and T2 wait indefinitely, each waiting for the other to unlock the required data item. Such situation is called as deadlock.

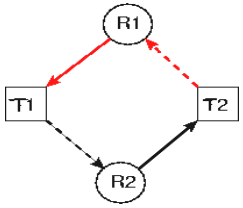
NECESSARY CONDITION

There are four necessary condition for dead lock to occur

1. Mutual Exclusion
2. Non Primitive Locking
3. Partial Allocation
4. Circular wait

1. **Mutual Exclusion:** A resource can be locked in exclusive mode by only one transaction at a time.
2. **NON -Primitive Locking:** A data item can only be unlocked by the transaction that lock it No other transaction can unlock it.
3. **PARTIAL ALLOCATION:** A transaction can acquire locks on data base in a piecemeal fashion.

4. **CIRCULAR WAITING:** Transaction lock part of the data resources needed and then wait indefinitely to lock the resources currently locked by other transaction.



APPROACHES FOR DEALING WITH DEAD LOCK:

1. Prevention
2. Avoidance
3. Detection and recovery

DEADLOCK PREVENTION

In order to prevent the dead lock one has to ensure that at least one of above condition does not occurs.

The better prevention algorithm have been evolved to prevent a deadlock having the basic logic: not to allow the circular wait to occur.

Their are 2 schemes to prevent deadlock

1. Wait die scheme

2. Wound wait scheme

Wait die scheme:

This schemes is based on the non primitive technique .

It is based on a simple rule.

If Ti request a data base resource that is held by Tj .

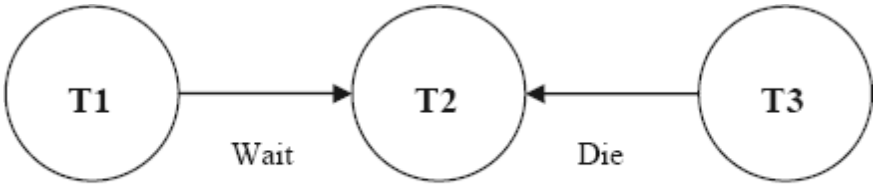
Then if Ti has a smaller time stamp than that of Tj, it is allowed to wait

else Ti aborts

A **timestamp** is defined as the system generated sequence no. that is unique for each transaction so a smaller time stamp means an older transaction.

For example:

Assume that 3 transaction T1,T2,T3, is generated in that sequence .if T1request for a data item which is currently held by transaction T2, it is allowed to wait as it has a smaller time stamp as that of T1, however T3 request for a data item which is current held by the T2 then T3 is rolled back(die or abort)

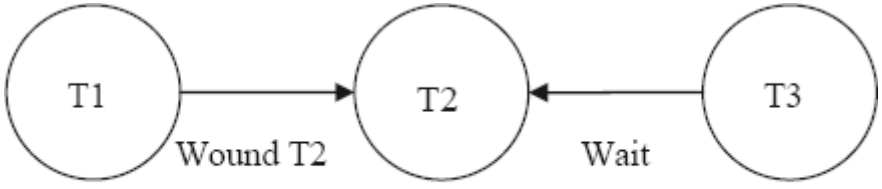


Wait-die Scheme of Deadlock prevention

2. WOUND WAIT

It is **b**ased **on** the primitive technique .
It is based on the simple rule:
If Ti request a database resources that is held by Tj
Then if Ti has a larger timestamp than that of Tj, It is allowed to wait
else Tj is wounded of by Ti
FOR EXAMPLE:

Assume that 3 transaction T1, T2, T3 are generated in that sequence. . If T1 request for a data item which is currently held by transaction T2 , then T2 is rolled back and data item is allowed to T1. however if T3 request for a data item which is currently held by T2 then T3 is allowed to wait.



Wound-wait Scheme of Deadlock prevention

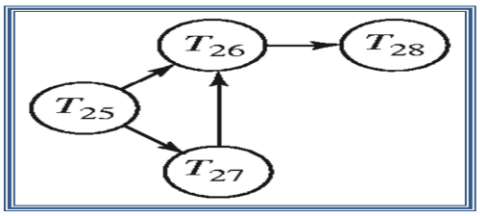
Here is the table representation of resource allocation for each algorithm

	Wait/Die	Wound/Wait
Older process needs a resource held by younger process	**Older process** waits	**Younger process** dies
Younger process needs a resource held by older process	**Younger process** dies	**Younger process** waits

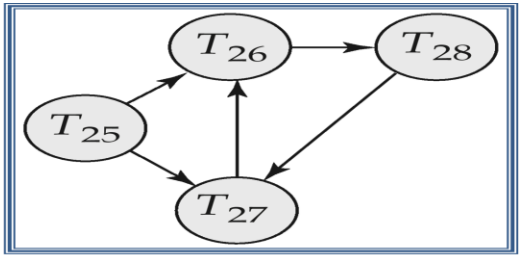
DEAD LOCK AVOIDANCE:
Transactions are not required to request resources a priori.
The transaction must obtain all the locks it need before it can be executed**.**
Transactions are not allowed to proceed unless a requested resources is available.
In case of conflict, transactions may be allowed to wait for a fixed time interval.
This technique avoids rollback of conflicting transaction by requiring that locks be obtained in a sequence.

DEAD LOCK DETECTION:

- Deadlocks can be described as a wait-for graph, which consists of a pair G = (V,E),
  - V is a set of vertices (all the transactions in the system)
  - E is a set of edges; each element is an ordered pair Ti →Tj.
- If Ti → Tj is in E, then there is a directed edge from Ti to Tj
  - implying that Ti is waiting for Tj to release a data item.
- When Ti requests a data item held by Tj, then Ti → Tj is inserted in the wait-for graph.
  - This edge is removed only when Tj is no longer holding a data item needed by Ti.
- The system is in a deadlock state if and only if the wait-for graph has a cycle.
- The system invokes a deadlock-detection algorithm periodically to look for cycles.



Wait-for graph without a cycle



Wait-for graph with a cycle

DEADLOCK RECOVERY:

1. Through preemption
2. Roll back

keep check point periodically
when a deadlock is detected , see which resource is needed.
Take away the resource from the process currently having it.
Later on , you can restart this process from a check pointed state where it may need to re acquire the resource. from deadlock.
If more then one process takes action , the deadlock detection algorithm can repeatedly trigger.

LIVELOCK

- A live lock is similar to a deadlock, except a condition that occurs when two or more process continuously change their state in response to change in other process.
- The result is that none of the process will complete its transaction.
- Live lock is a special case of resource starvation the general definition only state that a specific process is not progressing.
- A real world example of live lock is it occurs when two people meet in a narrow corridor and each tries to polite by moving a side to let the other pass but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time.
- If more than one process takes action, the deadlock detection algorithm can repeatedly trigger .
- This can be avoided by ensuring that only one process takes action.

Serializability

Serializability is a property of a transaction schedule (history). It relates to the *isolation* property of a database transaction.

- Basic Assumption – Each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
    - conflict serializability
    - view serializability
- We ignore operations other than **read** and **write** instructions, and we assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes. Our simplified schedules consist of only **read** and **write** instructions.

Chapter-8
SECURITY AND INTEGRITY

DATA ENCRYPTION

- Sometimes some users want to bypass the system by physically removing the part of the database or by tapping into a communication line.
- The Most effective counter measure against such threats is data encryption; it means storing and transmitting of data in an encrypted form.
- The original data is called plaintext the plain text is encrypted by subjecting it to an encryption algorithm, whose I/P's are the plain text and encryption key.
- The O/P's from this algorithm is the encrypted form of the plain text and is called as the cipher text.
- The details of the encryption algorithm can be made public but encryption key is kept secret.
- The cipher text is stored in the database and transmitted down the communication line.
- The data encryption can be made by two means :-

1. Substitution:-

In This type of encryption the plain text is substituted by another key, it gives the cipher text.

2. Rearrange

In this type of encryption, the plain text characters are simply rearranged into some different sequence and then it combines with the encryption key and gives the cipher text.
Neither of the approaches are particularly secure in itself but the algorithm that combines are to provide.

AUTHORIZATION:-

Authorization is the administrative policies of organisation ,expressed as a set of values that be used to determine which user has what type of access on which portion of the database .

The person who is in charge of specifying the authorization is usually called the authorizer. The authorizer is not the DBA but the person who owns the data.

The authorization usually maintained in the table called as access matrix . The access matrix contains rows called subject is columns called as objects .

The entity in the matrix at the portion corresponds to the intersection of a row & column indicates type of access that the subject has with respect to the object.

Objects:-

An object in the database environment could be a unit of data that needs to be protected. Thus a object can be data that needs to be protected. Thus a object can be data field, a record , a file or view.

Subjects:-

A subject can be an active element who operates on objects to a class of users or an application program.
If an user belong to more than one class of users, then access rights for a given access made by user that is being use by that user of access.

Access Type:-

The access allowed to the user or program could be for data manipulation or control. The manipulation operations are read, insert, update. The control operations are add, drop, alter.

VIEWS:-

- The relations that really exist in the database are referred to as base relation.
- Sometimes for security purpose it is undesirable to have all the users to see the entire relation. Any relation that is not part of the physical database i.e. virtual relation is called as views/subschemas.
- The view can be created from the base relation according to the requirement the views can be created by 'SQL' Query.

Types of View

There are two types of view,

- Simple View
- Complex View

Simple View	Complex View
Created from one table	Created from one or more table
Does not contain functions	Contain functions

Does not contain groups of data	Contains groups of data

SYNTAX:-

CREATE VIEW viewname AS (query expression);
OR
CREATE VIEW viewname AS (select expression);

Example: create view view1 as select empno, empname from emp;
To see the view: select *from viewname;

- A view is a relation and can be used in query expressions. Views are generally not stored the data in the base relation may change .
- The create view state is stored in the system catalogue, when a query refers to any view the virtual relation defined by the view.

SECURITY AND INTEGRITY :-

- Security & Integrity are frequently heard together in database, but the two concepts are actual quite distinct.
- Security refers to the protection of data unauthorised disclosure, alteration or destruction security involves ensuring that users all to do things while others are trying to do.
- Integrity refers to the accuracy or validity of data.
- Integrity means ensuring that the things are trying to do are correct.
- Modern DBMS support either or both of 2 approaches to data security . They are as follows:-

    1) Discretionary Model
    2) Mandatory Control

- Most of Systems supports the discretionary control and some systems supports the mandatory control.

Discretionary Control:-

- Here are given user will have different access regular on privileges on different object, different users will have different rights on the same object.
- This Scheme is very flexible.

SYNTAX:-

CREATE SECURITY RULE rule name GRANT

Privilege – list ON expression To user – list [ON ATTEMPTED VIOLATION ACTION];

EXAMPLE:-

CREATE SECURITY RULE SR3 GRANT RETRIVE, DELETE ON S WHERE CITY =’BLS’ TO X,Y,Z.
[ON ATTEMPTED VIOLATION REJECT];

The rule will register under system catalogue under the name SR3.

The privilege RETRIVE & DELETE are granted to the users by means of to clause where the city of table is Balasore.

Mandatory Control:-

- Mandatory control are applicable to database in which the data is rather static and rigid classification structure.
- This control is applied to the military or government environment.
- Here the data object has a classification level and user has a clearance level.
- User ’I’ can see to object ‘J’ if the clearance level of ‘I’ is greater than or equal to clearance level of ‘J’.
- User ‘I’ can modify object ‘J’ of the clearance level of ‘I’ is equal to the classification of ‘J’.

INTEGRITY CONSTRAINT:-

Integrity constraint gives the protection to the data by following means:-

- It enforces rules on the data in a table where a row is inserted, updated or deleted from that table.
- Prevent deletion of a table on the content of its, if there are dependencies from other tables.

The constraint types are :-

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

- The constraint can be imposed at the time of table creation or after the table has been created.
- The constraint can be defined at the column or table level.
- All the constraint are stored in the data dictionary.
- The Syntax when constraint is defined at the column level is a s follows:-

CREATE TABLE table name
Column data type
Column data type [CONSTRAINT CONSTRAINT-NO CONSTRAINT-TYPE];

- The Syntax when constraint is defined at the table level is a s follows:-

CREATE TABLE table name
Column datatype

[CONSTRAINT CONSTRAINT-NAME] CONSTRAINT-TYPE(COL);

NOT NULL:-

The NOT NULL constraint ensures the column contains no NULL values. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

SYNTAX:-

CREATE TABLE table name (Column name) data type

EXAMPLE:-

CREATE TABLE std (roll number (10) not null, name branch varchar2 (20, dob date);

UNIQUE:-

A unique key constraint requires that value in a column be unique i.e. no rows of table can have duplicate values.

SYNTAX:-

CREATE TABLE table name (Column name 1 data type, Column name 2 data type unique, Column name 3 data type);

EXAMPLE:-

CREATE TABLE std (roll number (10),name varchar2(10) unique, dob date not null, mark number (8));

PRIMARY KEY:-

A Primary key constraint creates a primary key for the table by which each row can be identify uniquely.

- Only one primary key can be created for each table.
- The primary key is the combination of unique & not null.

SYNTAX:-

CREATE TABLE table name (Column name 1 data type, Column name 2 data type primary key, Column name 3 data type);

EXAMPLE:-

CREATE TABLE std (roll number (10), CONSTRAINT PK-roll primary key, name varchar2 (10) unique, branch varchar2 (20), dob date not null);

FOREIGN KEY:-

- The foreign key or referential integrity constraint designates a column as a foreign key and establishes a relationship between primary key in the same table or a different table.
- The table where the foreign key is present is called the depend or child table where the primary key which is referred is present is called the referenced or parent table.

SYNTAX:-

CREATE TABLE table name 1 (Column name 1 data type primary key, Column name 2 data type reference table name 2, (Column));

EXAMPLE:-

CREATE TABLE std (roll number (10), CONSTRAINT PK-roll primary key, name varchar2 (10) unique, branch varchar2 (10), CONSTRAINT fR-branch);

CHECK:-

Check defines the condition that each row must satisfy. This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

SYNTAX:-

CREATE TABLE table name 1 (Column name 1 data type check (condition), Column name 2 data type);

EXAMPLE:-

CREATE TABLE sal (salary number (10) check (salary, name varchar2(10));

TRANSACTION FILE & MASTER FILE:-

- Master file is the main file, there is no other master file.
- Transaction file is derived file from the master file are a no. of transaction files.
- In Master file we can't do any updation or manipulation.
- All changes are done on transaction file.
- All the updation are done on the transaction file and are stored in the master file at a time on a particular day.
- Master file is the root and the transaction files are nodes.

SECURITY CONSTRAINTS:-

Security in a database involves both policies and mechanisms to protect the data and ensure that it is not accessed, altered or deleted without proper authorization.

There are four levels of defense or security constraints are generally recognized for database security: human factors, physical security, administrative control, and DBMS and Operating System Mechanisms .

1. **Human Factors:-.**
   - An organization usually performs some type of clearance procedure for personnel who are going to be dealing with sensitive information, including that contained in a database.
   - This clearance procedure can be a very informal one, in the form of the reliability and trust that an employee has earned in the eyes of management.
   - The authorizer is responsible for granting proper database access authorization to the user community.
2. **Physical Security:-**
   - Physical security mechanisms include appropriate locks and keys to computing facility and terminals.
   - Security and physical storage devices (magnetic disk packs etc.) within the organization and when being transmitted from one location to another must be maintained..
   - Authorized terminals from which database access is allowed to have to be physically secure, otherwise unauthorized persons may be able to access the data.
   - User identification and passwords have to be kept confidential.
3. **Administrative Controls**:- Administrative controls are the security and access control policies that determine what information will be accessible to what class of users and the type of access that will be allowed to this class.
4. **DBMS and Operating System Mechanisms**:
   - The proper mechanisms for the identification and verification of users.
   - Each user is assigned an account number and a password. The operating system ensures that access to the system is denied unless the number and password are valid.
   - In addition to the DBMS could also require a number and password before allowing the user to perform any database operations.

References

1. Book Reference: A. Silberschatz, H.F. Korth Database System Concepts
2. Book Reference: Database Management System (DBMS)A Practical Approach Book by Rajiv Chopra
3. Book Reference: Database Management System by P.K Yadav
4. Book Reference: An Introduction to Database Systems Bipin C.Desai
5. https://nptel.ac.in
6. https://www.javatpoint.com
7. https://www.geeksforgeeks.org