CS 2255– DATABASE MANAGEMENT SYSTEMS
(Common to CSE and IT)

L T P C
3 0 0 3

UNIT I FUNDAMENTALS 9

Purpose of database system – Views of data – Data models – Database languages– Database system architecture – Database users and administrator – Entity Relationship model (E-R Model) – E-R diagrams – Introduction to relational databases.

UNIT II RELATIONAL MODEL 9

The relational model – The catalog – Types – Keys – Relational algebra – Domain relational calculus – Tuple relational calculus – Fundamental operations – Additional operations – SQL fundamentals – Integrity – Triggers – Security – Advanced SQL features – Embedded SQL – Dynamic SQL – Missing information – Views – Introduction to distributed databases and client/server databases.

UNIT III DATABASE DESIGN 9

Functional dependencies – Non-loss decomposition – Functional dependencies – First – Second – Third normal forms – Dependency preservation – Boyce/codd normal form – Multi-valued dependencies and fourth normal form – Join dependencies and fifth normal form.

UNIT IV TRANSACTIONS 9

Transaction concepts – Transaction recovery – ACID properties – System recovery – Media recovery – Two phase commit – Save points – SQL facilities for recovery – Concurrency – Need for concurrency – Locking protocols – Two phase locking – Intent locking – Deadlock – Serializability – Recovery Isolation Levels – SQL Facilities for Concurrency.

UNIT V IMPLEMENTATION TECHNIQUES 9

Overview of Physical Storage Media – Magnetic Disks – RAID – Tertiary Storage – File Organization – Organization of Records in Files – Indexing and Hashing – Ordered Indices – B+ Tree Index Files – B Tree index files – Static hashing – Dynamic hashing – Query processing overview – Catalog information for cost estimation – Selection operation – Sorting – Join operation – Database Tuning.

Total: 45

TEXT BOOKS

1. Silberschatz, A., Korth, H.F. and Sudharshan, S., “Database System Concepts”, 5th Edition, Tata Mc-Graw Hill, 2006
2. Date, C. J., Kannan, A. and Swamynathan, S., “An Introduction to Database Systems”, 8th Edition, Pearson Education, 2006.

REFERENCES

1. Elmasri, R. and Navathe, S.B., “Fundamentals of Database Systems”, 4th Edition, Pearson / Addison Wesley, 2007.
2. Ramakrishnan, R., “Database Management Systems”, 3rd Edition, Mc-Graw Hill, 2003.
3. Singh, S. K., “Database Systems Concepts, Design and Applications”, 1st Edition, Pearson Education, 2006.

UNIT I - INTRODUCTION

Data

- Data are raw facts. (E.g.: Name, Telephone no etc.)

Information

- Processed raw data.
- Three key attributes of information
  - i) Accuracy
  - ii) Timeliness
  - iii) Relevancy

Information processing

It is
- i) Acquisition
- ii) Storage
- iii) Organization
- iv) Retrieval
- v) Display and
- vi) Dissemination

Data Base

Database is a collection of interrelated data.

Database Management Systems

DBMS is a collection of interrelated data and a set of programs to access those data. Applications: banking, airlines, universities, finance etc.

Why study Data Base?

- i) Shift from computation to information.
- ii) Data sets increasing in diversity and volume.
- iii) DBMS encompasses most of computer science

Why Use a Data Base?

A data base system provides the organization with centralized control of its data.

Data base involves,

- i) Definition of structures for information storage [Data Modeling]
- ii) Providing of mechanism for the manipulation of information.

iii) Concurrency control if the system is shared by users.

iv) Security and crash recovery.

Features of a database:

i) It is a persistent (Stored) collection of related data.

ii) The data is input (Stored) only once.

iii) The data is organized (In some fashion).

iv) The data is accessible and can be queried (Effectively and Efficiently)

Database Applications:

1. Banking: all transactions
2. Airlines: reservations, schedules
3. Universities: registration, grades
4. Sales: customers, products, purchases
5. Online retailers: order tracking, customized recommendations
6. Manufacturing: production, inventory, orders, supply chain
7. Human resources: employee records, salaries, tax deductions

Databases touch all aspects of our lives.

Data base technology is CORE TECHNOLOGY with links to:

- Information management / Processing
- Data analysis / Statistics
- Multimedia and hypermedia
- Office and document systems
- Business processes ,Work flow , CSCW
  (Computer Support Cooperative work)

But modern DB System depends on an infrastructure of:

- Networks (Both LAN and WAN)
- Client – Server computing architecture

DATA BASE MANAGEMENT SYSTEM

History of Database Systems 1950s and early 1960s:

- Data processing using magnetic tapes for storage
- Tapes provide only sequential access
- Punched cards for input

Late 1960s and 1970s:

- Hard disks allow direct access to data
- Network and hierarchical data models in widespread use
- High - performance (for the era) transaction processing
- Ted Codd defines the relational data model
  - Would win the ACM Turing Award for this work
  - IBM Research begins System R prototype
  - UC Berkeley begins Ingres prototype

1980s:

- Research relational prototypes evolve into commercial systems
  - SQL becomes industry standard
- Parallel and distributed database systems
- Object -oriented database systems

1990s:

- Large decision support and data-mining applications
- Large multi-terabyte data warehouses
- Emergence of web commerce

2000s:

- XML and XQuery standards
- Automated database administration
- Increasing use of highly parallel database systems
- Web-scale distributed data storage systems

Purpose of Database systems

Before dbms, data were stored in OS files. Permanent records are stored in various files and some application programs to extract and add records to those files.

The following are the disadvantages of file processing system.

Disadvantages of file processing systems

- Data redundancy and inconsistency
- Difficulty in accessing data
- Data isolation
- Integrity problems

- Atomicity problems
- Security problems

(*) Data redundancy and inconsistency:

Redundancy →the same information may be duplicated in several files. Inconsistency → data are modified in one file and not in another file.

(*) Difficulty in accessing data:

If there is no application program for specific task, accessing data is not possible.

(*)Data isolation:

Data are scattered in various files and each file may be in different format. Hence different application programs are needed.

(*)Integrity problems:

The data values stored in database must satisfy some constraints (ie) in a bank account the amount should not be less than 1000. According to this condition the application program should be developed.

(*) Atomicity problems:

If in case of system failure the system cannot restore to the consistent state that was before failure. This problem is known as atomicity problem.

Ex: $1000

A----------------/----→B

Failure

(*)Security problems:

Prevention of data access by unauthorized users.

Comparison of File Processing system and DBMS

DBMS	File Processing System
1. It allows access to tables at a time	It allows access to single file at a time
2. It co-ordinates the physical and logical access to the data	It co-ordinates only the physical access to the data.
3. It reduces the amount of data duplication	It often have redundant or duplicate data items
4. It is designed to allow flexibility in What queries give access to the data	It allows only pre-determined access to data(By specific compiled programs)

5. It is designed to co-ordinate and permit multiple users to access data at the same time	It is much more restrictive in simultaneous data access
6. It has a unique key or index in order to access data directly or randomly.	It do not have keys or indices in order to find data rapidly
7. It is collection of related tables	It is a collection of related records

Views of Data

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

Benefits of views

i) Views provide a level of security
ii) Views provides a mechanism to customize the appearance of the database.
iii) A view can present a consistent, unchanging picture of the structure of the database, even if the underlying database is changed.

Data Abstraction

System hides certain details of the data like "how the data are stored and maintained ". Since many database system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify user's interaction with the system. There are 3 levels of data abstraction.

(i) physical level
(ii) logical level
(iii) view level

View level →describe only the part of db
Logical level →what data are stored in db and what is the relationship between them
Physical level →how data structures are defined in the database and the way of storing data in db.

Objective of the three level Architecture:

To separate each user's view of the database from the way the database is physically represented.

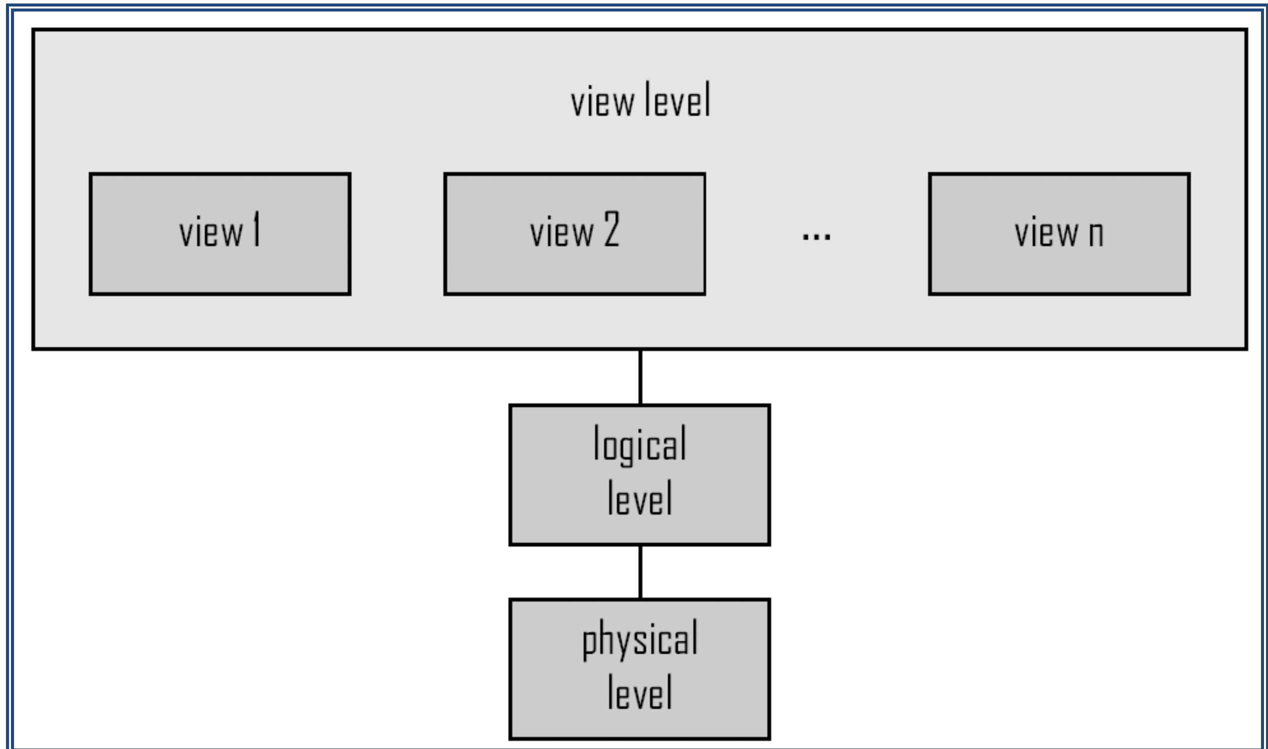


Fig : Architecture for a database system or Three levels of data abstraction

Instance and schema

The collection of information stored in a db at a particular moment is known as instance. An instance changes frequently. The overall design of db is known as schema. Schema changes occasionally. Database systems have several schemas partitioned according to the levels of abstraction.

The Physical schema describes the database design at the physical level, while the logical schema describes the database at the logical level. A database may also have several schemas at the view level, sometimes called sub schemas that describe different views of the database.

Physical level / Internal level : describes how a record (e.g., customer) is stored.

Logical level / External level : describes data stored in database, and the relationships among the data.

type *customer* = record
customer_id : string;
customer_name : string;
customer_street : string;
customer_city : string;
end;

View level / Conceptual level : application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

Instance and schema

Instance:

The collection of information/data stored in a dB at a particular moment is known as instance. An instance changes frequently.

Schema / Data base Schema:

The overall design of dB is known as schema.

Types of Schema:

Schema changes occasionally. Database systems have several schemas partitioned according to the levels of abstraction.

- External Schemas / logical Schema
- Conceptual Schema
- Internal / Physical Schema

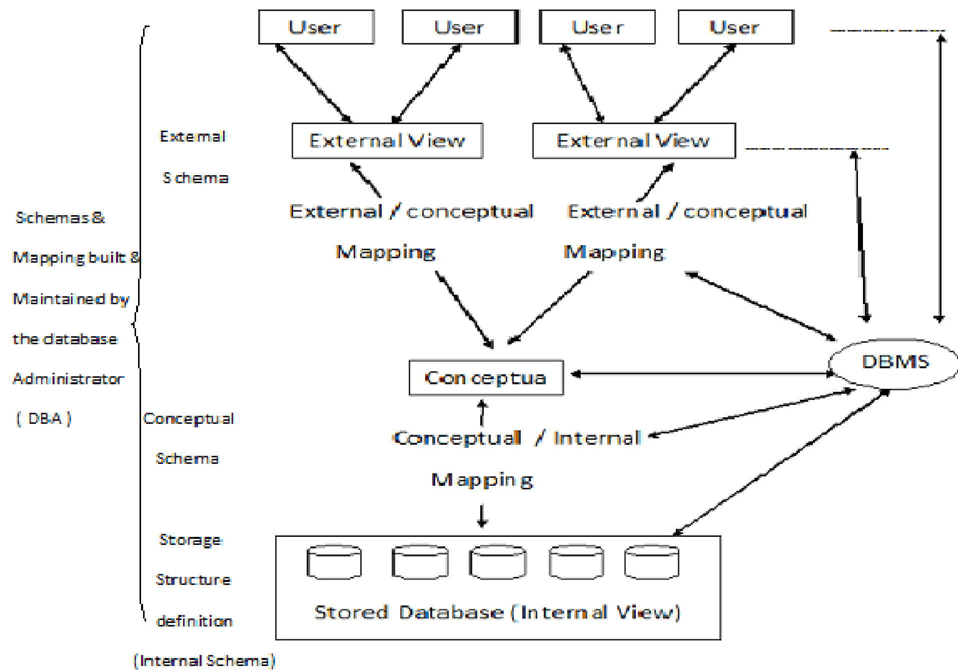


Fig : Three – Schema Architecture

Internal / Physical Schema:

It describes the database design at the physical level , which is the lowest level of abstraction describing how the data are actually stored.

External Schemas / logical Schema:

It describes the database design at the logical level, which describes what data are stored in the database and what relationship exits among the data.

Conceptual Schema:

The schemas at the view level are called sub schemas that describes different views of the database.

Data Models

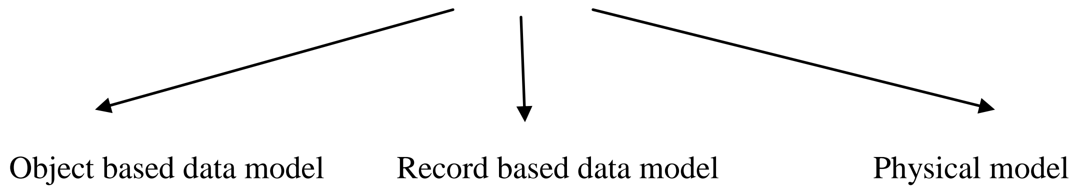
Data model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical and view level.

Data models can be classified into 4 categories:

- Relational model
- The entity relationship model

- Object based data model
- Semi structured data model

DATA MODEL



Object based data model	Record based data model
- E-R model	- Relational model
- Semantic model	- Network model
- Object oriented model	- Hierarchical model
- Functional model	

Relational Model

The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns otherwise known as attributes and each column has unique name. Since the database is structures in fixed format records of several types it is also known as record based model. Each record type defines a fixed number of fields or attributes. Each row in the table is called “tuple”.

The Entity Relationship Model

The E-R is based on a perception of a real world that consists of a collection of basic objects called entities and of relationships among these objects. An “entity” is a thing or object in the real world that is distinguishable from other objects.

The entity relationship (E-R) model consists of a collection of basic objects, called entities and of relationships among these entities.

Entity

An “entity” is a thing or object in the real world that is distinguishable from other objects.

Example:

Each person is an entity , and bank accounts is an entity.

Attributes

Entities are described in a database by a set of attributes.

Example :

The attributes of account entity are account – no, balance, etc.

Types of Attributes

i) Composite and simple (Atomic) attributes

ii) Single valued and multi valued attributes

iii) Stored and derived

iv) Null values

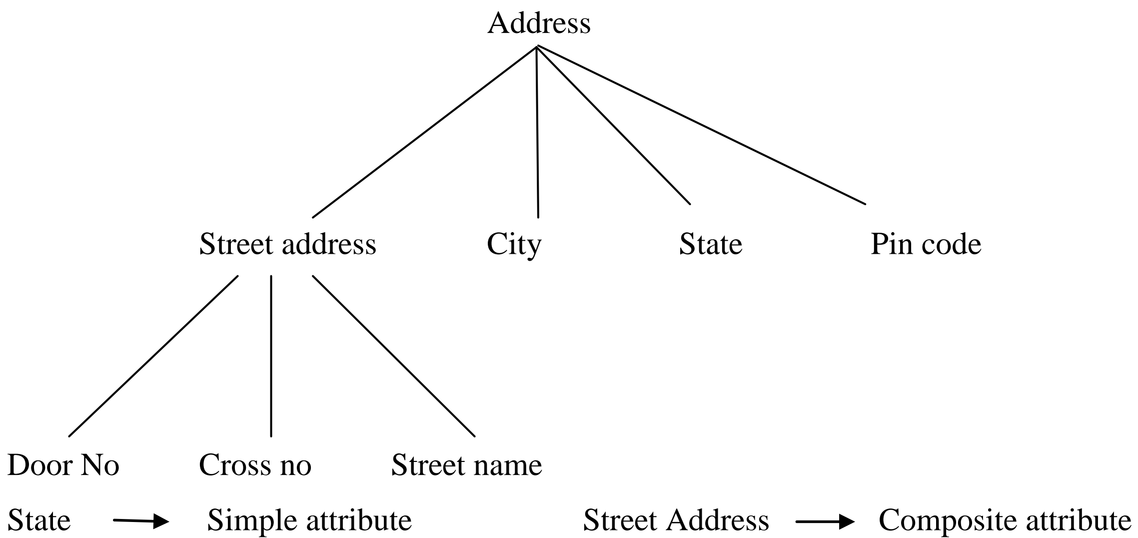
v) Complex attributes

Composite and simple (Atomic) attributes

Composite attribute,

Can be divided into smaller subparts

Can from hierarchy



State → Simple attribute Street Address → Composite attribute

Single valued and multi valued attributes

Single valued attributes have a single value for a particular entity.

(E.g. Age attributes of a person)

Multi valued attributes has multiple values for a particular entity.

(E.g. Color attributes of a car)

Stored and derived attributes

In some cases two or more attribute values are related. (E.g. Age and birth date attributes of a person)

For a particular person entity, the value of age can be determined from the current date and the value of that person's birth date.

Age attribute is called as derived attribute.

Birth attribute is called as stored attribute (or) Base attribute.

Null Values

An attribute takes a null value when an entity does not have a value for it

Null can also be designated that an attribute value in unknown. An unknown value may be either missing or not known.

Object Based Data Model

The object oriented data model can be seen as extending the E-R model with other notions of encapsulation, methods (functions), and object identity. The object relational data model combines features of the object –oriented data model and relational data model.

Semistructured Data Model

The Semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The Extendible Markup Language (XML) is widely used to represent semi structured data.

Hierarchical model

There are two types of data structure exits in this model. They are

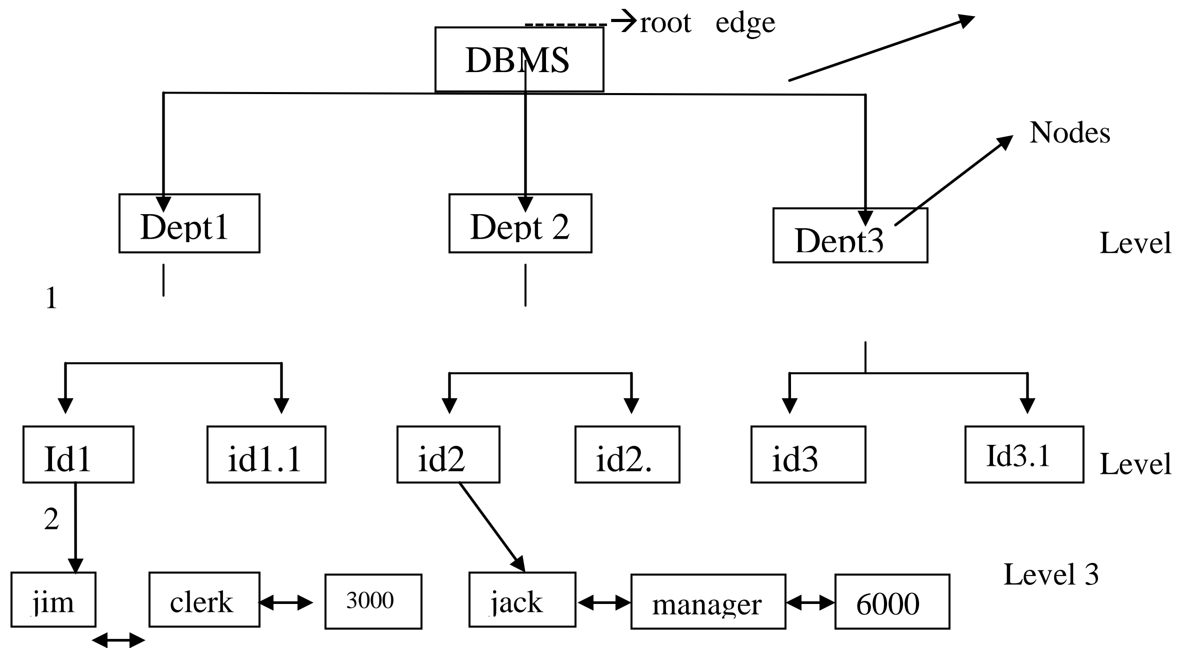
(i) Records

(ii) PCR - Parent Child Relationship.

Record → it is nothing but collection of fields. Each field has data type

Records of same type are grouped into record types.

PCR type→ one to many relationships between two record types.

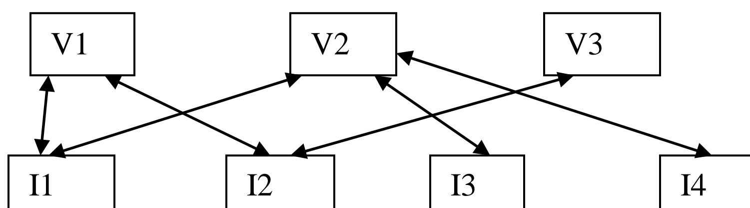


Advantage:	**Disadvantage:**
(i) High speed	complex to implement
(ii) Ease of updates	management problem
(iii) Simplicity	programming complexity
(iv) Data security	Implementation limit
(v) Data integrity	Redundancy of data.
(vi) Efficiency	

Network Model

In this model the data is represented as collection of records and relationships among data is represented by “links” (pointers).

To locate particular record it uses pointers.



Advantages:

- Simplicity
- Can handle 1: n and n: n
- Ease data access
- Data integrity and data dependency

Disadvantages:

- Detailed structure knowledge is required
- Lack of structural independency

Database Languages:

A database system provides a data definition language to specify the database schema and a data manipulation language to express database queries and updates. The DDL and DML are not two separate languages. Instead they simply form parts of a single database languages, such as SQL language.

Data Manipulation Language:

DML is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are

1. Retrieval of information stored in database
2. Insertion of new information into the database
3. Deletion of information from the database
4. Modification of information stored in the database,

These are basically two types:

✓ Procedural DMLs: requires a user to specify what data are needed and how to get those data.

✓ Declarative DMLs / non-procedural DMLs : requires a user to specify what data are needed without specifying how to get those data.

A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language.

Data Definition Language:

DDL is used to specify additional properties of the data. The storage structure and access methods used by the database systems are specified by a set of statements in a special type of DDL called data storage and definition language. The data values stored

in the database must satisfy certain consistency constraints. Some of the integrity constraints that the databases concentrate on are

1. Domain constraints
2. Referential integrity
3. Assertions
4. Authorization

Domain Constraints:

A domain of possible values must be associated with every attribute. Declaring an attribute to be a part of a particular domain acts as a constraint on the values that it can take. It is the most elementary part of integrity constraints that can be easily tested by the system whenever a new data item is entered into the database.

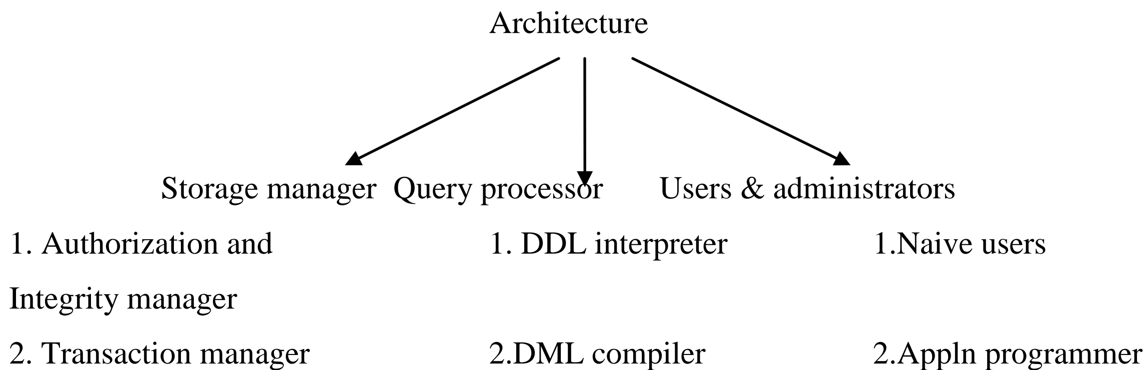
Referential integrity:

There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears for a set of attributes in another relation. Database modifications can cause violations of referential integrity. When referential integrity constraint is violated the normal procedure is to reject the action that caused the violation.

Assertions:

An assertion is any condition that the database must always satisfy. Domain constraints and referential integrity are special forms of assertions. When an assertion is created, the system tests it for validity. If the assertion is valid then any future modifications to the database is allowed only if it does not cause the assertion to be violated.

Database System Architecture:



3. File manager	3.Query evaluation	3.Sophisticated users
4. Buffer manager	Engine	4.Specialized users

Storage manager:

(i) Authorization and integrity manger:

Checks for integrity constraints and for authorized users.

(ii) Transaction manager:

Ensures that database remains in a consistent state when system fails and proceeds without conflicting.

(iii) File manager:

Manages the allocation of storage space on disk and data structures used to store that information.

(iv) Buffer manager:

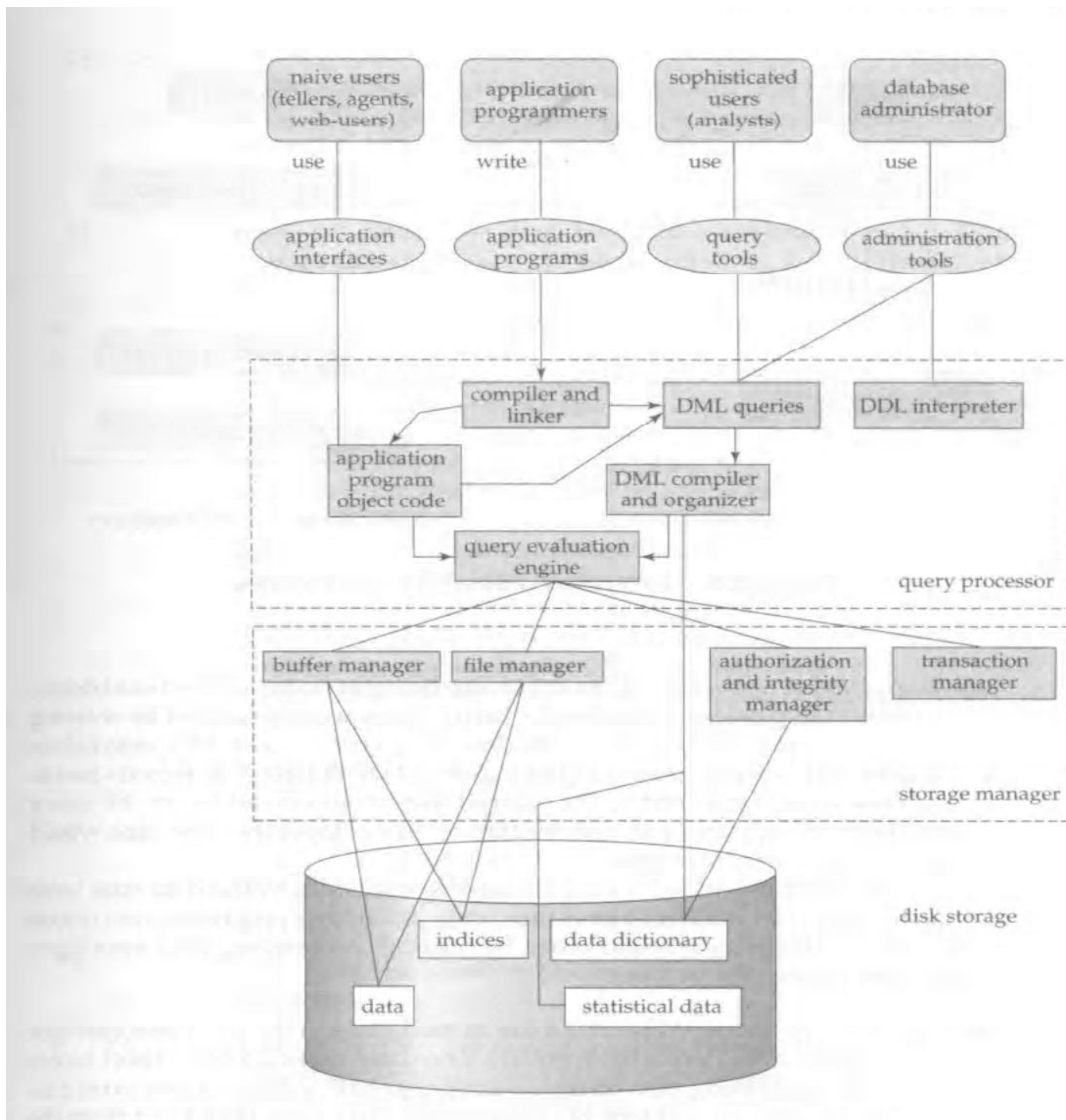
It is responsible for fetching data from disk into main memory.

Data structure used by storage manager

- Data files → where database is stored
- Data dictionary → contains all files in db and the no of records
- Indices → like index it provides fast access to data items

Query processor

- DDL Interpreter →the low level language DDL (stmts)is interpreted
- DML compiler →the low level language DML(stmts) are compiled
- Query evaluation → executes the low level instructions generated by DML engine compiler



Database users and administrators:

- Naïve users: → Naïve users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

  Eg: click the transfer button

- Application programmers:

→ Application programmers are computer professional who write application programs. They can choose from many tools to develop user interfaces.

→ Rapid Application Development Tools (RAD tools) are a tool that enables the application programmer to construct forms and reports with minimal programming effort.

- Sophisticated users → these users interact with the system without writing programs. They form their request in a database query language. They submit the query to the query processor, whose function is to break down DML statements into instructions that the storage manager understands.
- Specialized users: → are sophisticated users who write specialized database applications that do not fit into the traditional data processing framework. Among these applications are computer aided design systems, knowledge based and expert systems, systems that store data with complex data types and environmental modeling systems.

Data Administrator:

One of the main reasons for using DBMS is to have central control of both the data and the programs that access those data. A person who has the central control over the system is called a Database Administrator (DBA). The functions of DBA include:

1. Schema Definition:

The DBA creates the original database schema by executing a set of data definition in the DDL.

2. Storage structure and access method definition
3. Schema and physical organization modification:

The DBA carries out changes to schema and physical organization to reflect the changing needs of the organization or to alter the physical organization to improve performance.

4. Granting of authorization for data access:

By granting different types of authorization, the DBA can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

5. Routine Maintenance:

- Periodically backing up the database, either onto tapes or on to remote servers to prevent loss of data.

- Ensuring that enough free disk space is available of normal operations and upgrading disk space is required.
- Monitoring the jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by different users.

E-R MODEL:

Entity relationship model:

Entity is nothing but set of objects in the real world. There are 3 components in the E-R model.

They are (i) entity (ii) attributes (iii) relationship set.

(a) Entity set:

An entity is a "thing" or "object" in the real world that is distinguishable from other objects. Eg. Person, car, house etc. Each entity has its own properties. An entity set is a collection of entities having same properties.

(b) Attributes:

The properties that describe an entity is called an attribute. There are different types of attributes.

(i) Simple attributes →an attribute that cannot be divided into further Subparts. Eg. Cust_id of customer entity.

(ii) Composite attribute →an attribute that can be divided into set of subparts. Eg. Cust_name→first name, last name, middle name

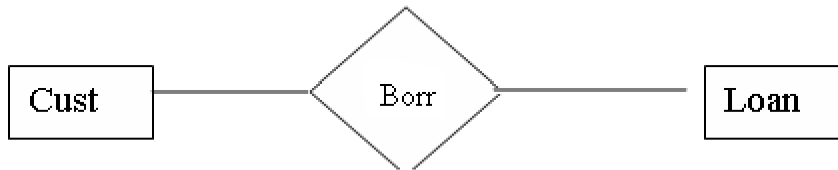
(iii) Single value attribute →an attribute having only one value in a particular Entity. Eg. In a customer entity, name, id, st are Single value attribute

(iv) Multi-valued attribute →an attribute having more than one value for a particular entity. Eg. Phone number.

(v) Derived attribute →an attribute which is derived from other related attributes (or) entities. Eg. Age attribute is derived from D.O.B attribute.

(c) Relationship set:

A relationship is an association among several entities. A relationship set is a set of relationships of same type.



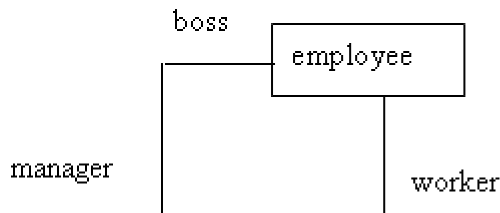
Relationship set

Recursive relationship set → an entity that participates more than once in different roles is known as recursive relationship set.

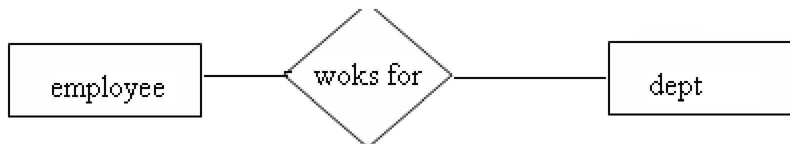
Degree of relationship set →the no of entity set participate in the entity set is the degree of the relationship set.

The following are the types of relationship sets:

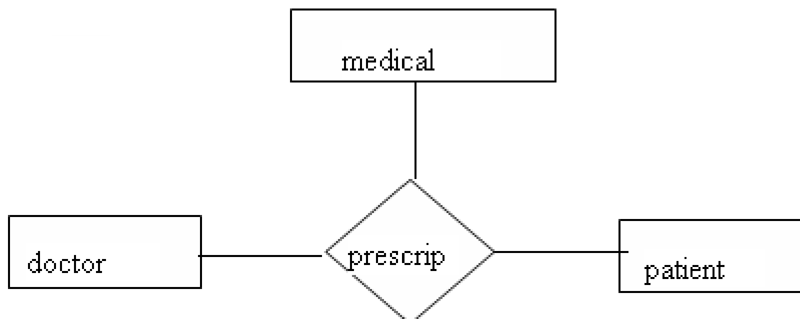
(i) Unary relationship set:



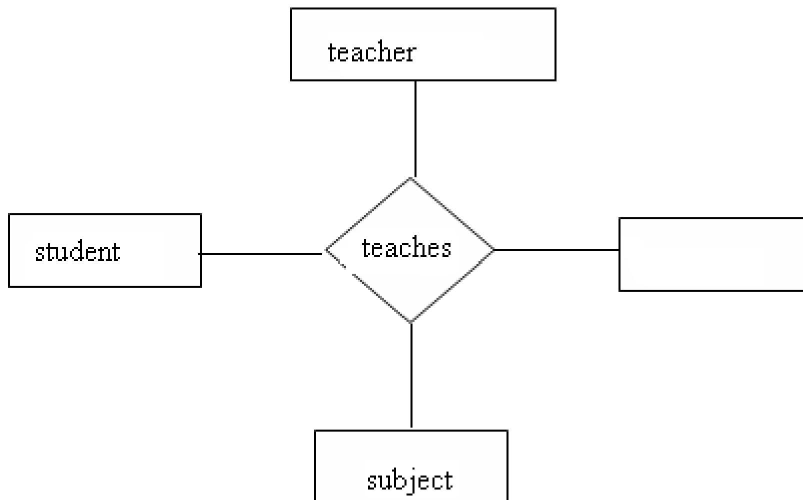
(ii)Binary relationship sets:



(iii)Ternary relationship sets:



(iv) Quaternary relationship set:



Mapping cardinalities:

The no of entities to which another entity can be associated via a relationship set is referred to as mapping cardinalities.

(i) one- to –one: (1 : 1)

one entity in A is associated with atmost one entity in B and vice versa.

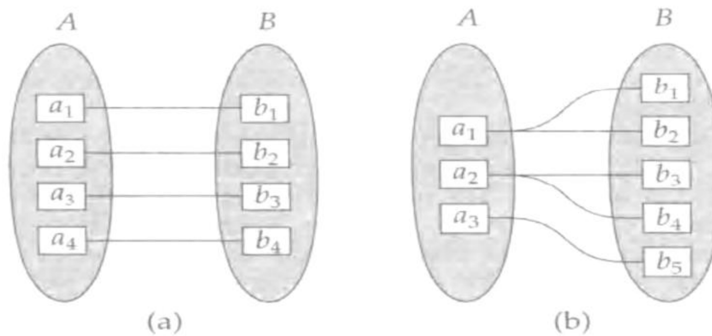
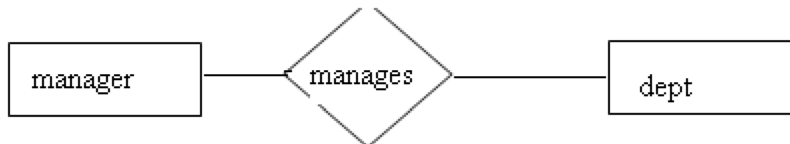
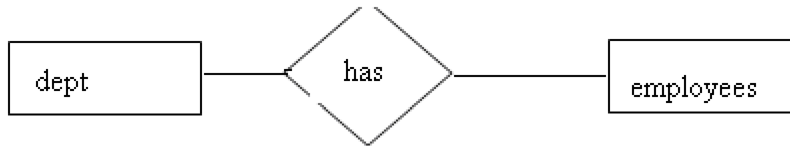


Figure 2.4 Mapping cardinalities. (a) One to one. (b) One to many.



(ii) one - to – many: (1 : M)

one entity in A is associated with more than one entity in B .



(iii) Many - to – many: (M:N)

Any no of attributes in A can be associated with any no of attributes in B.

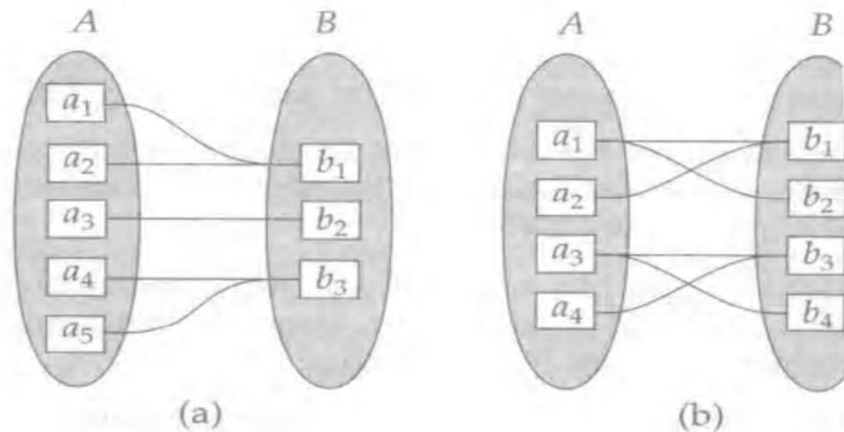
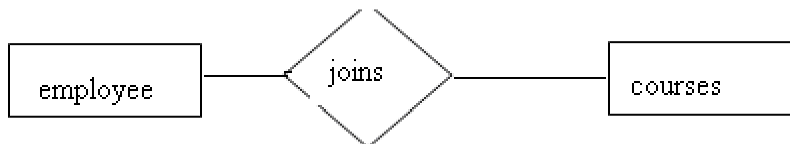
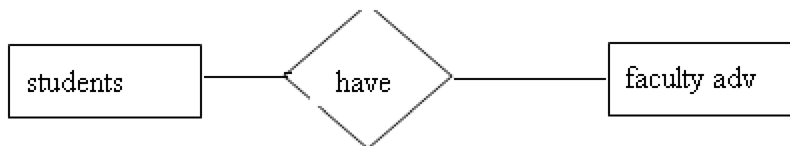


Figure 2.5 Mapping cardinalities. (a) Many to one. (b) Ma



(iv) Many - to –one: (M:1)

Any no of attributes in A can be associated with only one attribute in B.



E-R Diagram:

The logical representation of the overall logical structure of the database is called E-R diagram.

- Rectangle represents entity sets.
- Ellipse represents attributes.

- Diamond represents relationship sets.
- Lines links attributes to relationship sets and vice versa.
- Double ellipse represents multivalued attributes.
- Dashed ellipse which denote derived attribute.
- Double lines indicate total participation of an entity in a relationship set.
- Double rectangles represent weak entity sets.

(*) A weak entity set is an entity set that may not have sufficient attributes to form a primary key

(*) A strong entity set is an entity set that has a primary key.

Extended E-R model:

The E-R model that is supported with the additional semantic concept is called the extended Entity Relationship model (EER model).

- Specialization
- Generalization
- Aggregation

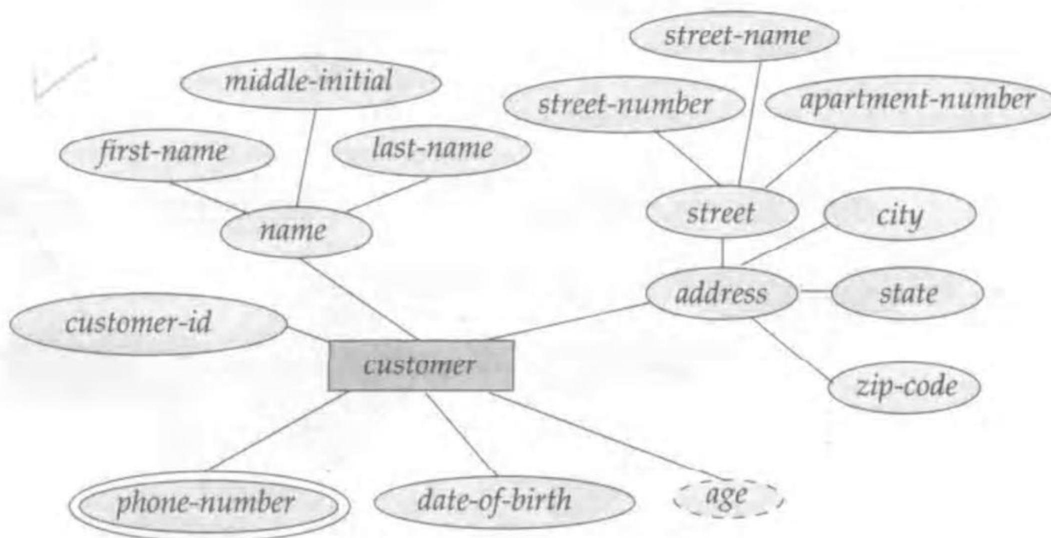


Figure 2.11 E-R diagram with composite, multivalued, and derived attributes.

Specialization:

The process of designating sub groupings within an entity set is called specialization. The specialization of person allows distinguishing among persons according to whether they are employees or customers. It is depicted by a triangle component labeled "ISA"

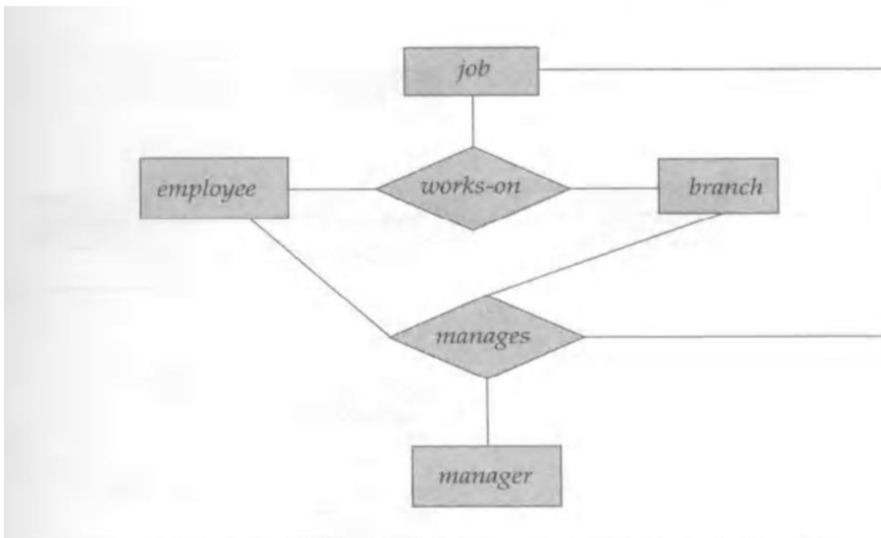
Generalization:

It is a process of defining a more general entity type from a set of more specialized entity types. For eg in the above diagram there are similarities between the customer entity set and employee entity set (i.e) they have more attributes in common. This commonality can be expressed by generalization. Here person is a higher entity set and customer and employee are lower entity sets, also called as super class and subclass.

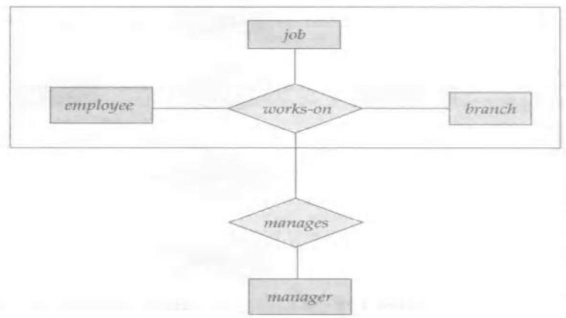
Aggregation:

Aggregation is an abstraction thro' which relationships are treated as higher level entities. It avoids redundancy and redundant relationships

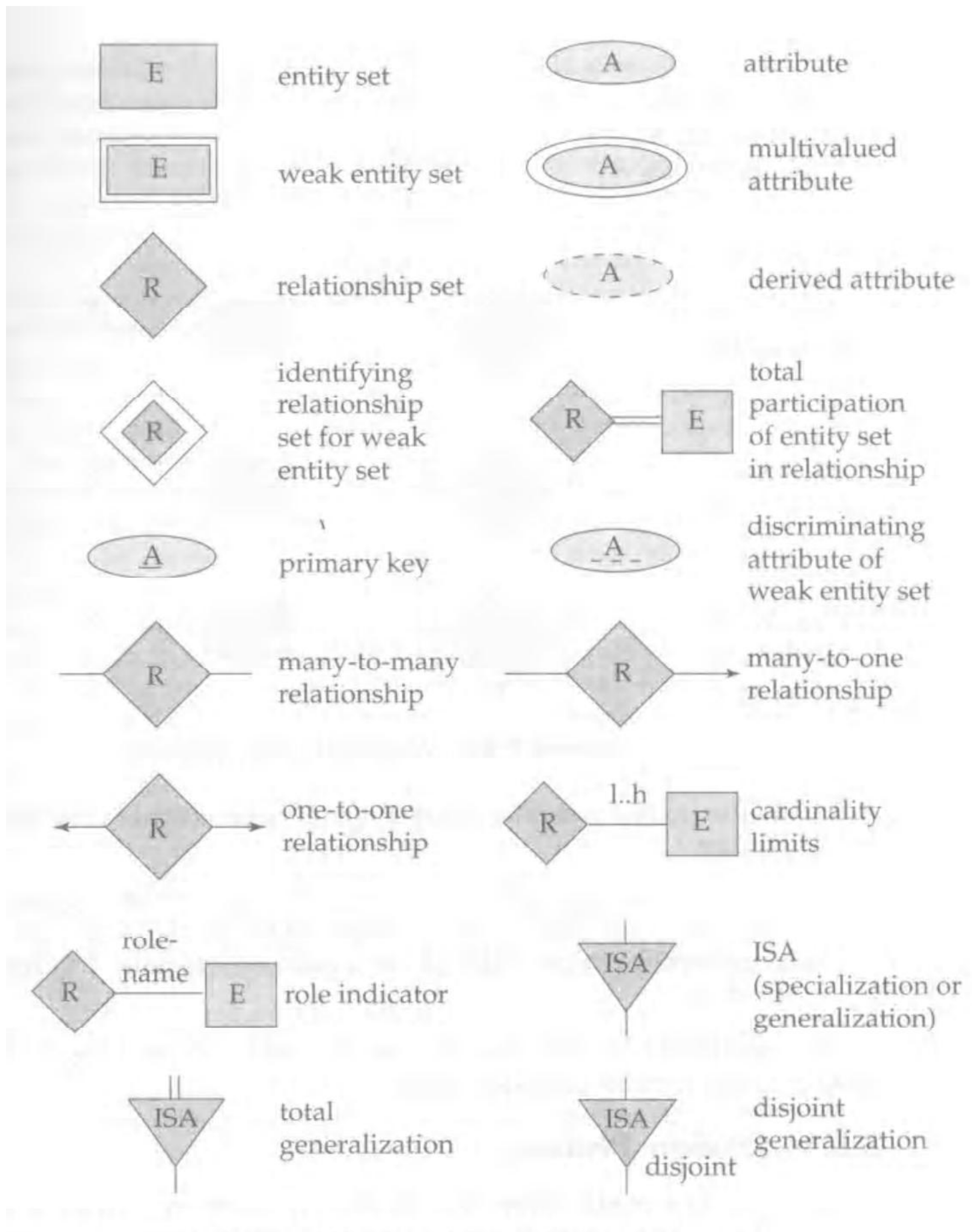
E-R diagram with redundant relationships:



E-R diagram with aggregation:



Other symbols used in E-R model are:



Introduction to Relational Database

A database can be understood as a collection of related files. How those files are related depends on the model used. Early models included the hierarchical model (where files are related in a parent/child manner, with each child file having at most one parent file), and the network model (where files are related as owners and members, similar to the network model except that each member file can have more than one owner).

The relational database model was a huge step forward, as it allowed files to be related by means of a common field. In order to relate any two files, they simply need to have a common field, which makes the model extremely flexible.

Poet

Code	First Name	Surname	Age
1	Mongane	Afrika	62
2	Stephen	Serote	58
3	Tatumkhulu	Watson	29

Poem

Title	Poet
Wakening Night	1
Thrones of Darkness	2
Once	3

These two tables relate through the code field in the poet table, and the poet field in the poem table. We can see who wrote the poem 'Once' by following the relationship, and see that it was poet 3, or Tatumkhulu Watson.

In 1970, when E.F. Codd developed the model, it was thought to be hopelessly impractical, as the machines of the time could not cope with the overhead necessary to maintain the model. Of course, hardware since then has come on in huge strides, so that

today even the most basic of PC's can run sophisticated relational database management systems.

Together with this went the development of SQL. SQL is relatively easy to learn and allows people to quickly learn how to perform queries on a relational database. This simplicity is part of the reason that relational databases now form the majority of databases to be found.

Basic Terms

An understanding of relational databases requires an understanding of some of the basic terms.

- Data are the values stored in the database. On its own, data means very little. "43156" is an example.
- Information is data that is processed to have a meaning. For example, "43156" is the population of the town of Littlewood.
- A database is a collection of tables.
- Each table contains records, which are the horizontal rows in the table. These are also called tuples.
- Each record contains fields, which are the vertical columns of the table. These are also called attributes. An example would be a product record.
- Fields can be of many different types. There are many standard types, and each DBMS (database management system, such as Oracle or MySQL) can also have their own specific types, but generally they fall into at least three kinds - character, numeric and date. For example, a product description would be a character field, a product release date would be a date field, and a product quantity in stock would be a numeric field.
- The domain refers to the possible values each field can contain (it's sometimes called a field specification). For example, a field entitled "marital status" may be limited to the values "Married" and "Unmarried".
- A field is said to contain a null value when it contains nothing at all. Fields can create complexities in calculations and have consequences for data accuracy. For this reason, many fields are specifically set not to contain NULL values.

- A key is a logical way to access a record in a table. For example, in the product table, the product_id field could allow us to uniquely identify a record. A key that uniquely identifies a record is called a primary key.
- An index is a physical mechanism that improves the performance of a database. Indexes are often confused with keys. However, strictly speaking they are part of the physical structure, while keys are part of the logical structure.
- A view is a virtual table made up of a subset of the actual tables.
- A one-to-one (1:1) relationship occurs where, for each instance of table A, only one instance of table B exists, and vice-versa. For example, each vehicle registration is associated with only one engine number, and vice-versa
- A one-to-many (1:m) relationship is where, for each instance of table A, many instances of the table B exist, but for each instance of table B, only once instance of table A exists. For example, for each artist, there are many paintings. Since it is a one-to-many relationship, and not many-to-many, in this case each painting can only have been painted by one artist.
- A many to many (m:n) relationship occurs where, for each instance of table A, there are many instances of table B, and for each instance of table B, there are many instances of the table A. For example, a poetry anthology can have many authors, and each author can appear in many poetry anthologies.
- A mandatory relationship exists where, for each instance of table A, one or more instances of table B must exist. For example, for a poetry anthology to exist, there must exist at least one poem in the anthology. The reverse is not necessarily true though, as for a poem to exist, there is no need for it to appear in a poetry anthology.
- An optional relationship is where, for each instance of table A, there may exist instances of table B. For example, a poet does not necessarily have to appear in a poetry anthology. The reverse isn't necessarily true though, for example for the anthology to be listed, it must have some poets.
- Data integrity describes the accuracy, validity and consistency of data.
- Database normalization is a technique that helps us to reduce the occurrence of data anomalies and poor data integrity.

UNIT II – RELATIONAL MODEL

Relational Model:

In this model data and their relationship are represented using a "table". A relation is used to represent a table.

Tuple → Each row in a table is called a tuple

Attribute → each column in a table is known as a attribute.

Domain → The set of all possible value of an attribute is called a domain.

Degree → The number of attributes in a table is referred to as degree.

Key → the minimal set of attributes used to uniquely define any row in a Table is called a key.

Catalogs for Relational DBMS

- The information stored in a catalog of an RDBMS includes th relation names, attribute names, and attribute domains(data types) as well as descriptions of constraints(primary keys, foreign keys, NULL, NOT NULL, and other types of constraints), views and storage structures and indexes.
- Security an authorization information is also kept in the catalog; this describes each user's privileges to access specific database relations and views, and the creator or owner of each relation.
- In relational DBMS it is common practice to store the catalog itself as relations and to use the DBMS software for querying, updating, and maintaining the catalog.
- A possible catalog structure for base relation information which stores relation names, attributes names,attribute type, and primary key information.
- The primary key of REL_AND_ATTR_CATALOG is the combination of the attributes {REL_NAME,ATTR_NAME}, because all relation names should be unique and all attribute names with a particular relation should also be unique.

Key:

There are different types of keys .they are

1. Super key
2. Candidate key
3. Primary key
4. Foreign key

Super key:

Set of one or more attributes that taken collectively, allow us to identify uniquely a tuple or row in a table is known as super key.

Eg: banking database. Cust_id, cust_name, cust_st.

Super key--→ {cust_id} {cust name,cust_st}

{cust_name,cust_id}

Candidate key:

Super key for which no proper subsets are superkeys, such minimal superkeys are candidate keys.

Eg:

{cust_id}, {cust_name, cust_st}

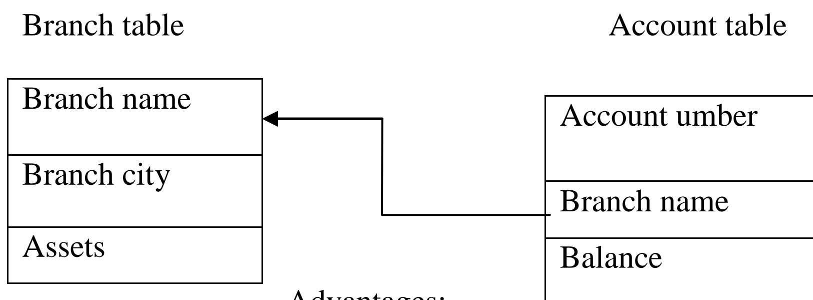
Primary key:

A single attribute is used to uniquely identify a row is called a primary key.

Eg: {cust_id}

Foreign key:

In database, tables are related with each other thro' a common attribute. An attribute in a table that references an attribute in another table is called a foreign key.



Advantages:

- Structural independence
- Simplicity
- Easy implementation, design, usage.
- Flexible

Disadvantages:

- Slower processing time than hierarchical and network models.
- Not good for transaction process compared to hierarchical and n/w model.

Relational Algebra:

The data stored in the database are retrived using a query language. A query Language is a language in which users requests information from the database.

There are two types of query language. They are

(i) Procedural (ii) non-procedural

Procedural:

The user instructs the system to perform a sequence of operations on the database to compute the derived results. (Relational algebra)

Non-procedural:

The user describes the desired information without giving specific procedures for obtaining that information. (Relational calculus)

Relational calculus

It is a formal query language where we can write one declarative expression to specify a retrieval request and hence there is no description of how to retrieve it. A calculus expression specifies what is to be retrieved rather than how to retrieve it. Relational calculus is considered to be non procedural language.

i) The domain relational calculus

The domain calculus differs from the tuple calculus in the type of variable used in formulas, rather than having variable range over tuples, the variable range over single values from domain of attributes.

An expression of the domain calculus is of the form $\{x_1, x_2,\ldots x_n/ \text{COND}(x_1, x_2,\ldots x_n, x_{n+1}\ldots x_{n+m})\}$

x_1 , x_2… are domain variable that range over domain of attributes.

COND – condition or formula of the domain relational calculus. A formula is made up of atoms.

1. An atom of the form $R(x_1,x_2\ldots x_j)$ R – name of a relation of degree j.each x_j, $1\leq i \leq j$ is a domain variable. This atom states that a list of values of $<x_1,x_2,\ldots x_j>$

must be a tuple in the relation whose name in R. x_i – value of the ith attribute value of the tuple.

2. An atom of the form x_i OP x_j

   OP is one of the comparison operators in the set {=, <, >, ≤, ≥, ≠} x_i and x_j are domain variables.

3. An atom of the form x_i OP C (or)

   C OP x_j

   C – constant

   Eg. Retrieve the birthdate and address of the employee whose name is 'ABC' {c,d / (a) (e) EMPLOYEE (a,b,c,d,e) and a = 'ABC'}

   A = name

   B = Eid

   C = Dob

   D = address

   E = dno

   {c, d / EMPLOYEE ('ABC' , b,c,d)}

   Eg. Retrieve the name and address of all employees who work for the 'Research' department.

   {a, d / (e)(p) (r) EMPLOYEE (a,b,c,d,e) and DEPARTMENT (pqr) and p = 'Research' and e = r)}

   [p - dname q - dept location r – dept.number]

ii) The Tuple relational calculus:

- The tuple relational calculus is based on specifying a number of tuple variables.
- Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as it value any individual tuple from that relation.
- A simple tuple relational calculus query is of the form.
- {t/COND(t)}

  t-tuple variable

  COND(t)- conditional expression involving it.
- The result of such a query is the set of all tuples t that satisfy COND(t).
- Eg. To find all employees whose salary is above 50,000

{t/EMPLOYEE(t) and t.Salary>50,000}

- Eg. To retrieve some of attributes.

  {t.FNAME,t.NAME/EMPLOYEE(t) and t.salary>50,000}

a)Expressions and formulas

A general expression of the tuple relational calculus of the form

$\{t.A_1,t_2.A_2,\ldots\ldots t_n.A_n/COND(t_1,t_2,\ldots.t_n,t_{n+1}\ldots.t_{n+m})\}$

$t_1,t_2,\ldots t_n,t_{n+1}\ldots t_{n+m}$-tuple variables

A_i-attribute of the relation on which t1 ranges.

COND is a condition or formula.

A formula is made up of predicate calculus atoms can be one of the following.

- ✓ An atom of the form $R(t_i)$

  R-relation name

  t_i-tuple variable

  The atom identifies the range of the tuple variable ti as the relation whose name is R.

- ✓ An atom of the form t_i. A OP t_j.B,OP is one of the comparison operation{=,<,>,>=,<=,≠}t_i and t_j are tuple variables.

  Where A- attributes of the relation on which t_i ranges.

  B-attributes of the relation on which tj ranges.

  OP-Relational operator

- ✓ An atom of the form t_i.A OP C or

  C OP t_j.B

  Where C- constant

  Each of the preceding atoms evaluate to either TRUE OR FALSE for a specific combination of tuples. This is called truth value of the atom.

b) Existential and Universal Quantifiers

A tuple variable t is bound if it is quantified, meaning that it appear in an (⁻ t) or (V t) clause otherwise it is free.

A tuple variabl in a formula as free or bound according to the following rules.

- An occurance of a tuple variable in a formula F that Is an atom is free in F.
- An occurrence of a tuple variable t is free or bound in a formula made up of logical connective (F_1 and F_2) , (F_1, or F_2) not (F_1) and not (F_2) depending on whether it is free or bound in F_1 or F_2.
- All free occurrence of a tuple variable t in F are bound in a formula F' of the forms

  F' = (T) (F) or F'= (t) (F)

  The tuple variable is bound to the quantifier specified inF'.
- If F is a formula, then (t) (F)

  T – tuple variable

  The formula (t) (F) is TRUE if the formula F evaluates to TRUE for some (atleast one) tuple assigned to free occurrences of t in F'; otherwise (t) (F) is FALSE.
- If F is a formula, then (t) (F)

  T – tuple variable

  The formula (t) (F) is TRUE if the formula F evaluates to TRUE for some tuple assigned to free occurrences of t in F'; otherwise (t) (F) is FALSE.

Eg. Retrieve the name and address of all employee who work for 'Research' department.

{t.Fname, t.Lname, t.Address/EMPLOEE (t) and (d) DEPARTMENT (d) and e.name = 'Research' and DNUMBER = t.DNO)}

Using the Universal Quantifier

Eg. Find the names of employees who work on all the projects controlled by department number 5.

{e.name, e.Fname/EMPLOYEE (e) and ((X) (not (project(x)) or not (X.DNUM = 5) or ((w) (WORKS _ ON (w) and w.Eid = e.Eid and X.PNUMBER=w.PNO))))}

Safe expressions:

A safe expression in relational calculus is one that is guaranteed to yield to a finite number of tuples as a result otherwise the expression is called unsafe.

Eg: {t/not (EMPLOYEE (t))}

Is unsafe because it yield all tuples in the universe that are not EMPLOYEE tuples, which are infinitely numerous.

We can define safe expressions more precisely by introducing the concept of the domain of a tuple relation calculus expression. This is the set of all values that either appear as constant values in the expression or exist in any tuple of the relations reference in the expression.

An expression is said to be safe if all values in its result are from the domain of the expression.

Relational Algebra:

It's a procedural language that consists of set of operations that take one or more relations as input and produces new relation as output.

- Basic operations
- Additional operations
- Extended operations

Fundamental Operations

Basic operations:

* Select
* Project
* Union
* Rename
* Cartesian products
* Intersection
* Join
* Set difference

Union: U

Let us consider two relations "Depositor" and "Borrower". Union function includes all the tuples that are either in depositor or borrower or in both. Duplicates are eliminated.

Name	CITY
Hayes	Pune
Johnson	Mumbai
Jones	Solapur
Smith	Nashek

Depositor Relation

Name	City
Adams	Mumbai
Hayes	Pune
Jackson	Solapur
Smith	Nashek

Borrower relation

Name	City
Hayes	Pune
Adams	Mumbai
Johnson	Mumbai
Jackson	Solapur
Jones	Solapur
Smith	Nashek

Depositor U Borrower

Intersection: ∩

This operation includes all the tuples that are in both depositor and borrower relation. (i.e) tuples that is common in both the relations.

Name	City
Hayes	Pune
Smith	Nashek

Depositor ∩ borrower

Difference:

Depositor – Borrower. It contains all tuples in depositor but not in borrower.

Name	CITY
Johnson	Mumbai
Jones	Solapur

Cartesian product:

It is also known as cross product or cross join.

Code	Name
1	Mc.Grawhill
2	PHA
3	Pearsons

ID	Title
1	DBMS
2	Compiler

Code	Name	Id	Title
1	Mc.Grawhill	1	DBMS
2	PHA	1	DBMS
3	Pearsons	1	DBMS
1	Mc.Grawhill	2	Compiler
2	PHA	2	Compiler
3	Pearsons	2	Compiler

Select: σ

This operation selects the tuples that satisfy the given predicate (condition).

σ <select condition> (R)

<select condition> → <attribute name> <comparison operator> <constant value>

< attribute name>

(R) -→Name of the table.

Eg: σ year = 2000;

Id	Title	Author	Year
1	DBMS	Silberscatz	2000
2	networks	Ferouzan	2000

Project: π

Selects certain columns from the table while discarding others.

Π <attribute list> (R)

Eg: π title, Arthur (book)

It displays the entire column title and Arthur from the relation table.

Select and Project operation:

Eg. Π title (σ price>300 (book))

This displays the title of the book having price greater than 300.

Rename:-

Either the attribute or the relation or both can be renamed.

ρ_s (new attribute names) (R) --- renames only the attribute

Old relation

New relation. ρ_s (R) --- renames only the relation

Additional operations:

- ✓ Natural join operation
- ✓ Assignment operation
- ✓ Division operation

Natural- join operation:

It is a combines selection and Cartesian product into one operation. The difference between Cartesian and natural join is selection operation is performed on the result of the Cartesian product.

$\Pi_{\text{empname, salary}} (\sigma_{\text{emp.empcode ==salary.empcode}} \text{(emp X salary))}$

Natural join query:

$\Pi_{\text{empname, salary}} \text{(emp salary)}$

Empcode	Empname
E 1	Hari
E 2	Om
E 3	Smith
E 4	Joy

Empcode	Salary
E 1	2000
E 2	5000
E 3	7000
E 4	10000

Empname	Salary
Hari	2000
Om	5000
Smith	7000
Joy	10000

Division Operation: ÷ **"for all"**

Two steps are involved in this operation. Let us consider banking example.

Query:

Find all the customers who have an account at all the branches located in Chennai.

ACCOUNT

Account	Branch name	Balance
101	Alwarpet	5000
102	Pune	4000
201	T.Nagar	9000
215	Marine	7000
217	North town	75000
222	Palladam	7000
305	Hyderabad	3500

Depositor

Cust name	Accno
harris	102
Johnson	101
Jones	201
Laurel	217
Smith	222
Turner	215
Rodger	305

Branch

Branch name	Branch city	Assets
T.Nagar	Chennai	7000
Alwarpet	Chennai	8000
Marine	Hyderabad	6000
North town	Ramanathapuram	15000
palladam	Hyderabad	9000

Step 1:

Find all the branch names in Chennai.

Π branch name(σ branch city-“Chennai”(branch))

Branch name
Alwarpet
T.nagar

Step: 2

Find all the customers in all the branches.

Π cust name, branch name(depositor account)

cust name	branch
Johnson	alwarpet
Harris	Pune
Johnson	T.nagar
Jones	North town
Smith	Marine

cust name
Johnson

Step: 3 r2 ÷ r1

Assignment operator: **‘←’**

It works like assignment in programming language

Extended relational algebra operations:

- Generalized projections
- Aggregate function
- Outer join

Generalized functions:

- Extended projection

Π fi,f2,f3….fn (E) where E is the relational algebra expression.

Π name, total/5 as percentage (student)

Roll no	Name	total
1	Hari	350
2	Om	400
3	Jay	375
4	Smith	425

Name	percentage
Hari	70
Om	80
Jay	85
Smith	75

salary
30000

dept	salary
comp	18000
It	12000

salary
75000

Aggregate function:

-Takes a collection of value and return a single value. Eg. Sum, average, count-distinct.

Empcode	Name	Salary	Dept
1	Hari	10000	Comp
2	Om	7000	It
3	Smith	8000	Comp
4	Jay	5000	It

Ç sum (salary) (emp salary)

Ç avg (salary) (emp salary)

Ç count-distinct (dept) (emp salary)

Outer join:

-Extension of join operation that deals with missing information.

Name	City
Hari	pune
Om	Mumbai
Smith	Nashik
Jay	Solapur

Name	Dept	salary
Hari	Comp	10000
Om	It	7000
Bill	Comp	8000
Jay	It	5000

Emp empsalary

Name	City	Dept	Salary
Hari	Pune	Comp	10000
Om	Mumbai	It	7000
Jay	Solapur	It	5000

Emp empsalary

Name	City	Dept	Salary
Hari	Pune	Comp	10000
Om	Mumbai	It	7000
Jay	Solapur	It	5000
Smith	Solapur	Null	null

Emp empsalary

Name	City	Dept	Salary
Hari	Pune	Comp	10000
Om	Mumbai	It	7000
Jay	Solapur	It	5000
Bill	null	Comp	8000

Emp empsalary

Name	City	Dept	Salary
Hari	Pune	Comp	10000
Om	Mumbai	It	7000
Jay	Solapur	It	5000
Smith	Nashik	Null	Null
Bill	null	Comp	8000

Modification of database:

* Delete:

- removes the selected tuples from the database. Only tuples can be deleted not values of any particular attribute.

$r \leftarrow r - E$

employee $\leftarrow$ employee - σ empname="smith" (employee)

empsalary $\leftarrow$ employee - σ dept="IT" (empsalary)

* Insertion:

$r \leftarrow r \cup E$

employee $\leftarrow$ employee U { "John", "Nagpur")}

employee $\leftarrow$ empsal U {("John" , "Computer", 6000)}

* Updating:

To change only one value in the tuple without changing all values in the tuple.

$r \leftarrow r - E$

$\Pi_{fi,f2,f3 \ldots fn\ (r)}$

empsalary $\leftarrow$ Π empname, dept, sal*1.05(empsalary)

empsalary $\leftarrow$ Π name, dept, sal*1.05(σ sal<=6000(empsalary)) U

Π empname, dept, sal*1.03(σ sal>6000(empsalary))

SQL Fundamentals:

The IBM developed the original version called "Sequel". The Sequel language has changed to "Structured Query Language". In 1986 ANSI and ISO published a SQL standard called SQL – 86. The most recent version is 2003.

SQL has several parts:

- Data Definition Language (DDL):
  It provides commands for defining schemas, defining relations and modifying relation schemas.
- Interactive Data Manipulation Language (DML):
  SQL DDL includes a query language based on both the relational algebra and the tuple relational calculus. It includes also commands to insert tuples into, delete tuples from and modify tuples in the database.

- Integrity:
  The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed.
- View Definition:
  The SQL DDL includes commands for defining views.
- Transaction Control:
  SQL includes commands for specifying the beginning and ending of transactions.
- Embedded SQL and Dynamic SQL:
  They define how SQL statements can be embedded within general purpose programming languages such as C, C++, and java, COBOL, Pascal and FORTRAN.
- Authorization:
  The SQL DDL includes commands for specifying access rights to relations and views.

Basic Structure:

SQL allows the use of null values to indicate that value is either unknown or does not exist. It allows a user to specify which attributes cannot be assigned null values.

The basic structure of SQL consists of 3 clauses:

1. Select - It corresponds to the projection operation of the relational algebra. It is used to list the attributes designed in the result of the query.

2. From - It corresponds to the Cartesian product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.

3. Where - It corresponds to the selection operation of the relational algebra. It consist of predicate involving attributes of the relations that appear in the from clause.

```
Select A1, A2,…….An
From r1, r2, ……….rm
Where p;
```

The Select Clause:

The select query can be used in different formats.

(*) select branch_name from loan;

-the result is a relation consisting of a single attribute with the heading branch_name.

(*) select distinct branch_name from loan;

- the duplicates present in the attribute branch_name will be eliminated.

(*) select all branch_name from loan;

-duplicates will also be displayed. Since it is default it is not necessary to use “all” .

The select clause may also contain expression involving the operators +, -, /, * operating on constants or attributes of tuples.

Select loan_no, branch_name, amount * 100 from loan;

The Where Clause:

SQL uses logical connectives “and “, “or “, “not “rather than mathematical symbols in the where clause.

```
Select loan_no
From loan
Where branch_name = “parris” and amount>1000;
```

SQL includes a “between “comparison operator to simplify where clauses that specify that a value ≤, ≥ etc.

```
Select loan_no                               select loan_no
From loan                                    from loan
Where amount<=10000 and amount>= 9000        where amount between 9000 and 10000
```

The From Clause:

The from clause by itself defines a Cartesian product of the relations in the clauses.

```
Select cust_name, borrower.loan_no, amount
From borrower, loan
Where borrower.loan_no = loan.loan_no
```

Relation name. Attribute name

This query finds the name, loan_no, loan amount of all the customers who have a loan from the branch.

```
Select cust_name, borrower.loan_no, amount
From borrower, loan
Where borrower.loan_no = loan.loan_no and branch_name = “ Parris”;
```

This query finds the cust_name, loan numbers and loan amount for all the loans at the branch “ Parris”.

The Rename operation:

SQL provides a mechanism for renaming both relations and attributes. For renaming, it uses “as”.

Syntax: oldname as newname

The “as” clause can appear in both the select and from clause.

```
Select cust_name, borrower.loan_no as loan_id, amt
From borrower, loan
Where borrower.loan_no = loan.loan_no;
```

Tuple Variable:

Tuple variables are most useful for comparing two tuples in the same relations.

```
Select distinct T. branch_name
From branch as T, branch as S
Where T.assets > S. assets and S. branch_city = " Bangalore";
```

This query is used to find the names of all branches that have assets greater than atleast one branch located in “Bangalore”.

```
Select cust_name, T.loan_no, S.amount
From borrower as T, loan as S
Where T.loan_no = S. loan_no;
```

This query is used to find all customers who have a loan from the bank, find their names, loan numbers and loan amount.

String Operations:

SQL specifies strings by enclosing them in single quotes for e.g. ‘Parris’. The most commonly a used operation on strings is pattern matching using the operator “like”. The patterns can be described by using two special characters.

% → the % character matches any substring’
- → The - character matches any character patterns are case sensitive.
‘Dbms%’ → matches any strings beginning with dbms.
‘%idge% → matches any string containing idge as substring. E.g. cartridge, bridge.
‘---‘ → matches any substring of exactly 3 characters.
‘---%’ → matches any string of atleast 3 characters.

```
Select cust_name
From customer
Where cust_st like '% main';
```

This query finds the names of all customers whose street address includes the substring ‘ main’. Other operators are “escape”, “not like”, “similar to” etc.

Ordering the display of tuples:

The order by clause causes the tuples to in result of a query to appear in sorted order.

```
Select distinct cust_name
From borrower, loan
Where borrower. Loan_no = loan.loan_no and branch_name = 'parris'
Orderby cust_name;
```

This query lists the name of the customer in alphabetical order who have loan at the parris branch. By default the orderby clause lists items in ascending order. To specify the sort order (i.e.) ascending or descending we specify “asc” and “desc”.

```
Select *
From loan
Orderby amount desc, loan_no asc;
```

If several loans have the same amount, we order them in ascending order.

SET Operations:

The following are the set operations.

- Union
- Intersect
- Except

Union operation:

The following query finds all the customer having a loan, an account or both at the bank.

```
(Select cust_name
From depositor)
Union
(Select cust_name
From borrower)
```

The union operation automatically eliminates duplicates. If duplicate relation should be allowed then "union all" should be used instead of union.

```
(Select cust_name
From depositor)
Union all
(Select cust_name
From borrower)
```

Fro e.g. if James has 3 accounts and 2 loans at same bank then there will be 5 tuples with the name James.

Intersection Operation:

The following tuple is used to find all customers who have both a loan and an account at the bank.

```
(Select distinct cust_name
From depositor)
Intersect
(Select distinct cust_name
From borrower)
```

The intersect operation automatically eliminates duplicates. If the duplicates want to be retained then use "intersect all".

```
(Select distinct all cust_name
From depositor)
Intersect all
(Select cust_name From borrower)
```

The number of tuples that appear in the result is equal to the minimum no of duplicates in both depositor and borrower.

The Except operation:

This query finds all the customers who have an account but no loan at the bank.

```
(Select distinct cust_name
From depositor)
Except
(Select cust_name
From borrower)
```

The except operation automatically eliminate duplicates. To retain duplicates "except all" should be used.

```
(Select cust_name
From depositor)
Except all
(Select cust_name
From borrower)
```

The number of duplicate copies of a tuple in the result is equal to the no of duplicate copies of the tuple in depositor minus the no of duplicate copies of the tuple in borrower, provided that the difference is positive.

Aggregate Function:

Aggregate functions are function that take a collection of values as input and return a single value. The 5 built in aggregate functions are

Average (avg)
Minimum (min)
Maximum (max)
Total (sum)
Count (count)

The input to sum and avg must be a collection of numbers, but other operators can operate on collections of non- numeric data types.

```
Select avg (balance)
From account
Where branch_name= 'parris'
```

This query is to find the average account balance at the 'parris' branch.

```
Select branch_name, avg (balance)
From account
Groupby branch_name= 'parris'
```

This query finds the average account balance at each branch.

```
Select branch_name, avg (balance)
From account
Groupby branch_name= 'parris'
Having avg (balance) > 1200
```

After grouping has been done, aggregate functions are used using having.

Null Values:

The null values are used to indicate the absence of information about the value of an attribute.

```
Select loan_no
From loan
Where amount is null.
```

To find all loan numbers that appear in the loan numbers that appear in the loan relation with null values for amount.

How SQL handles Null values?

If any of the input values is null then the result of an arithmetic expression is null.

- AND ➔ true and known → unknown
  False and unknown → false
  Unknown and unknown → unknown.
- OR ➔true or unknown → true
  False or unknown → unknown
  Unknown or unknown → unknown
- NOT ➔ not unknown → known

Nested Subqueries:

A subquery is a select- from- where expression that is nested within another query. A common use of subqueries is to perform tests for a set membership, make set comparisons and determine set cardinality.

➔ **Set membership:**

```
Select distinct cust_name
From borrower
Where cust_name in (select cust_name from depositor)
```

The output of this query is the cust_name who are in borrower from bank and who appear in the list of account holders.

```
Select distinct cust_name
```

From borrower
Where cust_name not in (select cust_name from depositor)

Select distinct cust_name
From borrower
Where cust_name not in (‘smith’,’ jones’)

➔ **Set comparison**:

Select distinct T.branch_name
From branch as T, branch as S
Where T.asset> S.asset and S. branch_city =’Brooklyn”

This query finds the names of all branches that have assets greater than those of atleast one branch located in Brooklyn.

“some” → greater than atleast one.

Select branch_name
From branch
Where assets > some (select assets from branch where branch_city =’Brooklyn’)

Generates the set of all assets values for all branches in Brooklyn

Integrity:

A value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. This condition is called “referential integrity”.

Data Constraints:

1. Column level constraints:
   Constraints are defined along with column definition. It can be applied to any column.

2. Table level constraints:
   Constraint span across multiple columns. Data constraint attached to a specific column in a table reference the content of another column in the table.
   (i) Null value concept:
   If a column lacks a data value or the value is unknown it is said to be null. It is not equivalent to zero. If defines as not null then the user must enter a value.
   (ii) Primary key concept:
   One or more columns in a table is/are used to uniquely identify each row in a table. The value should not be null.

3. Unique key concept:
   To ensure the information in the column for each record is unique. A table can have many unique keys.

Create table cust(code number(5) primary key, name varchar (10) not null, addr varchar(30) not null, licence no varchar(15) constraint ukln unique);

4. Default value concept:

If the column has some values to enter and left empty then it is assigned with a default value.

5. Foreign key concept:

The foreign key represents relation between tables. Foreign key is related to primary key of another table while referencing the data types should match.

Create table depositor
(Cust_name char (20), account_no char (10), primary key (cust_name, accno), foreign key (cust_name) references customer, foreign key(accno) references account)

6. Check integrity constraint:

It defines a condition that every row must satisfy. There can be more than one check constraint in a column. It can be defined both at column and table level constraint.

Create table branch
(Branch_name char (15), branch city char (30), assets integer, primary key (branch_name),
check(assets>=0))

Assertions:

An assertion is a predicate expressing a condition that the database always to satisfy.

Create assertion <assertion name> check <predicate>

e.g.

Create assertion sum- constraint check (not exits
(Select * from branch where (select sum (amount) from loan
Where loan.branch_name= branch.branch_name) > = (select sum (balance) from account
where account.branch_name=branch.branch_name)))

Triggers:

A trigger is a statement that the system executes automatically as a side effect of a modification of a database. To design a trigger mechanism, we must meet two requirements.

Specify when a trigger is to be executed. This is broken up into an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.

Specify the action to be taken when the trigger executes. The database stores triggers just as if they were regular data, so that they are persistent and are accessible to all database operations. Whenever a specified event occurs and corresponding condition satisfied the database executes the trigger.

Need For triggers:

Consider the banking system, suppose that instead of allowing negative balances, the bank deals with overdrafts by setting the account balance to zero and creating a loan in the amount of the overdraft. The bank gives this loan a loan_no identical to the account no of the account.

Suppose that Jones withdrawal of some money from an account made the account balance negative. Let't' denote the account tuple with a negative balance value. The actions to be taken are

- insert a new tuple s in the loan relation with
    - s[loan_no = t[accno]
    - s[branch_name] = t[branch_name]
    - s[amount] = -t[balance]
- insert a new tuple u in the borrower relation with
    - u[cust_name] = "Jones"
    - u[loan_no] = t[accno]
- set t[balance] to 0.

Thus for these reasons a trigger is needed to notify the changes made in the database.

Triggers in SQL:

- The triggering events and action can taken many forms
- The triggering event can insert and delete.
- For updates, the trigger can specify columns whose update causes the trigger to execute.
- Triggers can be activated before the event (insert/delete/update) instead of after the events.
- Instead of carrying out an action for each affected row, we can carry out a single action for the entire SQL statement that caused the insert/delete/update.

When not to use trigger?

1. Triggers should be written with great care, since a trigger error detected at run time causes the failure of the insert/delete/update statements that set off the trigger. The action of one trigger can set of another trigger. In worst case this could even load to an infinite chain of triggering.

2. The insert action then triggers yet another insert action, and so on. Database systems typically limit the length of such chains of triggers and consider longer chains of triggering an error

3. Triggers are occasionally called rules or active rules but should not be confused with datalog rules.

Security:

The data stored in the database need protection from unauthorized access and destruction or alteration. The following are the ways in which data may be misused or made inconsistent and the mechanism to guard against such occurrences

Security violations:

Among the forms of malicious access are

- Unauthorized reading of data (theft of information)
- Unauthorized modification of data.
- Unauthorized destruction of data.

Database security:

It refers to protection from malicious access. To protect the database security measures to be taken at several levels.

(i) Database systems:
Some database systems users may be authorized to access only a limited portion of the database. Other users may be allowed to issue queries but not to modify the data. It is the responsibility of the database system to ensure that these authorization restrictions are not violated.

(ii) Operating Systems:
Weakness in operating system security may serve as a means of unauthorized access to the database.

(iii) Network:
Since almost all db systems allow remote access through the terminals or networks, software level security within the network software is an important as physical security, both on internet and in private networks.

(iv) Physical:
Sites with computer systems must be physically secured against armed or entry by intruders.

(v) Human:
Users must be authorized carefully to reduce the chance of any user giving access to a intruder in exchange for a bribe or other favors.

Advanced SQL Features:

Create table extension:

An application often requires creation of tables that have the same schema as an existing table. SQL provides a “create table like “extension to support this task.

```
Create table temp_account like account
```

The above statement creates a new table temp_account which has the same schema as account.

When writing a complex query, it is often useful to store the result of a query as a new table, the table is usually temporary. Two statements are required, one to create the table (with appropriate columns) and the second to insert the query result into the table.

SQL provides a simpler technique to create a table containing the results of a query. For example, the following statements creates a table t1 containing the results of a query.

```
Create table t1 as
(Select *
From account
Where branch _name =’Perryridge”)
With data
```

By default, the names and data types of the column are inferred from the query result. Names can be explicitly given to the columns by listing the column names after the relation name. If the with data class is omitted, the table is created but not populated with data.

The above create table ….as statement closely resembles the create view statement and both are defined by using queries. The main difference is that the contents of the table are set when the table is created; where as the contents of a view always reflects the current query result.

More on subqueries:

SQL allows subqueries to occur whenever a value is required, provided the subquery is returns only one value. Such subqueries are called scalar subqueries. For example a subquery can be used along with select clause.

```
Select cust_name
(Select count (*)
From account
Where account.customer_name= customer.cust_name) as num_accounts
From customers
```

The above example lists all the customers along with the number of accounts they own. This subquery return only a single value since it has a count (*) aggregate without a groupby.

Subqueries in the form clause cannot normally access attributes of other relations in the form clause. SQL supports a lateral clause that allows a subquery in the form clause to access attributes of preceding subqueries in the form clause.

Thus the above query could be written alternatively

```
Select cust_name, num_accounts
        From customer
                Lateral (select count (*)
                        From account
                        Where account.cust_name= customer.cust_name)
                        As this_customer (num_accounts)
```

Advanced Constructs for Database Update:

```
Update account set balance = balance +
                (Select amount
                From funds_received
                Where funds_received.account_number = account.account_number)
        Where exists
                (Select *
                From funds_received
                Where funds_received.account_number = account.account_number)
```

The condition in the where clause of the update ensures that only accounts with corresponding tuples in funds_received are updated, while the subquery within the set clause computes the amount to be added to each such account.

There is a table called “master table” where the updates are received a s a batch. It has to be correspondingly updated. SQL provides a special construct, called the “merge” construct to simplify the task of performing such merging of information. The above example can be expressed using merge as follows

```
        Merge into account as A
        Using (select *
                From funds_received) as F
                On (A.account_number = F.account_number)
```

When matched then
Update set balance = balance+ F. amount

The merge statement can also have a when matched then clause, which permits insertion of new records into the relation.

When not matched then
Insert values (F.account_number, null, F.amount)

Embedded SQL

- Embedded SQL's are SQL statements included in the programming language.
- Programming language in which the SQL statements are included is called the Host language. Some of the host languages are C,COBOL,PASCAL,FORTRAN,PL/I etc.
- This embedded SQL source code is submitted to an SQL precompiler, which processes the SQL statements.

Embedded SQL features

1. Embedded SQL statements appear in the host language. SQL statements are written in uppercase or lowercase.
2. Embedded SQL statements are prefixed by a delimiter EXEC SQL.
   Embedded SQL statements extend over multiple lines, the host language,strategy for statement continuation is used.
   Every embedded SQL statement is terminated with a delimeter. In COBOL, it's END_EXEC.
   In Adc,C,pascal,PL/I, it's a semicolon.
   SQL statements can include reference to host variables. Such reference must be prefixed with a colon(:) to distinguish them from names of SQL objects like column names.
   Host variables and SQL colomns can have the same name.

Advantages of embedded SQL programs:

1. Mixing of SQL statements with the programming language statements is an efficient way of merging the strength of two programming environment.
   Programming language provides the flow of control, host variables, book structure, conditional branching , loop facilities and input /output functions etc.
   SQL handles the database access and manipulation.
2. The use of the precompiler shifts the CPU intensive passing and optimization to the development phase. So, the resulting executable program will be very efficient in the CPU usage.
3. The program's run-time interface to the private database routines is transparent to the application programmer.

Dynamic SQL

- The dynamic SQL component of SQL allows programs to construct and submit SQL queries at run time.
- In contrast, embedded SQL statements must be completely present at compile time, they are compiled by the embedded SQL preprocessor.
- Using dynamic SQL, programs can create SQL queries as strings at run time and can either have them executed immediately or have them prepared for subsequent use.
- SQL defines standards for embedding dynamic SQL calls in a host language, such as C, as in the following example.

```
Char*sqlprog = “update account set balance = balance * 1.05 where account_n0=?”

EXEC SQL prepare dynprog from :sqlprog;

Char account [10] = “A-101”;

EXEC SQL execute dynprog using account
```

The dynamic SQL program contains a? , which is a place holder for a value that is provided when the SQL program is executed.

Missing Information

How To Handle Missing Information Without Using Nulls

The person identified by Id is called Name and has the job of a Job, earning Salary pounds per year.
1237 Davinder ? ?
1236 Cindy ? 70,000
1235 Boris Banker ?
1234 Anne Lawyer 100,000
Id Name Job Salary
That predicate is at best approximate.
VARCHAR(20) and that of Salary DECIMAL(6,0). But NULL isn’t a value of type VARCHAR(20), nor of type DECIMAL(6,0).

Decompose into 2 or more tables by projection Also known as normalization. Several degrees of normalization were described in the 1970s: 1NF, 2NF, 3NF, BCNF,

4NF, 5NF. The ultimate degree, however, is 6NF: "irreducible relations". "Vertical", because the dividing lines, very loosely speaking, are between columns. It is fundamental that the table to be decomposed can be reconstructed by joining together the tables resulting from the decomposition.

Ultimate decomposition can be thought of as reducing the database to the simplest possible terms. There should be no conjunctions in the predicates of the resulting tables. Vertical decomposition removes "and"s. (The relational JOIN operator is the relational counterpart of the logical AND operator.)

Vertical Decomposition of PERS_INFO

1237 Davinder
1236 Cindy
1235 Boris
1234 Anne
Id Name
1237 ?
1236 ?
1235 Banker
1234 Lawyer
Id Job
1237 ?
1236 70,000
1235 ?
1234 100,000
Id Salary
CALLED DOES_JOB EARNS
The person identified by Id is called Name.
The person identified by Id does the job of a Job
The person identified by Id earns Salary pounds per year.
The predicates for DOES_JOB and EARNS are still not really appropriate.

The purpose of such decomposition is to isolate the columns for which values might sometimes for some reason be "missing". If that Job column never has any question marks in it, for example, then the idea of recombining DOES_JOB and CALLED into a three-column table might be considered. By "never has any question marks", we really mean that at all times every row in CALLED has a matching row in DOES_JOB and every row in DOES_JOB has a matching row in CALLED (and no row in either table has a question mark).

"Person 1234 earns 100,000", "We don't know what person 1235 earns", and
"Person 1237 doesn't have a salary" are different in kind.
The suggested predicate, "The person identified by Id earns Salary pounds
per year", doesn't really apply to every row of EARNS.

Views:

A view is an object that gives the user a logical view of data from an underlying table or tables.

```
Create view V as < query expression>

Create view allcust as
(Select branch_name, cust_name
From depositor, account
Where depositor.accno = account.accno)
Union
(Select branch_name, cust_name
From borrower.loan
Where borrower.loan_no = loan.loan_no)
```

This query creates a view named allcust where we can view the names of the customer who is a depositor and in the loan account.

Introduction to Distributed Database Architecture

A distributed database system allows applications to access data from local and remote databases. In a homogenous distributed system, each database is an Oracle database. In a heterogeneous distributed system, at least one of the databases is a non-Oracle database. Distributed database uses client-server architecture to process information requests.

- Homogenous Distributed Database Systems
- Heterogeneous Distributed Database Systems
- Client-Server Database Architecture

Homogenous Distributed Database Systems

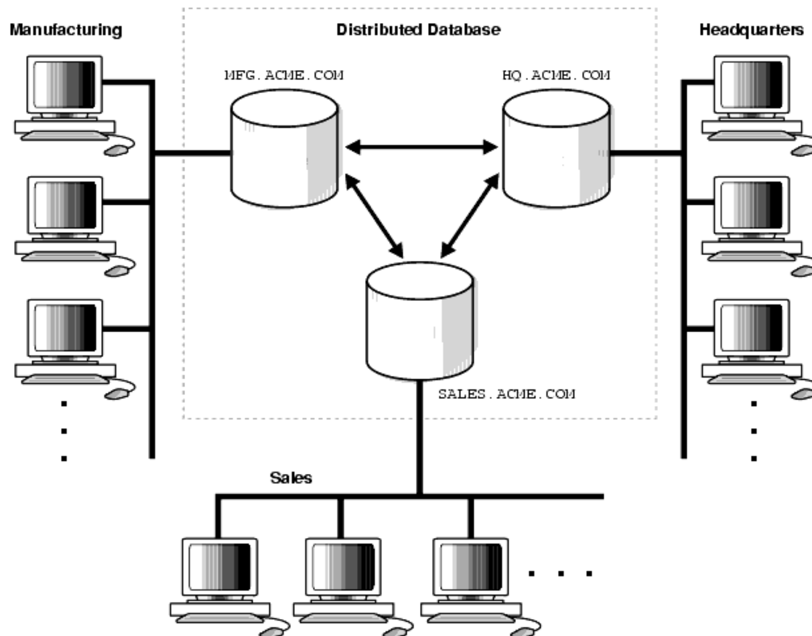
A homogenous distributed database system is a network of two or more Oracle databases that reside on one or more machines. Figure 30-1 illustrates a distributed system that connects three databases: HQ, MFG, and SALES. An application can simultaneously access or modify the data in several databases in a single distributed environment. For example, a single query on local database MFG can retrieve joined data from the PRODUCTS table on the local database and the DEPT table on the remote HQ database.

For a client application, the location and platform of the databases are transparent. You can also create synonyms for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database MFG yet want to access data on database HQ, creating a synonym on MFG for the remote DEPT table allows you to issue this query:

```
SELECT * FROM dept;
```

In this way, a distributed system gives the appearance of native data access. Users on MFG do not have to know that the data they access reside. s on remote databases.

Figure 30-1 Homogeneous Distributed Database



An Oracle distributed database system can incorporate Oracle databases of different versions. All supported releases of Oracle can participate in a distributed database system. Nevertheless, the applications that work with the distributed database must understand the functionality that is available at each node in the system--for example, a distributed database application cannot expect an Oracle7 database to understand the object SQL extensions that are only available with Oracle8i.

Heterogeneous Distributed Database Systems

In a heterogeneous distributed database system, at least one of the databases is a non-Oracle system. To the application, the heterogeneous distributed database system appears as a single, local, Oracle database; the local Oracle server hides the distribution and heterogeneity of the data.

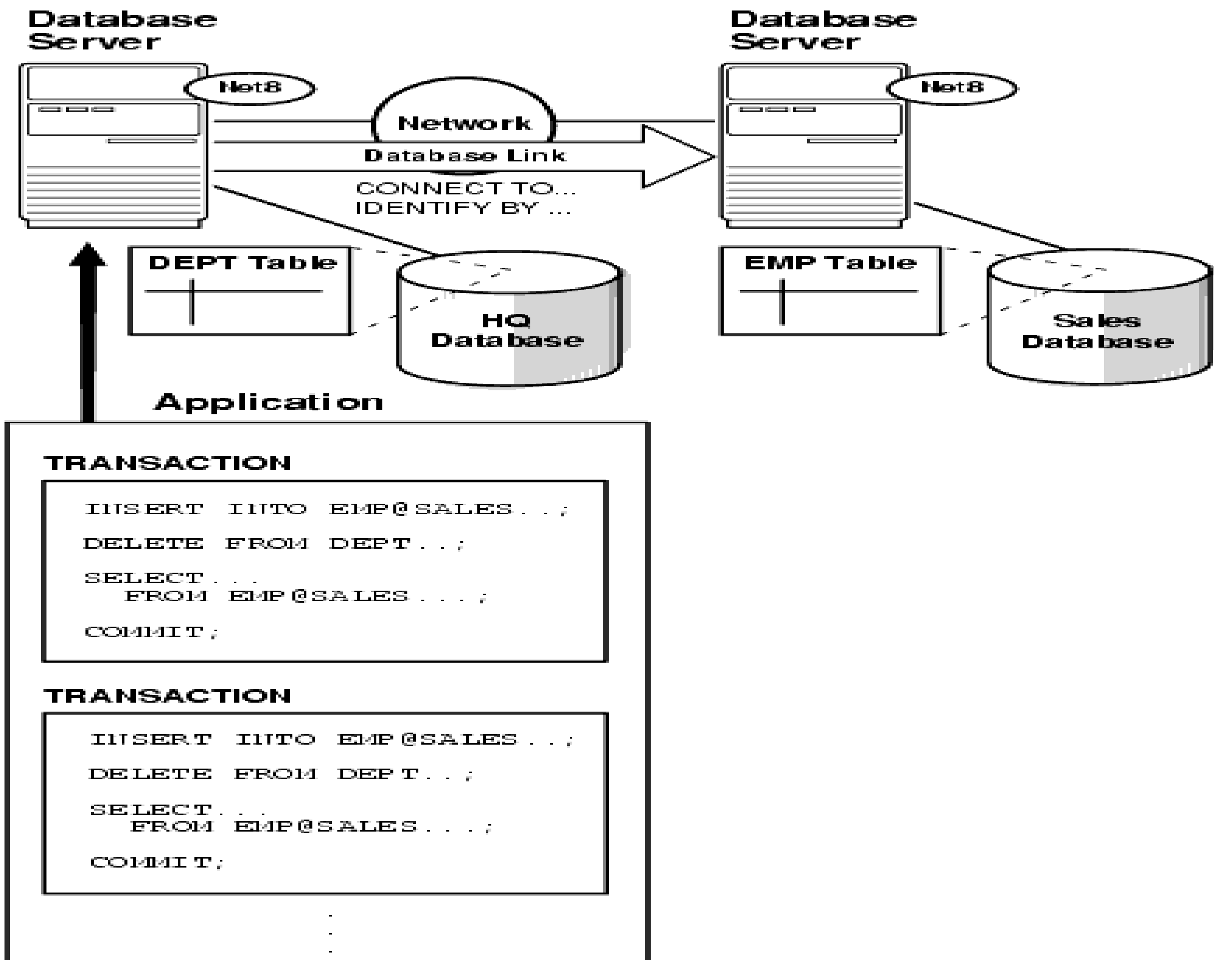
The Oracle server accesses the non-Oracle system using Oracle8i Heterogeneous Services and a system-specific transparent gateway. For example, if you include a DB2 database in an Oracle distributed system, you need to obtain a DB2-specific transparent gateway so that the Oracle databases in the system can communicate with it.

Client-Server Database Architecture

A database server is the Oracle software managing a database, and a client is an application that requests information from a server. Each computer in a network is a node that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.

The host for the HQ database is acting as a database server when a statement is issued against its local data (for example, the second statement in each transaction issues a statement against the local DEPT table), but is acting as a client when it issues a statement against remote data (for example, the first statement in each transaction is issued against the remote table EMP in the SALES database).

Figure 30-2 An Oracle Distributed Database System



Question Bank
2 Mark Questions:

1. What is Relational Algebra?
2. What is Relational Calculus?
3. How does Tuple-oriented relational calculus differ from domain-oriented relational calculus
4. How 'Natural – Join' operation is performed?
5. Why is a key essential? Write the different types of keys.
6. List the various join relations.
7. Define the term tuple
8. What is the difference between primary key and foreign key.
9. State the various operators used in relational algebra.
10. What is the difference between a key and a superkey
11. List the operations in relational algebra.
12. Define Query language. Give the classification of query language.
13. Distinguish between primary key and candidate key.
14. What is a view and how is it created? Explain with an example.
15. In what way is an embedded SQL different from SQL? Discuss.
16. What is embedded SQL? Explain briefly.
17. Name the different types of Joins supported in SQL.
18. What is static SQL? How does it differ from dynamic SQL?
19. Mention the advantages of embedded SQL?
20. Which condition is called referential integrity? Explain its basic concepts.
21. What is triggers in SQL?
22. Write about Assertions.
23. What are the different types of integrity constraints used in designing a relational database?
24. With an example explain referential integrity.
25. What is Domain Integrity Constraints? Give example
26. What are Entity Integrity Constraints?
27. Mention the different levels of security.
28. List out the forms of authorization on parts of the database.

16 Mark Questions

1. What are the relational algebra operations supported in SQL? Write the SQL statement for each operation.
2. Briefly explain about fundamental, additional operations in relational algebra with example?
3. Briefly explain about the modification of the database?
4. Explain about basic structure of SQL with example.
5. Write the Query using the following.
   Order by clause, in, not in, exist, as clause.
6. Briefly explain about set operations and aggregate functions.

UNIT III - DATABASE DESIGN

Functional dependencies

A functional dependency is a type of constraint that is a generalization of the notion of key.

Definition

- Consider a relation schema 'R' & let α c R & β c α.

The functional dependency α →β holders on schema 'R' , if in any Legal relation r(R), for all pairs of tuples t1 & t2 in r such that,

t1[α]= t 2[α], it's also the care that t1[β]=t2[β]

- Using the functional dependency notation, use say that 'k' us a superkey of R if K→R.

t1[k]=t2[k] it is also that,

t1[R]=t2[R] (i.e. t1=t2)

Consider the schema

Loan_info= (loan_no, branc_name, customer_name, amount)

The set of functional dependencies that hold on this relation are :

Loan-no → branch-name

Loan-no → amount

There is no functional dependency:

Loan- no → customer – name

As a giver loan can be made to more than 1 customers.

Use of functional dependencies:-

1. To test relation to see whether they are legal under a given set of functional dependencies. If a relation 'r' us legal under a set 'F' of functional dependencies, we say that 'r' satisfies F.
2. To specify constraints on the set of legal relations. If we wish to constrain ourselves to relations on schema R that satisfy a set F of functional dependencies, we say that F holds on K.

Let R be a relation on the relation scheme R. then R satisfies the functional dependency x → Y if a given set of values for each attribute in X uniquely determines each of the values of the attributes in Y. Y is said to be functionally dependent on X. The functional dependency (FD) is denoted as X → Y, where X is the left hand side or the determinant of the FD and Y is the right hand side of the FD>

A functional dependency X → Y is said to be trivial if Y X.

In order to verify if a given FD X → Y is satisfied by a relation R on a relation scheme R we find any two tuples with the same X value; if the FD X → Y is satisfied in R then the Y values in these tuples must be the same. We repeat this procedure until all pairs of tuples with the same X value are examined. Another approach involves ordering the tuples of R on the X values so that all tuples with the same X values are together. Then it is easy to verify if the corresponding Y values are also same and verify if R satisfies the FD X → Y.

Example 1:- consider following relation 'schedule' given in fig.

Prof	Course	Room	Max_enrollment	Day	Time
Smith	353	A532	40	Mon	11 : 45
Smith	353	A532	40	Wed	11 ; 45
Clark	355	H940	300	Tue	1 : 45
Clark	355	H940	300	Thur	2 : 10
Turner	456	B278	45	Mon	3 : 15
Turner	456	B278	45	Wed	4 : 50

Fig. 2.19 The schedule relation

For the above relation, the FD course → Prof is satisfied. However, the FD prof → Course is not satisfied, as one professor can teach to more than one course.

Closure of a set of functional Dependencies:-

- The set of functional dependencies that is logically implied by ‘F’ is called the closure of F & is written as F+

  Definition

  If F is a set of FDs on a relation scheme R, then F^+, the closure of F, is the smallest set of FDs such that F^+ F and no FD can be derived from F by using the inference axioms that are not contained in F^+.

- Suppose we are given a relation schema R = (A,B,C,G,H,I) & the set of functional dependencies:

  $A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H$

  The functional dependency

  $A \rightarrow H$ is logically implied.

  Proof:

  $A \rightarrow B$ and $B \rightarrow H$ $A \rightarrow H$

- Given F, we can compute F directly from the formal definition of functional dependency, If ‘F’ is large, this process would be lengthy, Axioms (or) rules of inference, provide a simpler technical for reasoning about functional dependencies.

Armstrong’s Axioms :-

- We can use following three rules to find logically implied functional dependencies. This collection of rules is called Armstrong’s Axioms in honor of the person who first proposed it.

1. Reflexivity Rule : If ‘α’ is set of attributes & $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.

2. Augmentation Rule : If α-> β holds and y is a set of attributes, then yα ->yβ holds.

3. Transitivity Rule: If α-> β holds and β->y holds, then α->y holds.

Armstrongs axioms are sound, because they do not generate any incorrect functional dependencies.They are complete, because for a given set 'F' of functional dependencies they allow us to generate all F^+.

Additional Rules:

Although Armstrong's axioms are complete, it is tiresome to use them directly for the computation of F^+. To simplify further, we list additional rules.

1. Union Rule: If α-> β holds and α->y holds, then α-> βy holds.

2. Decomposition Rule: If α-> βy holds, then α-> β holds and α->y holds.

3. Psuedo transitivity rule: If α-> β holds and x β->δ holds, then αy-> δ holds.

Consider schema R= (A,B,C,G,H,I), and Set of FDS (A->B,A->C, CG->H,CG->I,B->H)
Some of members of F^+ are:

- A->H : By Transitivity rule

  ➔ A->B andB->H therefore B->H

- CG->HI

  CG->H and CG->I
- Therefore by union rule

  CG->HI
- AG->I:

  A->C and CG->I
- By Psudotransistivity Rule

  AG->I
  OR
  A->C
- By Augmentation Rule

  AG->CG and CG->I
- By Transitivity Rule

  AG->I

Canonical cover

A canonical cover F_C for F is a set of dependencies such that 'F' logically implies all dependencies in F_C and F_C logically all in F. F_C must have the following properties

1. No functional dependency in F_c contains an extraneous attribute.
2. Each Left side of functional dependency in F_c is unique. That is no two dependencies $\alpha 1\ \alpha \rightarrow \beta \rightarrow \beta 1$ & $\alpha 2 \rightarrow \beta 2$ in F_c such that $\alpha 1 = \alpha 2$.

Extraneous Attributes

An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies. The formal definition of extraneous attributes is as follows:

Consider a set F of functional dependencies & the functional dependency $\alpha \rightarrow \beta$ in F

1. Attribute 'A' is extraneous in α if Aeα, & F Logically implies

   (F- {α →β}) U {(α-A)->β}
2. Attribute 'A' is extraneous in β if AeB & the set of functional dependencies

   (F-- {α →β}) U {(α ->) B-A} logically implies F.

For ex. Suppose we have FD_s AB->C & A→C then A→ C in F. then B is extraneous in AB->C.

Another example is, suppose the set F contains FDs AB → CD and A → C. Then C would be extraneous in AB → CD.

Canonical cover for a set of functional dependencies F can be constructed as follows:

$F_c = F$

Repeat

Use the union rule to replace any dependencies in F_c of the form $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_1$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

find a FD $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β.

If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

Until F_c does not change.

Example 2: consider the following set F of functional dependencies on schema (A,B,C).

A → BC

B → C

A → B

AB → C.

The canonical cover for F is computed as follows:

- There are two FDs with the same set of attributes on the left side of the arrow:

  A → BC

  A → B

  Combine these FDs into A → BC
- A is extraneous attribute in AB → C because F logically implies (F – {AB → C}) U {B → C). This assertion is true because B → C is already in our set of FDs.
- C is extraneous in A → BC because F logically implies (F – {A → BC}) U {A → B}. This assertion is true because A → B is already in our set of FDs.

  Thus, our canonical cover is,

  A → B

  B → C.

NORMALIZATION

- It's an essential part of database design
- A good understanding of the semantics of data helps the designer to build efficient design wins the concept of normalization.

Purpose of normalization

1. Minimize redundancy in data
2. Remove insert, delete & update anamoly during database activities
3. Reduce the need to reorganize data when its modified or enhanced.
4. Normalization reduces a complex user view to a set of small & stable subgroups of fields/relations this process helps to design a logical data model known as conceptual data model.

Normalization forms

Different normalization forms are:

1. First normal form (1NF):

   A relation is said to be in the first normal form it it's already in normalized form & it has no repeating group.

2. Second normal form (2NF):

   A relation is said to be in the second normal form it it's already in first normal form & it has no partial dependency.

3. Third normal form (3NF):

   A relation is said to be in the third normal form it it's already in second normal form & it has no transitive dependency.

4. Boyce-codd normal form (BCNF):

   A relation is said to be in BCNF if it's already in the third normal form & every determinant is a candidate key it's a stronger version of third normal form.

5. Forth normal form (4NF):

   A relation is said to be in the forth normal form it it's already in BCNF & it has no multivalued & dependency.

6. Fifth normal form (5NF):

   A relation is said to be in fifth normal form if it's already in 3NF form and it has no join dependency.

Different Terminologies:-

The different terminologies used in various normal forms are explained below:

1. Partial dependency :-

❖ In a relation having more than one key field , a sunset of non-key fields may depend on all the key fields but another sunset / a particular non – key field may depend on only one of the key fields (i.e. may not depend on all the key fields). Such dependency is called partial dependency.

2. Tramitive dependency

❖ In a relation, there may be dependency among non- key fields. Such dependency is called as transitive dependency.

3. Determinant :-

❖ A determinant is any field (simple field (or) composite field) on which some other fi3eld is fully functionally dependent.

4. Multivalued dependency :-

❖ Consider X,Y & Z in relation. If for each value of X, there is well defined set of values of Y & set of values of Z & the set of values of Y is independent of the set of values of Z, then multivalue d dependency exists.

i.e. $X \rightarrow\rightarrow Y/Z$

5. Join dependency :-

❖ A relation which has a join dependency cannot be decomposed by projection into other relation without any difficulty & undesirable results.

First normal form:

2) customer (Cust-No, Cust-name, Cust-Add)

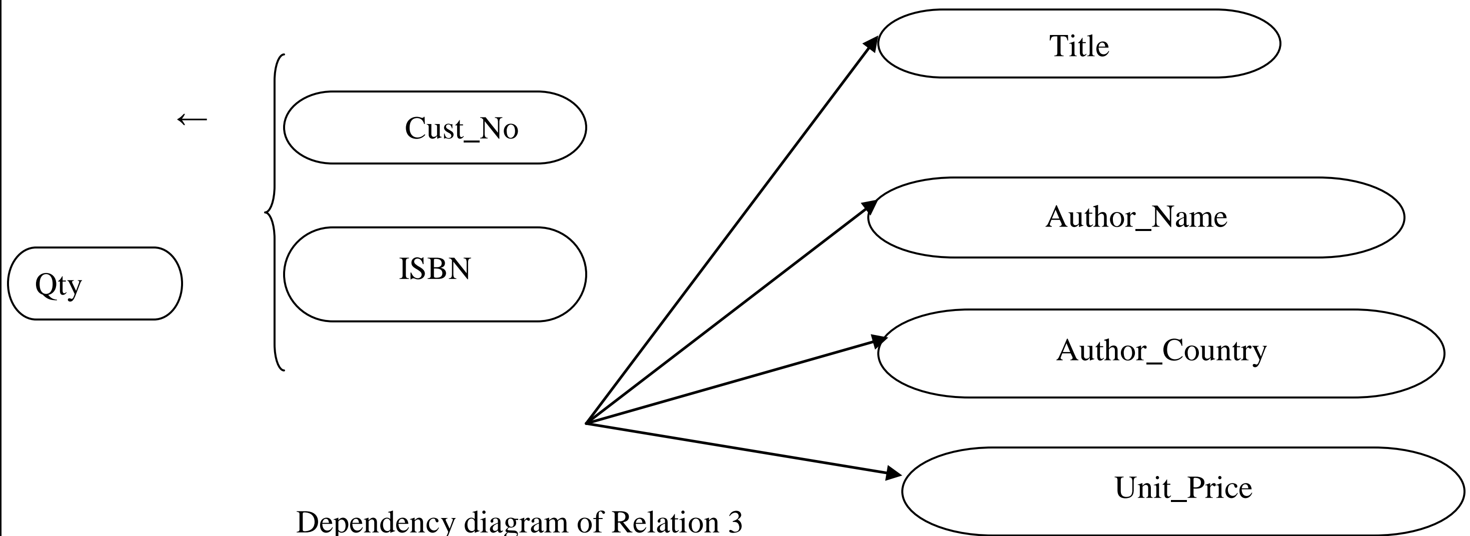
3) customer-Book (cust-No, ISBN, Title, Author - Name, Author-country, Qty, unit-price)

Now each of the above relations (i.e. relation 2 & relation 3 is in 1NF).

Second normal form

- Second normal form removes partial dependency among attributes of relation.
- In relation 2, the normal form of key fields is only one & hence there is no scope for partial dependency.
- The absence of partial dependency in relation 2 takes it into second normal form without any modification.
- In relation 3, the normal form of key fields are two. The dependency diagram or Relation 3 is shown below:

Dependency diagram of Relation 3



Dependency diagram of Relation 3

- Qty depends on Cust - No & ISBN, but the remaining non-key fields (Title, Author- country, unit-price) depend only on ISBN. Thus, there exists partial dependency.
- The existence of Partial dependency will result into

1. Insertion
2. Update
3. Deletion Anomaly

1. **Insertion Anomaly :-**

   - In Relation 3, if we want to insert data of a new book (ISBN, Title, Author – Name, Author – Country, Unit – Price) there must be atleast one customer. This means that the data of a new book can be entered into Relation 3 only when the first customer buys the book.

2. **Update Anomaly :-**

   - In relation 3, If we want to change any of the non-key fields like Title, Authore – name, it will result into inconsistency because the same is available in more than one record.

3. **Deletion Anomaly :-**

   - In Relation 3, if book is purchased by only one customer, then the book data will be lost when we delete that record after fully satisfying that customer order.

Hence, Relation 3 in divided into 2 Relations:-

4) Salas (Cust - No , ISBN , Qty)

5) Book - Auther (ISBN ,Title, Author-name, Author - country, unit - Price)

Relation 4&5 are now in second normal form.

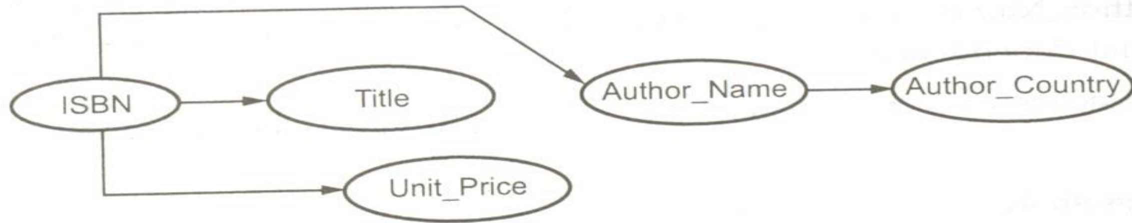
Third Normal Form (3NF)

Third normal form removes transitive dependency among attributes of relation.

In relation 4, there is only one nonkey field. So, there is no question of dependency between non – key fields. Thus, there is no transitive dependency. Hence Relation 4 is in 3NF.

In relation 2, there is no dependency between the non – key fields. This means that it has no transitive dependency. Hence, Relation 2 is also in 3NF.

In relation 5, author's country depends on author name. This means that Relation 5 has transitive dependency. The dependency diagram is shown in fig



The existence of transitive dependency will result into insert, update and delete anomaly:

Insertion anomaly:

Consider the book company has resident authors who are n the process of developing new books, it will be difficult to include the author's details in Relation 5. This means that there should be at least one published book to insert the details of a resident author.

Update anomaly:

If author's country is to be modified, then it is necessary to modify number of tuples as the same data is in number of tuples.

Deletion anomaly:

If the only one book of a resident author is not reprinted, then the respective author's data will be lost. Hence, to over come all these anomalies, Relation 5 is subdivided into two relations:

6) Book (ISBN, Title, Unit_Price, Author_Name)

7) Author (Author_Name, Author _ country)

In Relation 6, Author_Name is underlined with dotted line to indicate that it is a foreign key.

Boyce – Codd Normal Form (BCNF)

BCNF is more rigorous form of 3NF. It deals with relational tables which has multiple candidates keys, composite candidate keys, and candidate keys that overlapped. BCNF is based on the concept of determinant. A determinant is a column on which some of the columns are fully functional dependent. A relational table is in BCNF if every determinant is candidate key.

Case 1: Multiple candidate keys

Consider following relational table

A#	Aname	Aqualification	Astatus	TitleID	Royalty
100	Arora	Ph.D	10	T1	3000
110	Sharma	M.Tech	20	T2	4000

A relational table

Suppose Author's status (Astatus) depends on his / her qualification. If author is Ph.D his status (Astatus) is 10, if M.Tech it is 20 and so on. This relation has three determinants: A#, TitleID, Aqualification. But only (A#, TitleID) combination is a candidate key; so this relation is not in BCNF. For a relation to be in BCNF each determinant must be a candidate key.

Case2: Composite Candidate Keys

Suppose there are three relations as given below:

A#	Title ID	Royalty
A1	T1	5000
A2	T2	7000

Fig:- Author – title table

A#	Title ID	Royalty
A1	John	Ph.D
A2	Tom	M.Tech

Fig:- Author Table

Aqualification	Astatus
Ph.D	10
M.Tech	20
B.Tech	30
Others	40

Fig:- Author – qualification – status table

- Author – title table has one determinant (A#, TitleID). It is also a candidate key so the relation is in BCNF.
- Author table has one determinant A# and it is also a candidate key so Author is in BCNF.

- Similarly Author – qualification – status is also in BCNF.

Case 3: Candidate keys that overlapped

Consider following relation

A#	Aname	Title ID	Royalty
A1	John	T1	3000
A2	Tom	T2	4000

Fig: Author Table

Suppose that each Author's name is unique. Then in above relation, the keys are: (A#, Title ID), and (AName, TitleID). These are two determines A# and Aname. The attributes of each possible relation which we can make out of the original relation (A#, Aname, TitleID, Royalty) must depend on these two determines (A#, Aname). But the attributes of relation (A#, Aname) does not depend on the determinant since each A# and Aname is independent of the other. Hence the relation Author is not in BCNF. But this relation is in Third Normal Form. So a relation which is in Third Normal Form need not be in BCNF, but the converse is true. So BCNF is stronger than the 3NF.

Multivalued Dependencies and Fourth Normal Form

Formal Definition of Multivalued Dependency

Formally, a multivalued dependency (MVD) X → Y specified on relation schema R, where X and Y are both subsets of R, Specifies the following constraint on any relation state r of R: If two tuples t_1 and t_2 exist in r such that $t_2[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist r with the following properties.

- $T_3[X] = t_4[X] = t_1[X] = t_2[X]$
- $T_3[Y] = t_1[Y] = t_4[Y] = t_2[Y]$.
- $T_3[Z] = t_2[Z] = t_4[Z] = t_1[Y]$.

EMP

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

EMP_PROJECTS

ENAME	PNAME
Smith	X
Smith	Y

EMP_DEPENDENTS

ENAME	DNAME
Smith	John
Smith	Anna

SUPPLY

SNAME	PARTNAME	PROJNAME
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

R1

SNAME	PARTNAME
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R2

SNAME	PROJNAME
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R3

PARTNAME	PROJNAME
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Fig 15.4 Fourth and fifth normal form. (a) The EMP relation with two MVDs: ENAME → PNAME AND ENAME → DNAME. (b) Decomposing EMP into two relation in 4NF. (c) The relation SUPPLY with no MVDs satisfies 4NF but does not satisfy 5NF if the JD(R1, R2, R3) holds. (d) Decomposing SUPPLY into three NF relations.

Whenever X → Y holds, we say that X multidetermines Y. Because of the symmetry in the definition, whenever X → Y holds in R, so does X → Y. Hence, X → Y implies X → Z, and therefore it is sometimes written as Z → Y | Z.

The formal definition specifies that, given a particular value of X, the set of values of Y determined by this value of X is completely determined by X alone and does not depend on the values of the remaining attributes Z of R. Hence, Whenever two tuples

exist that have distinct values of Y but the same value of X, these vales of Y must be repeated in separate tuples with every distinct value of Z that occurs with that same value of X. This informally corresponds to Y being a multivalued attribute of the entities represented by tuples in R.

In fig 15.4(a) the MVDs ENAME → PNAME and ENAME → DNAME (or ENAME → PNAME | DNAME) hold in the EMP relation. The employee with ENAME ‘smith’ works on projects with PNAME ‘X’ and ‘Y’ and has two dependents with DNAME ‘John’ > and <’smith’ , ‘Y’, ‘Anna’>), we would incorrectly show associations between project ‘X’ and ‘John’ and between project ‘Y’ and ‘Anna’; these should not other two tuples (<’smith’ , ‘X’, ‘Anna’ > and <’smith’, ‘Y’ , ‘John’>) to show that {‘X’, ;Y’} and {‘John’ , ‘Anna’} are associated only with ‘smith’; that is, there is no association between PNAME and DNAME which means that the two attributes are independent.

An MVD X → Y in R is called a trivial MVD if (a) Y is a subset of X, or (b) X U Y = R. For example, the relation EMP_PROJECTS in Fig 15.4 (b) has the trivial MVD ENAME → PNAME. An MVD that satisfies neither (a) nor (b) is called a nontrivial MVD. A trivial MVD will hold in any relation state r of R; it is called trivial because it does not specify any significant or meaningful constraint on R.

Inference Rules for Functional and Multivalued Dependencies:

Assume that all attributes are included in a “universal” relation schema R = {A, A2,…..An} and that X, Y, Z and W are subsets of R.

IR1 (reflexive rule for FDs) : If X   Y, then X → Y.

IR2 (augmentation rule for FDs): {X → Y} |= XZ → YZ.

IR3 (transitive rule for FDs) : {X → Y, Y → Z} |= X → Z.

IR4 (complementation rule for MVDs) : {X→ Y} |= {X → (R – (X U Y))}.

IR5 (augmentation rule for MVDs): If X → Y and W   Z then WX → YZ.

IR6 (transitive rule for MVDs): {X → Y, Y → Z} |= X → (Z – Y).

IR7 (replication rule for FD to MVD): {X → Y} |= X → Y.

IR8 (Coalescence rule for FDs and MVDS): If X → Y and there exists W with the properties that (a)

W   Y is empty, (b) W --> Z, and (c) Y   Z, then X → Z.

IR1 through IR3 are Armstrong's inference rules for FDs alone. IR4 through IR6 are inference rules pertaining to MVDs only. IR7 and IR8 relate FDs and MVDs.

Fourth Normal Form

A relation schema R is in 4NF with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every nontrivial multivalued dependency X → Y in F^+, X is superkey for R.

The EMP relation of fig 15.4 is not in 4NF because in the nontrivial MVDs ENAME → PNAME and ENAME → DNAME , ENAME is not a superkey of EMP. We decompose EMP into EMP _ PROJECTS and EMP _ DEPENDENTS , shown in fig 15.4 (b). Both EMP_PROJECTS and EMP_DEPENDENTS are in 4NF, because the MVDs. No other nontrivial MVDs hold in either EMP_PROJECTS or EMP_DEPENDNETS.

The EMP relation with an additional employee, 'Brown', who has three dependents ('Jim', 'Joan', and 'Bob') and works on four different projects ('W','X','Y', and'Z'). There are 16 tuples in EMP in fig 15.5(a). If we decompose EMP into EMP_PROJECTS and EMP_DEPENDENTS, as shown in fig 15.5 (b), we need to store a total of only 11 tuples in both relations. Not only would the decomposition save on storage, but also the update anomalies associated with multivalued dependencies are avoided. For example, if Brown starts working on another project, we must insert three tuples in EMP ------- one for each dependent. If we forget to insert any one of those , the relation violates the MVD and becomes inconsistent in that it incorrectly implies a relationship between project and dependent.

(a) **EMP**

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John
Brown	W	Jim
Brown	X	Jim
Brown	Y	Jim
Brown	Z	Jim
Brown	W	Joan
Brown	X	Joan
Brown	Y	Joan
Brown	Z	Joan
Brown	W	Bob
Brown	X	Bob
Brown	Y	Bob
Brown	Z	Bob

(b) **EMP_PROJECTS**

ENAME	PNAME
Smith	X
Smith	Y
Brown	W
Brown	X
Brown	Y
Brown	Z

EMP_DEPENDENTS

ENAME	DNAME
Smith	Anna
Smith	John
Brown	Jim
Brown	Joan
Brown	Bob

The EMP relation in Fig is not in $NF, because it represents two independent 1:N relationships ---- one between employees and the projects they work on and the other between employees and their dependents. We sometimes have a relationship between three entities that depends on all three participation entities, such as the SUPPLY relation shown in Fig 15.4(c). (Consider only the tuples in Fig 15.4 (c) above the dotted line for now). In this case a tuple represents a supplier supplying a specific part to a particular project, so there are no nontrivial MVDs. The SUPPLY relation is already in 4NF and should not be decomposed. Notice that relations containing nontrivial MVDs tend to be all key relations – that is, their key is all their attributes taken together.

Join Dependencies and Fifth Normal Form

A Join dependency (JD), denoted by JD(R_1, R_2, …., R_n), specified on relation schema R, specifies a constraint on the states r of R. The constraint states that every legal state r of R should have a lossless join decomposition into R1, R2, …… , Rni that is, for every such r we have

- $(\pi R_1 (r) , \pi R_2(r), \ldots\ldots \pi R_n (r)) = r$

Notice that an MVD is a special case of a JD where n = 2. That is, a JD denoted as JD (R_1, R_2) implies an MVD (R_1 R_2) → (R_1 - R_2) (or by symmetry, (R_1 R_2) → (R_2 - R_1)). A Join dependency JD (R_1, R_2, …….R_n), specified on relation schema R, is a trivial JD if one of the relation sehemes Ri in JD (R_1, R_2, …….R_n) is equal to R. Such a

dependency is called trivial because it has the lossless join property for any relation state r of R and hence does not specify any constraint on R. We can now define fifth normal form, which is also called project – join normal form. A relation schema R is in fifth normal form (5NF) (or project – join normal form (PJNF)) with respect to a set F of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD (R_1, R_2, …….R_n) in F_+ (that is, implied by F), every R_i is a superkey of R.

Question Bank

2 Mark Questions:

1. What is normalization?
2. What is Functional Dependency?
3. When is a functional dependency F said to be minimal?
4. What is Multivalued dependency?
5. What is Lossless join property?
6. What is 1 NF (Normal Form)?
7. What is Fully Functional dependency?
8. What is 2NF?
9. What is 3NF?
10. What is BCNF (Boyce-Codd Normal Form)?
11. What is 4NF?
12. What is 5NF?
13. What is Domain-Key Normal Form?
14. What is join dependency and inclusion dependency?

16 Mark Questions

1. What is normalization? Explain 1NF,2NF with example.
2. Explain 3NF and BCNF with example.
3. What is FD? Explain the role of FD in the process of normalization.
4. Compare BCNF and 3NF.
5. Explain Canonical cover with example.

UNIT –IV TRANSACTIONS

Transaction:

Collections of operations that form a single logical unit of work are called transactions.

Transaction concept:

A transaction is a unit of program execution that accesses and possibly updates various data items. A transaction is initiated by a user program written in a high level data manipulation language or programming language, where it is delimited by statements of the form begin transaction and end transaction. The transaction consist of all the operations executed between the begin transaction and the end transaction.

To ensure integrity of data, the database system should maintain the following properties of the transactions.

Atomicity:

Either all operations of the transactions are reflected properly in the database or none are.

Consistency:

Execution of a transaction in isolation preserves the consistency of the database.

Isolation:

Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions Ti and Tj , it appears to Ti that either Tj finished execution before Ti started, or Tj started execution after Ti finished. Thus each transaction is unaware of other transactions executing concurrently in the system.

Durability:

After a transaction completes successfully, the changes it has made to the database persit, even if there are system failures.

These properties are often called the ACID properties.

Consider banking system consisting of several accounts and a set of transactions that access and update those accounts. Transactions access data using two operations.

- Read(X) → transfers the data item X from the database to a local buffer belonging to the transaction that executed the read operation.
- Write(X) →transfers the data item X from the local buffer of the transaction that executed the write back to the database.

Let Ti be the transaction that transfers $50 from account A to B. this transaction can be defined as

```
Ti  :    read(A)
         A := A - 50
         write(A)
         read(B)
         B: = B + 50
         write(B)
```

Consistency:

The consistency requirement here is that the sum of A and B be unchanged by the execution of the transaction. If the database is consistent before an execution of the transaction, the database remains consistent after the execution of the transaction.

Atomicity:

Suppose that just before the execution of transaction Ti the values of accounts A and B are $ 1000 $ 2000, resp. Now suppose that during the execution f the transaction Ti, a failure occurs after the write(A) operation but before Write(B) operation, that prevents Ti from completing its execution successfully(power failures, hardware failures, and s/w errors), in this case the values of accounts A and B reflected in the database are $950 and $2000. The system destroyed $50 as a result of this failure. The sum A+B is no longer preserved. Such a state is known as inconsistent state.

If the atomicity property is present, all actions of the transaction are reflected in the database or none are. The basic idea behind ensuring atomicity is, the database system keeps track of the old values of any data on which a transaction performs a write, and if the transaction does not complete its execution, the database system restores the old values to make it appear as though the transaction never executed.

Ensuring atomicity is the responsibility of the database system itself. It is handled by a component called the transaction-management component.

Durability:

The durability property guarantees that, once a transaction completes successfully all updates that it carried out on the database persist, even if there is a system failure after the transaction completes execution. Durability is guaranteed by ensuring either

1. The updates carried out by the transaction have been written to disk before the transaction completes.
2. Information about the updates carried out by the transaction and written to disk is sufficient to enable the database to reconstruct the updates when the database system is restarted after failure.

Ensuring durability is the responsibility of a component of the database system called the recovery-management component.

Isolation:

Even if the consistency and atomicity properties are ensured for each transaction, if several transactions are executed concurrently, their operations may interleave in some undesirable way, resulting in an inconsistent state. A way to avoid the problem of concurrently executing transactions is to execute transactions serially, that is one after the other.

The isolation property of a transaction ensures that the concurrent execution of transaction results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in some order.

Ensuring the isolation property is the responsibility of a component of the database system called the concurrency-control component.

Transaction state:

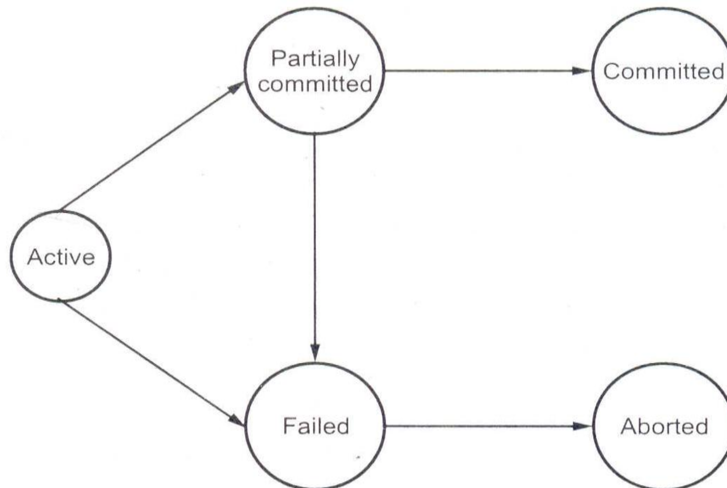
A transaction must be one of the following states.

- Active → the initial state; the transaction stays in this state while it is executing.
- Partially committed → after the initial state has been executed.
- Failed → after the discovery, that 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.

Once the changes caused by an aborted transaction have been undone, it means the transaction has been rolled back. A transaction that completes its execution successfully is said to be committed. A transaction is said to have terminated if has either committed or aborted.

A transaction enters the failed state after the system determines that the transaction can no longer proceed with its normal execution. Such a transaction much be roll backed. At this point the system has two options:

- It can restart the transaction, but only if the transaction was aborted as a result of some hardware or software error that was not created thro the internal logic of the transaction. A restarted transaction is considered as a new transaction.
- It can kill the transaction. It usually does so because of some internal logic error that can be corrected only by rewriting the application program, or because the input was bad, or becoz the desired data were not found in the database.



Concurrent Executions:

Transaction processing system usually allows multiple transactions to run concurrently which causes inconsistency of data. To overcome this problem the transactions where allowed to execute serially. However the two good reasons for allowing concurrency is

- Improved throughput and resource utilization:

The number of transactions executed in a given amount of time is known as throughput. A transaction consists of many steps. Some involve I/O activity and some involve CPU activity. The CPU and I/O activity can operate in parallel. Therefore I/O activity can be done in parallel with processing at the CPU. This leads to multiple transactions to be executed in parallel. Hence the throughput increases. The processor and disk utilization also increase.

- Reduced waiting time:

The transactions running on a system can be short as well as long. If transactions run serially, a short transaction can wait for a long transaction to complete that leads to delay in running a transaction. A concurrent execution reduces delay in running transactions. It also reduces the average response time. (i.e) the average time for a transaction to be completed after it has been submitted.

Let T1 and T2 be two transactions that transfer funds from one account to another. Ti transfers $50 from account A to account B.

```
T1 :    read(A);
        A := A - 50;
        write(A);
        read(B);
        B := B + 50;
        write(B);
```

T2 transfers 10 percent of the balance from account A to account B.

```
T2 :    read(A);
        temp := A * 0.1;
        A := A - temp;
        write(A);
        read(B);
        B := B + temp;
        write(B);
```

Let the current values of accounts A and B be $1000 and $2000. if suppose the transactions are executed serially .

T1	T2
read(A)	
A := A - 50	
write(A)	
read(B)	
B := B + 50	
write(B)	
	read(A)
	temp := A * 0.1
	A := A - temp
	write(A)
	read(B)
	B := B + temp
	write(B)

If T1 is executed first and then T2 is executed, then the final values of accounts A and B after the execution is $855 and $ 2145. The sum A+B is preserved after the execution of both transactions.

Similarly if the transactions are executed one at a time in the order T2 followed by T1, then the account values of A and B will be $850 and $ 2150.

T2

```
read(A)
temp := A * 0.1
A := A - temp
write(A)
read(B)
B := B + temp
write(B)
```

T1

```
read(A)
A := A - 50
write(A)
read(B)
B := B + 50
write(B)
```

The execution sequences are called schedules. They represent the chronological order in which instructions are executed in the system. A schedule for a set of transactions must consist of all instructions of those transactions, and must preserve the order in which the instructions appear in those transactions.

Suppose that the two transactions are executed concurrently in the following way,

T1	T2
read(A)	
A := A - 50	
write(A)	
	read(A)
	temp := A * 0.1
	A := A - temp
	write(A)
read(B)	
B := B + 50	
write(B)	
	read(B)
	B := B + temp
	write(B)

In the above transaction, after the execution takes place, the state will be same as the serial transaction T1 followed by T2. and also A+B is preserved.

T1	T2
read(A)	
A := A - 50	
	read(A)
	temp := A * 0.1
	A := A - temp
	write(A)
	read(B)
write(A)	
read(B)	
B := B + 50	
write(B)	
	B := B + temp
	write(B)

The above transaction is also executed concurrently. But the final values of accounts A and B are $950 and $2100. Hence the final state is inconsistent and A+B is not preserved. "Not all concurrent executions result in correct state."

System Recovery:

- Modifying the database without ensuring that the transaction will commit may leave the database in an inconsistent state.
- Consider transaction T i that transfers $50 from account A to account B ; goal is either to perform all database modifications made by T i or none at all.
- Several output operations may be required for T i (to output A and B). A failure may occur after one of these modifications have been made but before all of them are made.
- To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself.
- We study two approaches:
  - log-based recovery , and
  - shadow-paging
- We assume (initially) that transactions run serially, that is, one after the other.

Complete Media Recovery

Media recovery commands

There are three basic media recovery commands, which differ only in the way the set of files being recovered is determined. They all use the same criteria for determining if files can be recovered. Media recovery signals an error if it cannot get the lock for a file it is attempting to recover. This prevents two recovery sessions from recovering the same file. It also prevents media recovery of a file that is in use. You should be familiar with all media recovery commands before performing media recovery.

RECOVER DATABASE

RECOVER DATABASE performs media recovery on all online datafiles that require redo to be applied. If all instances were cleanly shutdown, and no backups were restored, RECOVER DATABASE indicates a no recovery required error. It also fails if any instances have the database open (since they have the datafile locks). To perform media recovery on an entire database (all tablespaces), the database must be mounted EXCLUSIVE and closed.

RECOVER TABLESPACE

RECOVER TABLESPACE performs media recovery on all datafiles in the tablespaces listed. To translate the tablespace names into datafile names, the database must be mounted and open. The tablespaces must be offline to perform the recovery. An error is indicated if none of the files require recovery.

RECOVER DATAFILE

RECOVER DATAFILE lists the datafiles to be recovered. The database can be open or closed, provided the media recovery locks can be acquired. If the database is open in any instance, then datafile recovery can only recover off-line files.

MEDIA RECOVERY TYPES

Complete Media Recovery can be classified into three categories:

Performing Closed Database Recovery

If the database is open, shut it down using the Server Manager Shutdown Abort mode of the Shutdown Database dialog box, or the SHUTDOWN command with the ABORT option.

If files are permanently damaged, restore the most recent backup files (taken as part of a full or partial backup) of only the datafiles damaged by the media failure. Do not restore any undamaged datafiles or any online redo log files. If the hardware problem has been repaired, and damaged datafiles can be restored to their original locations, do so, and skip Step 6 of this procedure. If the hardware problem persists, restore the datafiles to an alternative storage device of the database server and continue with this procedure.

Start Server Manager and connect to Oracle with administrator privileges.

Start a new instance and mount, but do not open, the database using either the Server Manager Startup Database dialog box (with the Startup Mount radio button selected), or the STARTUP command with the MOUNT option.

If one or more damaged datafiles were restored to alternative locations in Step 3, the new location of these files must be indicated to the control file of the associated database.

All datafiles you want to recover must be online during complete media recovery. To get the datafile names, check the list of datafiles that normally accompanies the current control file, or query the V$DATAFILE view. Then, issue the ALTER DATABASE command with the DATAFILE ONLINE option to ensure that all datafiles of the database are online.

To start closed database recovery of all damaged datafiles in one step, use either the Server Manager Apply Recovery Archive dialog box, or an equivalent RECOVER DATABASE statement.

To start closed database recovery of an individual damaged datafile, use the RECOVER DATAFILE statement in Server Manager.

Now Oracle begins the roll forward phase of media recovery by applying the necessary redo log files (archived and online) to reconstruct the restored datafiles. Unless the application of files is automated, Oracle prompts you for each required redo log file.

Oracle continues until all required archived redo log files have been applied to the restored datafiles. The online redo log files are then automatically applied to the restored datafiles and notifies you when media recovery is complete. If no archived redo log files

are required for complete media recovery, Oracle does not prompt for any. Instead, all necessary online redo log files are applied, and media recovery is complete.

After performing closed database recovery, the database is recovered up to the moment that media failure occurred. You can then open the database using the SQL command ALTER DATABASE with the OPEN option.

Performing Open-Database, Offline-Tablespace Recovery

At this point, an open database has experienced a media failure, and the database remains open while the undamaged datafiles remain online and available for use. The damaged datafiles are automatically taken off-line by Oracle.

The starting point for this recovery operation can vary, depending on whether you left the database open after the media failure occurred.

If the database was shut down, start a new instance, and mount and open the database. Perform this operation using the Server Manager Startup Database dialog box (with the Startup Open radio button selected), or with the STARTUP command with the OPEN option. After the database is open, take all tablespaces that contain damaged datafiles offline.

If the database is still open and only damaged datafiles of the database are offline, take all tablespaces containing damaged datafiles offline. Oracle identifies damaged datafiles via error messages. Tablespaces can be taken offline using either the Take Offline menu item of Server Manager, or the SQL command ALTER TABLESPACE with the OFFLINE option. If possible, take the damaged tablespaces offline with temporary priority (to minimize the amount of recovery).

Correct the hardware problem that caused the media failure. If the hardware problem cannot be repaired quickly, you can proceed with database recovery by restoring damaged files to an alternative storage device.

If files are permanently damaged, restore the most recent backup files (taken as part of a full or partial backup) of only the datafiles damaged by the media failure. Do not restore undamaged datafiles, online redo log files, or control files. If the hardware problem has been repaired and the datafiles can be restored to their original locations, do so. If the hardware problem persists, restore the datafiles to an alternative storage device of the database server.

If one or more damaged datafiles were restored to alternative locations (Step 3), indicate the new locations of these files to the control file of the associated database.

After connecting with administrator privileges, use the RECOVER TABLESPACE statement in Server Manager to start offline tablespace recovery of all damaged datafiles in one or more offline tablespaces using one step.

Oracle begins the roll forward phase of media recovery by applying the necessary redo log files (archived and online) to reconstruct the restored datafiles. Unless the applying of files is automated, Oracle prompts for each required redo log file.

Oracle continues until all required archived redo log files have been applied to the restored datafiles. The online redo log files are then automatically applied to the restored datafiles to complete media recovery. If no archived redo log files are required for complete media recovery, Oracle does not prompt for any. Instead, all necessary online redo log files are applied, and media recovery is complete.

The damaged tablespaces of the open database are now recovered up to the moment that media failure occurred. You can bring the offline tablespaces online using the Place Online menu item of Server Manager, or the SQL command ALTER TABLESPACE with the ONLINE option.

Performing Open-Database, Offline-Tablespace Individual Recovery

Identical to the preceding operation, here an open database has experienced a media failure, and remains open while the undamaged datafiles remain online and available for use. The damaged datafiles are automatically taken offline by Oracle. The starting point for this recovery operation can vary, depending on whether you left the database open after the media failure occurred.

If the database was shut down, start a new instance, and mount and open the database. Perform this operation using the Server Manager Startup Database dialog box (with the Startup Open radio button selected), or with the STARTUP command with the OPEN option. After the database is open, take all tablespaces that contain damaged datafiles offline.

If the database is still open and only damaged datafiles of the database are offline, take all tablespaces containing damaged datafiles offline. Oracle identifies damaged datafiles via error messages. Tablespaces can be taken offline using either the Take Offline menu item of Server Manager, or the SQL command ALTER TABLESPACE with the OFFLINE option. If possible, take the damaged tablespaces offline with temporary priority (to minimize the amount of recovery).

Correct the hardware problem that caused the media failure. If the hardware problem cannot be repaired quickly, you can proceed with database recovery by restoring damaged files to an alternative storage device.

If files are permanently damaged, restore the most recent backup files (taken as part of a full or partial backup) of only the datafiles damaged by the media failure. Do not restore undamaged datafiles, online redo log files, or control files. If the hardware problem has been repaired and the datafiles can be restored to their original locations, do so. If the hardware problem persists, restore the datafiles to an alternative storage device of the database server.

If one or more damaged datafiles were restored to alternative locations (Step 3), indicate the new locations of these files to the control file of the associated database.

After connecting with administrator privileges, use the RECOVER DATAFILE statement in Server Manager to start offline tablespace recovery of all damaged datafiles in one or more offline tablespaces using one step.

Oracle begins the roll forward phase of media recovery by applying the necessary redo log files (archived and online) to reconstruct the restored datafiles. Unless the applying of files is automated, Oracle prompts for each required redo log file.

Oracle continues until all required archived redo log files have been applied to the restored datafiles. The online redo log files are then automatically applied to the restored datafiles to complete media recovery. If no archived redo log files are required for complete media recovery, Oracle does not prompt for any. Instead, all necessary online redo log files are applied, and media recovery is complete.

The damaged tablespaces of the open database are now recovered up to the moment that media failure occurred. You can bring the offline tablespaces online using

the Place Online menu item of Server Manager, or the SQL command ALTER TABLESPACE with the ONLINE option.

Two phase commit

Consider a transaction T initiated at site S_i , where the transaction coordinator is C_i.

Two phase commit protocol

When T completes its execution that is, when all the sites at which T has executed inform C_i that T has completed C_i starts the two phase commit protocol.

- Phase 1:
  - C_i adds the record <prepare T> to the log, and forces the log onto stable storage.
  - It then sends a prepare T message to all sites at which T executed.
  - On receiving such a message, the transaction manager at the site determines whether it is willing to commit its portion of T.
  - If the answer is no, it addsa record <no T> to the log, and then responds by sending an abort T message to C_i.
  - If the answer is yes, it adds a record <ready T> to the log, and forces the log (with all the log records corresponding to T) onto stable storage.
  - The transaction manager then replies with a ready T message to C_i.
- Phase 2:
  - When Ci receives responses to the prepare T message from all the sites, or when a prespecified interval of time has elapsed since the prepare T message was sent out, Ci can determine whether the transaction T can be committed or aborted.
  - Transaction T can be committed if Ci received a ready T message from all the participating sites.
  - Otherwise, transaction T must be aborted.
  - Depending, on the verdict, either a record <commit T> or a record <abort T> is added to the log and the log is forced onto stable storage.
  - The coordinator sends either a commit T or an abort T message to all participating sites.
  - When a site receives that message, it records the message in the log.

A site at which T executed can unconditionally abort T at any time before it sends the message ready T to the coordinator.
Once the message is sent, the transaction is said to be in the ready state at the site.

Commit point of a Transaction

A transaction T reaches its commit point when all its operations that access the database have been executed successfullly and the effect all the transaction operations on the database have been recorded in the log.

ARIES Recovery Algorithm

ARIES

- ARIES is a state of the art recovery method
    - Incorporates numerous optimizations to reduce overheads during normal processing and to speed up recovery
    - The "advanced recovery algorithm" we studied earlier is modeled after ARIES, but greatly simplified by removing optimizations
- Unlike the advanced recovery algorithm, ARIES
    - Uses log sequence number (LSN) to identify log records
        - Stores LSNs in pages to identify what updates have already been applied to a database page
    - Physiological redo
    - Dirty page table to avoid unnecessary redos during recovery
    - Fuzzy checkpointing that only records information about dirty pages, and does not require dirty pages to be written out at checkpoint time
        - More coming up on each of the above …

ARIES Optimizations

- Physiological redo
    - Affected page is physically identified, action within page can be logical
        - Used to reduce logging overheads
            - e.g. when a record is deleted and all other records have to be moved to fill hole
                - Physiological redo can log just the record deletion
                - Physical redo would require logging of old and new values for much of the page
        - Requires page to be output to disk atomically
            - Easy to achieve with hardware RAID, also supported by some disk systems
            - Incomplete page output can be detected by checksum techniques,
                - But extra actions are required for recovery
                - Treated as a media failure

ARIES Data Structures

- Log sequence number (LSN) identifies each log record
  - Must be sequentially increasing
  - Typically an offset from beginning of log file to allow fast access
    - Easily extended to handle multiple log files
- Each page contains a PageLSN which is the LSN of the last log record whose effects are reflected on the page
  - To update a page:
    - X-latch the pag, and write the log record
    - Update the page
    - Record the LSN of the log record in PageLSN
    - Unlock page
  - Page flush to disk S-latches page
    - Thus page state on disk is operation consistent
      - Required to support physiological redo
  - PageLSN is used during recovery to prevent repeated redo
    - Thus ensuring idempotence
- Each log record contains LSN of previous log record of the same transaction
  - LSN in log record may be implicit
- Special redo-only log record called compensation log record (CLR) used to log actions taken during recovery that never need to be undone
  - Also serve the role of operation-abort log records used in advanced recovery algorithm
  - Have a field UndoNextLSN to note next (earlier) record to be undone
    - Records in between would have already been undone
    - Required to avoid repeated undo of already undone actions

LSN TransId PrevLSN RedoInfo UndoInfo LSN TransID UndoNextLSN RedoInfo

- DirtyPageTable
  - List of pages in the buffer that have been updated
  - Contains, for each such page
    - PageLSN of the page
    - RecLSN is an LSN such that log records before this LSN have already been applied to the page version on disk
      - Set to current end of log when a page is inserted into dirty page table (just before being updated)
      - Recorded in checkpoints, helps to minimize redo work
- Checkpoint log record
  - Contains:
    - DirtyPageTable and list of active transactions

        - For each active transaction, LastLSN, the LSN of the last log record written by the transaction
    - Fixed position on disk notes LSN of last completed checkpoint log record

ARIES Recovery Algorithm

- ARIES recovery involves three passes
- Analysis pass : Determines
    - Which transactions to undo
    - Which pages were dirty (disk version not up to date) at time of crash
    - RedoLSN : LSN from which redo should start
- Redo pass :
    - Repeats history, redoing all actions from RedoLSN
        - RecLSN and PageLSNs are used to avoid redoing actions already reflected on page
- Undo pass :
    - Rolls back all incomplete transactions
        - Transactions whose abort was complete earlier are not undone
            - Key idea: no need to undo these transactions: earlier undo actions were logged, and are redone as required

ARIES Recovery: Analysis

- Analysis pass
- Starts from last complete checkpoint log record
    - Reads in DirtyPageTable from log record
    - Sets RedoLSN = min of RecLSNs of all pages in DirtyPageTable
        - In case no pages are dirty, RedoLSN = checkpoint record's LSN
    - Sets undo-list = list of transactions in checkpoint log record
    - Reads LSN of last log record for each transaction in undo-list from checkpoint log record
- Scans forward from checkpoint
    - .. On next page …
- Scans forward from checkpoint
    - If any log record found for transaction not in undo-list, adds transaction to undo-list
    - Whenever an update log record is found
        - If page is not in DirtyPageTable, it is added with RecLSN set to LSN of the update log record
    - If transaction end log record found, delete transaction from undo-list
    - Keeps track of last log record for each transaction in undo-list

        - May be needed for later undo
- At end of analysis pass:
    - RedoLSN determines where to start redo pass
    - RecLSN for each page in DirtyPageTable used to minimize redo work
    - All transactions in undo-list need to be rolled back

ARIES Redo Pass

- Redo Pass: Repeats history by replaying every action not already reflected in the page on disk, as follows:
- Scans forward from RedoLSN. Whenever an update log record is found:
    - If the page is not in DirtyPageTable or the LSN of the log record is less than the RecLSN of the page in DirtyPageTable, then skip the log record
    - Otherwise fetch the page from disk. If the PageLSN of the page fetched from disk is less than the LSN of the log record, redo the log record
    - NOTE: if either test is negative the effects of the log record have already appeared on the page. First test avoids even fetching the page from disk!

ARIES Undo Actions

- When an undo is performed for an update log record
    - Generate a CLR containing the undo action performed (actions performed during undo are logged physicaly or physiologically).
        - CLR for record n noted as n ' in figure below
    - Set UndoNextLSN of the CLR to the PrevLSN value of the update log record
        - Arrows indicate UndoNextLSN value
- ARIES supports partial rollback
    - Used e.g. to handle deadlocks by rolling back just enough to release reqd. locks
    - Figure indicates forward actions after partial rollbacks
        - records 3 and 4 initially, later 5 and 6, then full rollback

ARIES: Undo Pass

- Undo pass
- Performs backward scan on log undoing all transaction in undo-list
    - Backward scan optimized by skipping unneeded log records as follows:
        - Next LSN to be undone for each transaction set to LSN of last log record for transaction found by analysis pass.
        - At each step pick largest of these LSNs to undo, skip back to it and undo it

- After undoing a log record
  - For ordinary log records, set next LSN to be undone for transaction to PrevLSN noted in the log record
  - For compensation log records (CLRs) set next LSN to be undo to UndoNextLSN noted in the log record
    - All intervening records are skipped since they would have been undo already
- Undos performed as described earlier

Other ARIES Features

- Recovery Independence
  - Pages can be recovered independently of others
    - E.g. if some disk pages fail they can be recovered from a backup while other pages are being used
- Savepoints:
  - Transactions can record savepoints and roll back to a savepoint
    - Useful for complex transactions
    - Also used to rollback just enough to release locks on deadlock
- Fine-grained locking:
  - Index concurrency algorithms that permit tuple level locking on indices can be used
    - These require logical undo, rather than physical undo, as in advanced recovery algorithm
- Recovery optimizations: For example:
  - Dirty page table can be used to prefetch pages during redo
  - Out of order redo is possible:
    - redo can be postponed on a page being fetched from disk, and performed when page is fetched.
    - Meanwhile other log records can continue to be processed

SQL SERVER DATABASE RECOVERY

SQL server database recovery is essential when system failure has occurred. If you take a look at businesses that experience data loss in a disaster, a whopping ninety percent go out of business within two years.

Data loss emergencies can be attributed to physical or logical problems. An outstanding seventy percent of breakdowns are due to physical failures that most often require the attention of a professional data recovery expert.

SQL Server Database

The SQL server database is a management system used to store data

on networked file servers. Companies depend on these databases to remain accessible in order for business to run smoothly.

Media in database servers can suffer from the same failure points as drives in PCs and workstations.

Symptoms

- Unable to boot
- Unable to access drives and partitions
- Unable to run or load data
- Corrupted data
- Virus attacks
- Hard drive failure or crashes
- Fire and water damage
- Media contamination or damage
- Accidental reformatting of partitions
- Accidental deletion of data

Backing Up

System failures occur even with the best configured systems. Therefore frequent backups are crucial to prevent data loss and can speed up the recovery process. SQL recovery can be successfully restored when supported by a well tested back up program. The different database recovery models make backing up more efficient when breakdowns occur.

System Maintenance Programs

When systems break down and backup devices become corrupt, access to the database is prevented. Downtime can set any company back therefore it is imperative that a professional data recovery specialist be consulted as soon as possible.

Usage of regular system maintenance programs in most cases will not be able to recover all of the data and can often create bigger problems.

SQL Server Database Recovery

SQL server database recovery is usually handled in the same fashion as a server / RAID recovery. A database can be successfully rebuilt by analyzing the contents of the drive images, tables and records containing data.

Concurrency

Need for concurrency control

Transaction processing system allows multiple transactions to run concurrently. Concurrent execution of multiple transactions causes several complications with consistency of the data. However, there are two good reasons for allowing concurrency:

(a) Improved throughput and resource utilization

Throughput is the number of transactions executed in a given amount of time.

A transaction consists of many steps. Some involve I.O activity; others involve CPU activity. The CPU and the disks in a computer system can operate in parallel. The parallelism of the CPU and the I/O system can therefore be exploited to run multiple transactions in parallel. If one transaction is reading or writing data on disk, another can be running in the CPU.

All of this increases throughput of the system correspondingly, the processor and disk utilization also increases. Thus the processor and disk spend less time idle, or not performing any useful work.

(b) Reduced waiting time

There may be a mix of transactions running on a system, some short and some long. If transactions run serially, a short transaction may have to wait for a proceding long transaction to complete, which can lead a unpredictable delays in running a transaction. If the transactions are operating on different parts of the database, it is better to run them concurrently sharing the CPU cycles and disk accesses among them.

Concurrent execution reduces the unpredictable delays in running transactions. It also reduces the average response time.

Example: Let T1 and T2 are two transactions. Transaction T1 transfers Rs. 2000 from account A to account B. It is defined as

```
T1:     read(A);
        A:= A-2000;
        Write (A);
        Read(B);
        B:=B + 2000;
        Write (B);
```

Transaction T2 transfers 10 percent of the balance from account A to account B. It is defined as

```
T2:     read (A);
        Temp: =A & 0.1;
        A:=A – temp;
```

```
Write (A);
read(B);
B:= B + temp;
Write (B);
```

Suppose the current values of account A and B are Rs. 10,000 and Rs. 5,000 respectively. Suppose the two transactions are executed in the order T1 followed by T2.

T1	T2
Read (A); A:= A – 2000; Write (A); Read (B); B = B + 2000; Write (B);	Read (A); Temp:= A * 0.1; A:= A – temp; Write (A); Read(B); B:=B + temp; Write(B);

The final values of account A and B are Rs. 7,2000 and Rs. 7,800 respectively. Thus the sum A + B is preserved.

If the transactions are executed in the order T2 followed by T1 the corresponding execution sequence is shown.

T1	T2
	Read (A); Temp:= A * 0.1; A:= A – temp; Write (A); Read(B); B:=B + temp; Write(B);
Read (A); A:= A – 2000; Write (A); Read (B); B = B + 2000; Write (B);	

After the execution of schedule 2, the sum A + B is preserved, and the final values of account A and B are Rs. 7,000 and Rs. 8,000 respectively.

The execution sequences which represents the chronological order in which instructions are executed in the systems, are called schedules.

Schedule 1 and schedule 2 are serial schedules several execution sequences are possible, since the various instructions from both the transactions may be interleaved.

Given two transactions can also be executed concurrently. One possible is shown.

T_1	T_2
Read (A); A:= A – 2000; Write (A);	
	Read (A); Temp:= A * 0.1; A:= A – temp; Write (A);
Read (B); B = B + 2000; Write (B);	
	Read(B); B:=B + temp; Write(B);

After execution of this schedule, we arrive at the same state as the one in which the transactions are executed serially in the order T1 followed by T2. The sum A + B is preserved.

Not all current executions result in a correct state. Consider a schedule shown. After the execution of this schedule, we arrive at a state where the final values of accounts A and B are Rs. 8,000 and Rs. 6,000, respectively. This final state is an inconsistent state.

T_1	T_2
Read (A); A:= A – 2000;	
	Read (A); Temp:= A * 0.1; A:= A – temp; Write (A); Read(B)
T_1	T_2
Write(A) Read(B) B:= B + 2000 Write(B)	
	B : = B + temp Write(B)

Schedule 4 – a concurrent schedule

We can ensure consistency of the database under concurrent execution by making such that any schedule that executed has the same effect of serial schedule.

Concurrency control:

- The system must control the interaction among the concurrent transactions. This control is achieved through one of concurrency control schemes. The concurrency control schemes are based on the serializability property. Now we will consider the management of concurrently executing transactions.

- Different types of protocols/schemes used to control concurrent execution of transactions are:

Lock Based Protocols:

- To ensure serializability it is required that data items should be accessed in mutual exclusive manner; if one transaction is accessing a data item, no other transaction can modify that data item. To implement this requirement locks are used. A transaction is allowed to access a data item only if it is currently holding a lock on that item.

Lock:

There are two models in which a data item may be locked:

- Shared mode lock:
  If a transaction Ti has obtained a shared mode lock on item Q, then Ti can read, but cannot write Q. It is denoted by S.

- Exclusive :
  If a transaction Ti has obtained as exclusive mode lock on item Q, then Ti can read and also write Q. It is denoted by X.

A transaction requests a shared lock on data item Q by executing the lock-S(Q) instruction. Similarly, a transaction requests an exclusive lock through the lock-X(Q) instruction. Similarly, a transaction can unlock a data item Q by the unlock (Q) instruction.

Given a set of lock modes, we can define a compatibility function on them as follows: Let A and B represent arbitrary lock modes. Suppose that a transaction Ti request a lock of mode A on item Q on which transaction Tj (Ti≠Tj) currently hold a lock of mode B. If transaction Tj can be granted a lock of Q immediately, in spite of the presence of the mode B lock, then we say mode A is compatible with mode B. Such a function is represented by a matrix.

	S	X
S	True	False
X	False	false

Lock compatibility matrix ‘comp’

- To access a data item, transaction Ti must first lock that item. If the data item is already locked by another transaction in an incompatible mode, the concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. Thus, Ti is made to wait until all incompatible locks held by other transactions have been released.

Example: Let A and B be two accounts that are accessed by transaction T1 and T2. Transaction T1 transfer $50 from account B to account A. It is shown in figure. Transaction T2 displays that total amount of money in accounts A and B.

```
T1: Lock-x (B);
Read (B);
B: =B-50;
Write (B);
Unlock (B);
Lock-X (A);
Read (A);
A: =A + 50;
Write (A);
Unlock (A);
```

Transaction T1

```
T2: lock-S(A);
Read (A);
Unlock (A);
Lock-S (B);
Read (B);
Unlock (B);
Display (A +B);
```

Transaction T2

T3	T4
Lock-x(B);	
Read(B);	
B :=B-50;	
Write(B);	
	lock-S(A);
	Read(A);
	Lock-S(B);
lock-S(A);	

Here T3 is holding an exclusive-mode lock on B and T4 is requesting a shared-mode lock on B, T4 is waiting for T3 to unlock B. Similarly, since T4 is holding a shared-mode lock on A and T3 is requesting an exclusive-mode lock on A, T3 is requesting an exclusive mode lock on A, T3 is waiting for T4 to unlock A. Thus, in requesting an exclusive mode lock on A, T3 is waiting for t4 to unlock A. Thus, in this situation neither of transactions can proceed with normal execution. This situation is called dead lock. When deadlock occurs, the system must rollback one of the two transactions. Once a transaction has been rolled back, the data items that were locked by that transaction are unlocked. These data items are then available to the other transaction, which can continue with its execution.

- Locking Protocols :

  Each transaction in the system should follow a set of rules, called a locking protocol, indicating when a transaction may lock and unlock each of the data items.
- Granting of locks:

When a transaction requests a lock on a data item in a particular mode an no other transaction has a lock on the same data item in a conflicting mode, the lock can be granted. However, care must be taken while granting the locks. For example, consider a transaction T2 has a shared-mode lock on data item Q, and another transaction T1 requests a exclusive-mode lock on the same data item. In this case, T1 has to wait for T2 to release the lock. Mean while, another transaction T3 requests shared mode lock on Q. The lock request is compatible with the lock granted to T2, so t3 may be granted the shared-mode lock. At this point, even though T2 releases the lock, T1 has to wait for T3 to release the lock. Thus, it is possible that sequence of transactions requests shared-mode lock on Q, and each transaction releases the lock a short while after it is granted, but T1 may never gets the exclusive mode lock on the data item. The transaction T1 may never make progress 1 and is said to be starved.

- Starvation of transaction can be avoided by granting locks in the following manner. When a transaction Ti requests a lock on a data item Q in a particular mode M, the concurrency control manager grants the lock provided that
  1. There is no other transaction holding a lock on Q in a mode that conflicts with M.
  2. There is no other transaction that is waiting for a lock on Q, and that made its lock request before Ti.

Two Phase Locking Protocol:

This protocol requires that each transaction issue lock and unlock requests in two phases.

i. Growing phase: In this phase, a transaction may obtain locks, but may not release any lock.
ii. Shrinking phase: in this phase, a transaction may release locks, but may not obtain any new lock.

Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters in the shrinking phase, and it cannot issue more lock requests.

Transaction T1 and T2 are not two phase while transaction T3 is two phase.

```
T3 :    Lock-x (B);
        Read (B);
        B: =B-50;
        Write (B);
        Unlock (B);
        Lock-X (A);
        Read (A);
        A: =A + 50;
        Write (A);
        Unlock (A);
```

Transaction T3

Advantage:

- The two-phase locking protocol ensures conflict serializability. Consider any transaction, the point in the schedule where the transaction has obtained it final lock is called the lock point of the transaction. Now, transaction can be ordered according to their lock points. This ordering is a serializability ordering for the transactions.

Disadvantages:

- Cascading rollbacks may occur under two-phase locking.

T5	T6	T7
Lock-X (A);		
Read (A);		
Lock-S(B);		
Read(B);	Lock-X (A);	
Write (A);	Read (A);	
Unlock (A);	Write (A);	
	Unlock (A);	
		Lock-S(A)
		Read(A)

Partial schedule under two-phase locking.

- Transactions T5,T6 and T7 are two phase, but failure of T5 after the read(A) instruction of T7 leads to cascading rollback of T6 and T7. Cascading rollbacks can be avoided by a modification of two-phase locking.

i. **Strict two-phase locking protocol:**

- This protocol requires that locking should be two phase, and all exclusive-mode locks taken by a transaction should be held until the transaction. This requirement prevents any transaction from reading the data written by any uncommitted transaction under exclusive mode until the transaction commits.

ii. **The rigorous two phase locking protocol.**

- This protocol requires that all locks be held until the transaction commits.

Timestamp Based Protocols

- Time stamp based protocol ensures serializability. It selects an ordering among transactions in advance using time stamps.

I. Timestamps:

- With each transaction in the system, a unique fixed timestamp is associated. It is denoted by TS(Ti). This timestamp is assigned by the database system before the transaction Ti starts execution. If a transaction Ti has been assigned timestamp TS(Ti), and new transaction Tj enters the system, then TS(Ti)<TS(Tj).
- Two methods are used for implementing timestamp:
  i) Use the value of the system clock as the timestamp, that is, a transactions timestamp is equal to the value of the clock when the transaction enters the system.
  ii) Use a logical counter, that is a transactions timestamp is equal to the value of logical counter, when transaction enters the system. After assigning a new timestamp, value of timestamp is increased.
- The timestamps of the transactions determine the serializability order. Thus, if TS(Ti)>TS(Tj), then the system must ensure that in produced schedule transaction Ti appears before transaction Tj.

To implement this scheme, two timestamps are associated with each data item Q.

i) W-timestamp(Q)denotes the largest timestamp of any transaction that executed write (Q) successfully.

ii) R-timestamp(Q) denotes the largest timestamp of any transaction that executed read(Q) successfully.

These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

The Timestamp Ordering Protocol:

The timestamp ordering protocol ensures that any conflicting read and write operations executed in timestamp order. This protocol operates as follows:

1. Suppose that transaction Ti issues read(Q).

   a) If TS(Ti) < W-timestamp(Q), then Ti needs a value of Q that was already overwritten. Hence, read operation is rejected, and Ti is rolled back.
   b) If TS(Ti) ≥W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and TS(Ti).
2. Suppose that transaction Ti issues write(Q).
   a) If TS(Ti)< R-timestamp(Q), then the value of Q that Ti is producing was needed previously,and the system assumed that the value would never be produced. Hence, the system rejects write operation and rolls Ti back.
   b) If TS(Ti) ≥W-timestamp(Q), then the Ti is attempting to write an obsolete value of Q. hence, the system rejects write operation and rolls Ti back.
   c) Otherwise, the system executes the write operation and sets W-timestamp(Q) to TS(Ti).

If a transaction Ti is rolled by the concurrency control scheme, the system assigns it a new timestamp and restarts it.

Example: Consider two transaction T14 and T15. Transaction T14 displays the sum of account A and B transaction T15 transfer $50 from account B to account A and displays the sum of both.

```
T14 :       Read (A);
            Read(B);
          Display( A+B);
T15 :       Read (B);
            B: =B-50;
            Write (B);
            Read (A);
            A: =A + 50;
            Write (A);
            Display(A+B);
```

A concurrent schedule for these two transactions:

T14	T15
Read (B);	Read (B);
	B: =B-50;
Read(A);	Write (B);
	Read (A);
	A: =A + 50;
	Write (A);
	Display(A+B);
Display(A+B);	

Schedule 3

As shown in fig., in schedule 3, TS(T14)<TS(T15) and the schedule is possible under timestamp protocol.

Advantages:

1. The timestamp ordering protocol ensures conflict serializability. This is because conflicting operations are processed in timestamp order.
2. The protocol ensures freedom form deadlock, since no transaction ever waits.

Disadvantages:

1. There is a possibility of starvation of long transactions if a sequence of conflicting short transactions causes repeated restarting of the long transaction.

   If a transaction is found to be getting restarted repeatedly, conflicting transactions need to be temporarily blocked to enable the transaction to finish.
2. The protocol can generate schedules that are not recoverable.

Thomas' Write Rule:

- Thomas' write rule is a modified version of timestamp ordering protocol.

Consider schedule 4 given in following Fig.

T16	T17
Read (Q)	
	Write(Q)
Write(Q)	

- Here, T16 starts before T17, therefore TS(T16)<TS(T17). The read(Q) operation of T16 succeeds, similarly the write(Q) operation of T17. When T16 attempts its write (Q) operation, it is rejected by the system and T16 is rolled back; as (TS(T16)<W-timestamp(Q)). Since W-timestamo(Q)=TS(T17);
- In this case, T17 has already written Q an the value of Q that T16 is attempting to write is one that will never need to be read. Thus, the rollback of T16 is required by timestamp ordering protocol, but it is unnecessary.
- Thomas' write rule, modifies the timestamp ordering protocol.

Thomas' write rule is:

Suppose that transaction Ti issues write (Q):

a) If TS(Ti) < R-timestamp(Q), then the value of Q that Ti is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls Ti back.
b) If TS(Ti) < W-timestamp(Q),then Ti is attempting to write an obsolete value of Q. hence, this write operation can be ignored.
c) Otherwise, this system executes the write operation and sets w-tiimestamps(Q) to TS(Ti).

The difference between timestamp ordering process and Thomas' write rule lies in the second rule. In the timestamp ordering protocol, if Ti issues write (Q) and TS(Ti) < W-timestamp(Q),Ti is rolled back. However, in Thomas' write rule if TS(Ti) < W-timestamp(Q),and TS(Ti) ≥ R-timestamp(Q), the write (Q) operation can be ignored.

Intent locks

Intent locks are used when SQL Server wants to acquire a shared lock or exclusive lock on some of the resources lower down in the hierarchy.

Intent locks include:

- intent shared (IS)
- intent exclusive (IX)
- shared with intent exclusive (SIX)
- intent update (IU)
- update intent exclusive (UIX)

Intent shared (IS) locks are used to indicate the intention of a transaction to read some resources lower in the hierarchy by placing Shared (S) locks on those individual resources.

Intent exclusive (IX) locks are used to indicate the intention of a transaction to modify some resources lower in the hierarchy by placing Exclusive (X) locks on those individual resources.

Shared with intent exclusive (SIX) locks are used to indicate the intention of the transaction to read all of the resources lower in the hierarchy and modify some resources lower in the hierarchy by placing Intent exclusive (IX) locks on those individual resources.

Intent update (IU) locks are used to indicate the intention to place Update (U) locks on some subordinate resource in the lock hierarchy.

Update intent exclusive (UIX) locks are used to indicate an Update (U) lock hold on a resource with the intent of acquiring Exclusive (X) locks on subordinate resources in the

lock hierarchy.

Shared intent update (SIU) locks are used to indicate shared access to a resource with the intent of acquiring Update (U) locks on subordinate resources in the lock hierarchy.

Deadlock:

- "A system is in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set. In other words, there exists a set of waiting transactions{T0,T1,…………..,Tn} such that T0 is waiting for a data item that T1 holds, and T1 is waiting for a data item that T2 holds, and ……….., and Tn-1 is waiting for a data item that Tn holds, and Tn is waiting for a data item that T0 holds. In such situation, none of transaction can make progress.

  There are two principal methods for dealing with the deadlock problem.

  i) Deadlock prevention: this approach ensures that system will never enter in deadlock state

  ii) Deadlock detection and recovery: this approach tries to recover from deadlock if system enters in deadlock state.

Deadlock Prevention:

There are two approaches for deadlock prevention:

1) One approach ensures that no cyclic waits can occur by ordering the requests for locks, or requiring all locks to be acquired together. This approach requires that each transaction locks all data items before it begins execution. It is required that, either all data items should be locked in one step, or none should be locked.

Disadvantages of this approach are:

a) It is hard to predict before the transaction begins, what data items need to be locked.
b) Data-item utilization may be very low, since many of the data items may be locked but unused for a long time.

2) The second approach for deadlock prevention is to use preemption and transaction rollbacks. In preemption when a transaction T2 requests a lock that transaction T1 holds, the lock granted to T1 may be preempted by rolling back T1, and granting of lock to T2. To control preemption, a unique timestamp is assigned to each transaction. The system uses timestamp to decide whether a transaction should wait or roll back.

Two different deadlock preventation schemes using timestamp are:

1) Wait die

- The wait-die scheme is non-preemption technique. In this, when transaction Ti requests a data item held by Tj, Ti is allowed to wait only if it has a timestamp smaller than Tj.(i.e. Ti is older than Tj). Otherwise, Ti is rolled back(dies).
- For example, consider three transactions T1,T2 and T3 with timestamps 5,10 and 15 respectively. If T1 requests a data item held by T2, then T1 will wait. If T3 requests data item held by T2, then T2 will be rolled back.

2) Wound wait

- The wound-wait is preemptive technique. In this, when transaction Ti request data item held by Tj, Ti is allowed to wait,only if it has timestamp greater than Tj (i.e. Ti is younger than Tj). Otherwise Tj is rolled back.
- Returning to same example, if T1 requests a data item held by T2, then the data item will be preempted by T2, and T2 will be rolled back. If T3 requests a data item held by T2 then T3 will wait.

Timeout Based Schemes

This approach for deadlock handling is based on lock timeouts. In this approach a transaction that has requested a lock waits for at most a specified amount of time. If the lock has not been granted within that time, the transaction is said to be time out, and it rolls back itself and restarts. Thus, if there was a deadlock one or more transactions involved in the deadlock will time out and roll back, allowing the others to proceed.

Advantages

- This scheme is easy to implement.
- It works well if transactions are short, and if long, waits are likely to be due to deadlocks.

Disadvantages

- It is hard to decide how long a transaction should wait. Too long waits results in unnecessary delays once a deadlock has occurred, and too short waits result in transaction rollbacks even when there is no deadlock.
- Starvation is also possible with this scheme.

Deadlock Detection and Recovery

This approach uses an algorithm that examines the state of the system periodically to determine whether a deadlock has occurred. If one has occurred, then the system attempts to recover from the deadlock.

Deadlock Detection.

- Deadlock can be described in terms of directed graphs called a wait-for graph. This graph consists of pair G=<V,E> ,where:

V - Set of vertices consist of all transaction in the system.
E - set of edges.

If Ti→Tj is in E, then there is a directed edge from transaction Ti to Tj. When transaction Ti requests a data item currently held by transaction Tj, then the edge Ti→Tj is inserted in the wait for graph. A deadlock exists in the system if and only if the wait for graph contains a cycle. Each transaction involved in the cycle is said to be dead locked.

Example : consider the wait for graph shown if figure:

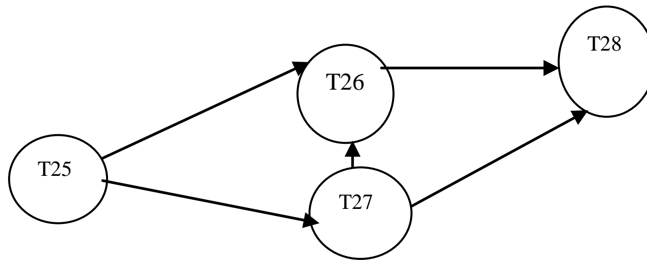
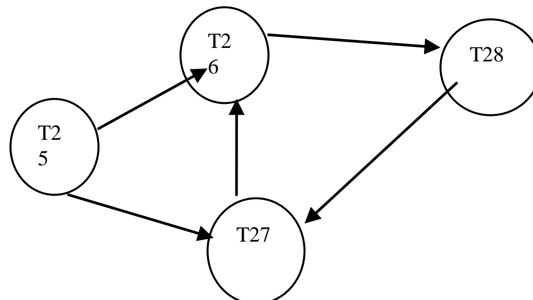


Fig (A) Wait for graph with no cycle

The graph depicts following situation:

- Transaction T25 is waiting for transactions T26 and T27.
- Transaction T27 is waiting for transaction T26.
- Transaction T26 is waiting for transaction T28.
- This graph has no cycle, therefore the system is not in a deadlock state.

Consider the graph shown in figure (B).



(B) Wait for graph with a cycle

Above the graph contains a cycle:
T26→T28→T27→T26

Therefore, the system is in deadlock state.

Recovery from Deadlock:

- When a deadlock detection algorithm determines that a deadlock exists, the system must recover from the deadlock. The most common solution is to roll back one or more transaction.

Three actions need to be taken:

1) Selection of transactions : Given a set of deadlocked transactions, we must determine which transactions should be rolled back to break the deadlock. The method used for this is : rollback those transactions that will incur minimum cost. Many factors determine the cost of transactions.
   i) How long the transaction has computed and how much longer the transaction will compute before it completes its task?
   ii) How many data items the transaction has used?
   iii) How many more data items the transaction needs to complete its task?
   iv) How many transactions will be involved in the rollback?
2) Rollback: once we have decided to roll back particular transaction, we must determine how far transaction should be rolled back. The solutions are:
   i) Total rollback: Abort the transaction and then restart it.
   ii) Partial rollback: It is more effective to roll back the transaction only as far as necessary to break the deadlock.
3) Starvation: It is possible that, same transaction will be rolled back number of times to break the deadlock. As the result, this transaction never completes its transactions can be picked as a victim only a small number of times.

Serializability

The database system must control concurrent executions of transactions, to ensure that the database state remains consistent. Some schedules will ensure consistency and some will not. Since transactions are programs it is difficult to determine exactly what operations a transaction performs and how operations of various transactions interact. Hence we consider only two operations, read and write.

There are two types of Serializability. They are

(i) Conflict Serializability
(ii) View Serializability.

Conflict Serializability:

Let us consider a schedule A in which there two consecutive instructions Ii are and Ij of transactions Ti and Tj resp (i not equal to j). if Ii and Ij refer to different data items, then Ii and Ij can be swaped without affecting the results of any instructions in the

schedule. However if Ii and ij refer to the same data item Q then the order of the two steps may matter. There are four cases to consider:

1. I_i = read(Q), I_j = read(Q). The order of I_i and I_j does not matter, since the same value of Q is read by T_i and T_j regardless of the order.
2. I_i = read(Q), I_j = write(Q). If I_i comes before I_j, then T_i does not read the value of Q that is written by T_j in instruction I_j. If I_j comes before I_i, then T_i reads the value of Q that is written by T_j. Thus the order of I_i and I_j matters.
3. I_i = write (Q), I_j = read (Q). The order of I_i and I_j matters, reason is same as previous case.
4. I_i = write(Q), I_j = write(Q). Since both instructions are write operations, the order of these instructions does not affect either T_i or T_j. However, the value obtained by the next read(Q) instruction of S is affected, since the result of only the latter of the two write instructions is preserved in the database.

Thus only in the case where Ii and Ij are Read instructions the execution matters. Ii and Ij conflict if they are operations by different transactions on same data item, and atleast one of these instructions is a write operation.

read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Here the Write (A) instruction of T1 conflicts with read (A) instruction of T2. However the Write (A) instruction of T2 does not conflict with the Read (B) instruction of T1, bcoz the two instructions access different data items. Since the Write (A) instruction does not conflict with the Read (A) in schedule 3, the instructions can be swapped to generate an equivalent schedule as below.

T1	T2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	

read(B)
write(B)

Schedule 3 and 5 produce a same final system.

Non conflicting instructions can be swapped as follows.

- Swap the read(B) instruction of T_1 with read(A) instruction of T_2.
- Swap the write(B) instruction of T_1 with write(A) instruction of T_2.
- Swap the write(B) instruction of T_1 with the read(A) instruction of T_2.

If a schedule S can be transformed into a schedule S' by a series of swaps of non conflicting instructions, then S and S' are said to be conflict equivalent. The concept of conflict equivalence leads to the concept of conflict Serializability. The schedule S is conflict serializable, if it is conflict equivalent to a serial schedule. Thus schedule 1 and schedule 3 are conflict serializable.

TI	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

- Schedule's' can be transformed into a schedule s by a series of swaps of non conflicting instructions we say that s and s' are conflict equivalent.
- Concept of conflict equivalence leads to the concept of conflict serializability.
- Schedule S is conflict serializable , if it's conflict equivalent to a serial schedule.
- Schedule 3 is conflict serializable, since it's conflict equivalent to the serial schedule1.

T3	T4
Read(Q)	Write(Q)
Write(Q)	

- It consists of 2 Transaction T3 and T4.
- This schedule is not conflict serializable, since it's not equivalent to either the serial schedule (T3,T4) (or) Serial schedule (T4,T3).

View Serializablity:

- Two schedules S and S' , where the same set of transactions participates in both schedules.
- The schedules S and S' are said to be view equivalent if three conditions are met

1. For each data item Q, if transaction Ti reads the initial values of Q in schedule S, then transaction Ti must, in schedule S' also read the intial value of Q.
2. For each data item Q, if transaction Ti executes read(Q) in schedule S and if the value was produced by a write(Q) operation executed by Tj, then the read(Q) operation of Ti must in schedule S' and also read the values of Q that was produced by the same write (Q) operation of transaction Tj.
3. For each data item Q, the transaction that performs the final write(Q) operation in schedule S must perform the final write (Q) operation in schedule S'.

- Concept of view equivalence to the concept of view serializablility.
- Schedule S is view serializable, if it's view equivalent to a serial schedule.

Schedule 8 – View serializable schedule:

T3	T4	T6
Read(Q)	Write(Q)	
Write(Q)		Write(Q)

- It's view equilvalent to the serial schedule <T3,T4,T5> since one read (Q) instruction reads the initial value of Q in both schedules and T5 performs the final write of Q in both schedules and T5 perform the final write of Q in both schedules.
- Every conflict serializable schedule is also view serializable, but there are view serializable schedules that are not conflict serializable.
- Schedule 8, transactions T4 and T6 performs write(Q) operations without having performed a read(Q) operations. Write of this sort are called blind writes.
- Blind writes appear in any view serializable schedule that is not conflict serializable.

Testing of Serializablity:

- It's done by using a directed graph, called procedure graph,

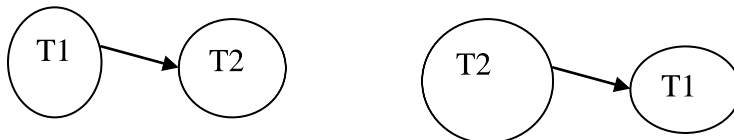
constructed form schedule.

- Graph consists of a pair G=(V,E) wehre 'v' is a set of vertices and 'E' is a sert of edges.
- Set of vertices consists of all transactions in Schedule.
- Set of edges consists of all edges Ti -> Tj for 3 conditions holds:

1. Ti executes write(Q) before Tj executes read(Q).
2. T i executes read(Q) before Tj executes write(Q).
3. Ti executes write (Q) before Tj executes write(Q).

- The precedence graph for schedule 1 contains a single edge T1->T2, since all the instructions of T1 are executed before the first instruction of T2 is executed.

a) Schedule 1 b) schedule 2

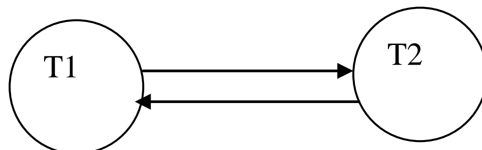


- T2 -→ T1 : All the instructions of T2 are executed before the first instruction of Ti.

Schedule 9:

T1	T2
Read(A);	
A:=A-50;	
	Read(A)
	Temp :=A * 0.1;
	A:=A-temp;
	Write(A);
	Write(B);
Write(A);	
read(B);	
B:=B+50;	B:=B+temp;
Write(B);	Write(B);

The precedence Graph for Schedule 9 is:



- A serializablility order of the transactions can be obtained through topological sorting, which determines a linear order consistent with the particular order of the precedence graph.

Recovery Isolation Levels:

Recovery system:

- It's an integral part of the database system.
- It stores the database to the consistant state that existed before the failure

- The recovery should provide high availability, that is, it must minimize the time for which the database is not usable after a crash.

Failure classification:

1. Transaction Failure:

   2types of errors.

   1. Logical error:
      - It occurs because of same internal condition,such as bad input, data not found, overflow or resource limit exceeded.
      - When logic error occurs, transaction cannot continue with it's normal execution.
   2. System Error:
      - Eg of system error is deadlock, when system error occurs, the system enters in an undesirable state and as a result, transaction cannot continue with it's normal execution.

1. System crash:
   - There is hard ware mal function or a bug-in the database software or in the operating system, that causes the loss of the content of volatile storage and brings transaction processing to a halt.
   - The assumption that hardware errors and bugs in the software bring system to a halt, but do not corrupt the non-volatile storage contents is known as the fail- stop.
2. Disk failure:
   - Disk block loses it's content as a result of either a head crash or failure during a data transfer operation.
   - Copies of the data on other disks or archival backup on tertiary media, such as tapes are used to recover from the failure.

Recovery schemes:

1. log-based Recovery.
2. Shadow Paging.

Log- based Recovery:

- Log is the most widely used structure for recording data base modification.
- The log is a sequence of log records, recording all the update activites in the data bases.

- There are several types of log records. An update log records describes a single database write.

1.Transaction Identifier:

- It's the unique indentifier of the transaction that performed the write operation.

2.Data-item Identifier:

- It's the unique identifier of the data item written. Typically it's the location on disk of the data item.

3. Old value is the value of the data items prior to the write
4. New value is the value that the data item will have after the write.

Various types of Log records are represented as:

- <Ti start>: Transaction Ti has started.
- <Ti, xj,v1,v2> : Transaction Ti has performed a write on data item xj.xj had value v1 before the write and will have value v2 after the write.
- <Ti commit>: Transaction Ti has committed.
- <Ti abort>: Transaction Ti has aborted.

Two techniques that uses log to ensure transaction atomicity despite failures are,

1. Deferred database Modification.
2. Immediate database modification.
3.

Deferred Database Modification (Deferred Update)

Eg: consider 2, Transactions to and T1 transaction T0.
Transfers $50 from account A to account B.

To :

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

Let transaction T1 withdraws $100 from account C

T1 :

```
read(C);
A:=A-50;
Write(A);
Read(B);
```

B:=B + 50;
Write (B);

Let transaction T1 withdrawn $100 from account C

T1 : Read(0);
C:C -100;
Write (c);

Portion of the database log corresponding to T0 and T1:

<T0 Start>
<T0, A, 950>
<T0,B, 2050>
<T0 commit>
<T1 start>
<T1,c,600>
<T1 commit>

State of the Log and DataBase corresponding to T0 and T1:

Log	Data Base
<To start> <T0, A, 950> <To, B,250> <T0 , commit> <T1, Start> <T1,C,600> <T1,commit>	A=950 B=2050 C=600

Redo(Ti):

- It sets the value of all data items updated by Transaction Ti to the new values.
- The same log as that in shown at 3 different times:

(a)	(b)	(c)
<To start>	<To start>	<To start>
<To,A,950>	<To,A,950>	<To,A,950>
<To,B,2050>	<To,B,2050>	<To,B,2050>
	<To,commit>	<To commit>
	<T1, start>	<T1 , Start>
	<T1,c,600>	<T1,c,600>
		<T1 commit>

1. Immediate DB Modification(Immediate Update)
    - It allows database modifications to be output to the database while the transaction is still in the active state.
    - Data modification written by active transactions are called uncommitted Modifications.

      Eg: <To start>
      <To,A,1000,950>

<To,B,2000,2050>
<To, commit>
<T1 start>
<T1,c,700,600>
<T1, commit>

State of System Log and database corresponding to T0 and T1:

Log	Data Base
<To start> <T0, A, 950> <To, B,2050> <T0 , commit> <T1, Start> <T1,C,600> <T1,commit>	 A=950 B=2050 C=600

- The recovery scheme uses 2 Recovery procedure:

1) Under (T1) restores the values of all data items updated by transaction Ti to old values.

2) Redo (Ti) Sets the values of all data items updated by transaction Ti to the new values.

- After a failure has occurred, the recovery scheme consults the log to determine which transaction need to be redone and which need to be undone:

- Transaction Ti needs to be undone if the log contains the record <Ti start>, but does not contain the record <Ti commit>

- Transaction Ti needs to be redone if the log contains both the record <Ti start> and the record <Ti commit>
  Case 1: write (B)
  Case 2: write (C)
  Case 3: <T1 commit>

  The same log, shown at 3 different times

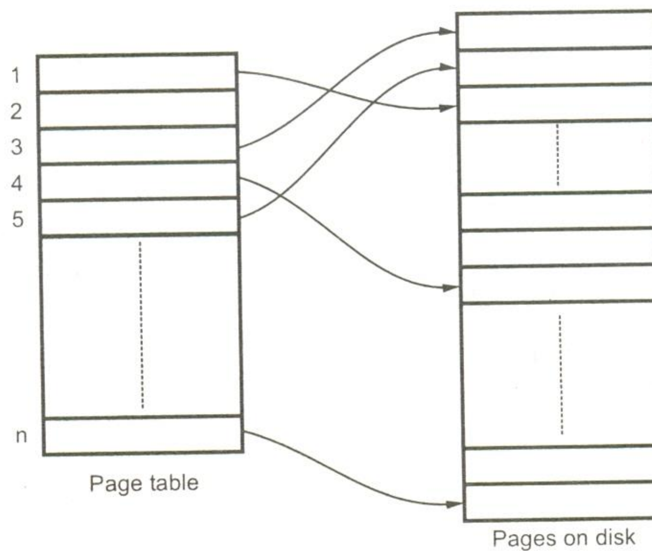
<To start>	<To start> <To,A,1000,950>	<To start> To,A,1000,950>

	<To,B,2000,2050>	<To,B,2000,2050>
	<To commit>	<To commit>
	<T1 start>	<T1 start>
	<T1, ci 700,600>	<T1, ci 700,600>rt>
		<T1 commit>

2. Shadow paging

- Its an alternative to log base crash-revocery tech. this scheme is useful if transcations execute serially.
- The Data base is partitioned into some number of fixed length blocks, which are referred to as pages
- The pages are stored in any random order on disk
- Therefore, there should be some way to find the page of db for nay given for this purpose page table is used.
- Page table has n-entries which points to n diff. pages on the disk
- Each entry of the page table contains pointer to one page on the disk.

Sample page table



The key idea behind the shadow paying tech. is to maintain 2 pages tables during the life of a transaction.

1. Current page table
2. Shadow page table.

Write operation is executed as follows:

1. If the I th page is not already in main memory then issue i/p(x)
2. If this is the write first performed on the ith page by this transaction, then the system modifies the current page table as follows:
    - ✓ Find an unused page on disk
    - ✓ Delete the pagefound in step2 (a) from the list of free page frames
    - ✓ Modify the current page table such that the ith entry points to the page found in step2 (a)
3. Assign the value of xj to x in the buffer page

Shadow and current page table

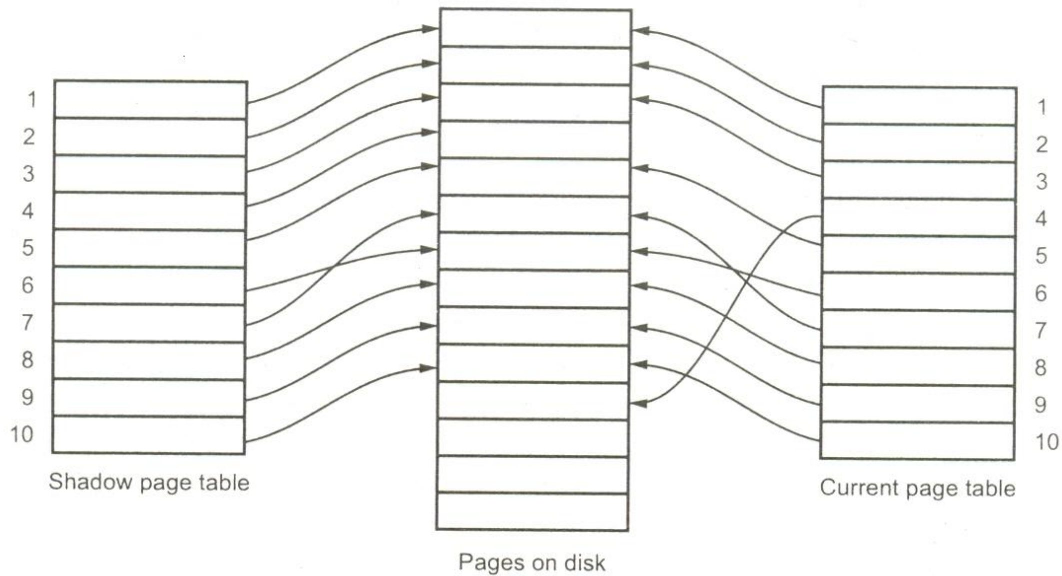


Fig. 4.47 Shadow and current page table.

Advantages

1. Shadow paying requires fewer disk accesses then the log base recovery
2. No overhead of writing record.
3. Recovery from crashes is significantly faster, since no undo and redo operations are needed.

Disadvantages

1. Commit overhead
2. Data fragmentation
3. Garbage collection
4. Hard to extend algorithms to allow transaction to run concurrently.

SQL Facilities for Concurrency:

The SQL standard defines four levels of transaction isolation in terms of three phenomena that must be prevented between concurrent transactions. These undesirable phenomena are:

Dirty read

A transaction reads data written by a concurrent uncommitted transaction.

No repeatable read

A transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read).

Phantom read

A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-committed transaction..

SQL Transaction Isolation Levels

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

PostgreSQL offers the read committed and serializable isolation levels.

Read Committed Isolation Level

Read Committed is the default isolation level in PostgreSQL. When a transaction runs on this isolation level, a SELECT query sees only data committed before the query began; it never sees either uncommitted data or changes committed during query execution by concurrent transactions. (However, the SELECT does see

the effects of previous updates executed within its own transaction, even though they are not yet committed.) In effect, a SELECT query sees a snapshot of the database as of the instant that that query begins to run. Notice that two successive SELECTs can see different data, even though they are within a single transaction, if other transactions commit changes during execution of the first SELECT.

UPDATE, DELETE, and SELECT FOR UPDATE commands behave the same as SELECT in terms of searching for target rows: they will only find target rows that were committed as of the query start time. However, such a target row may have already been updated (or deleted or marked for update) by another concurrent transaction by the time it is found. In this case, the would-be updater will wait for the first updating transaction to commit or roll back (if it is still in progress). If the first updater rolls back, then its effects are negated and the second updater can proceed with updating the originally found row. If the first updater commits, the second updater will ignore the row if the first updater deleted it, otherwise it will attempt to apply its operation to the updated version of the row. The query search condition (WHERE clause) is re-evaluated to see if the updated version of the row still matches the search condition. If so, the second updater proceeds with its operation, starting from the updated version of the row.

Because of the above rule, it is possible for updating queries to see inconsistent snapshots --- they can see the effects of concurrent updating queries that affected the same rows they are trying to update, but they do not see effects of those queries on other rows in the database. This behavior makes Read Committed mode unsuitable for queries that involve complex search conditions. However, it is just right for simpler cases. For example, consider updating bank balances with transactions like

```
BEGIN;

UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 12345;

UPDATE accounts SET balance = balance - 100.00 WHERE acctnum = 7534;

COMMIT;
```

If two such transactions concurrently try to change the balance of account 12345, we clearly want the second transaction to start from the updated version of the account's row. Because each query is affecting only a predetermined row, letting it see the updated version of the row does not create any troublesome inconsistency.

Since in Read Committed mode each new query starts with a new snapshot that includes all transactions committed up to that instant, subsequent queries in the same transaction will see the effects of the committed concurrent transaction in any case. The point at issue here is whether or not within a single query we see an absolutely consistent view of the database.

The partial transaction isolation provided by Read Committed mode is adequate for many applications, and this mode is fast and simple to use. However, for applications that do complex queries and updates, it may be necessary to guarantee a more rigorously consistent view of the database than the Read Committed mode provides.

Serializable Isolation Level

Serializable provides the strictest transaction isolation. This level emulates serial transaction execution, as if transactions had been executed one after another, serially, rather than concurrently. However, applications using this level must be prepared to retry transactions due to serialization failures.

When a transaction is on the serializable level, a SELECT query sees only data committed before the transaction began; it never sees either uncommitted data or changes committed during transaction execution by concurrent transactions. (However, the SELECT does see the effects of previous updates executed within its own transaction, even though they are not yet committed.) This is different from Read Committed in that the SELECT sees a snapshot as of the start of the transaction, not as of the start of the current query within the transaction. Thus, successive SELECTs within a single transaction always see the same data.

UPDATE, DELETE, and SELECT FOR UPDATE commands behave the same as SELECT in terms of searching for target rows: they will only find target rows that were committed as of the transaction start time. However, such a target row may have already been updated (or deleted or marked for update) by another concurrent transaction by the time it is found. In this case, the serializable transaction will wait for the first updating transaction to commit or roll back (if it is still in progress). If the first updater rolls back, then its effects are negated and the serializable transaction can proceed with updating the originally found row. But if the first updater commits (and actually updated or deleted the row, not just selected it for update) then the serializable transaction will be rolled back with the message

ERROR: Can't serialize access due to concurrent update because a serializable transaction cannot modify rows changed by other transactions after the serializable transaction began.

When the application receives this error message, it should abort the current transaction and then retry the whole transaction from the beginning. The second time through, the transaction sees the previously-committed change as part of its initial view of the database, so there is no logical conflict in using the new version of the row as the starting point for the new transaction's update.

Note that only updating transactions may need to be retried --- read-only transactions will never have serialization conflicts.

The Serializable mode provides a rigorous guarantee that each transaction sees a wholly consistent view of the database. However, the application has to be prepared to retry transactions when concurrent updates make it impossible to sustain the illusion of serial execution. Since the cost of redoing complex transactions may be significant, this mode is recommended only when updating transactions contain logic sufficiently complex that they may give wrong answers in Read Committed mode. Most commonly, Serializable mode is necessary when a transaction performs several successive queries that must see identical views of the database.

Question Bank
2 Mark Questions:

1. What is transaction?
2. What are ACID properties?
3. What are the two operations accomplished by the access of data base?
4. What are all the transaction state?
5. What are the two options available in aborted state?
6. What are the two types of serializability?
7. What are the necessary conditions for conflict serializability?
8. Write an example for non-recoverable procedures.
9. Why is it necessary to have control of concurrent execution of transaction? How is it made possible?
10. What is time stamp-ordering scheme? Specify
11. Define deadlock.
12. Define timestamp.
13. What are the various modes in which a data item be locked?
14. State the benefits of strict two-phase locking.
15. What are the two methods to prevent deadlock using timestamp?
16. What is check point?
17. List the recovery and advanced recovery techniques.
18. Define uncommitted modification
19. How can we handle disk crashes
20. What is shadow paging?

16 Mark Questions:

1. Explain the following protocols for concurrency control:
    a. Lock based protocols.
    b. Time stamp based protocols.
2. Write short notes on shadow paging.
3. Discuss on two-phase locking protocol.
4. Write short notes on log-based recovery.
5. Explain testing for serializability with respect to concurrency control schemes.

UNIT – V IMPLEMENTATION TECHNIQUES

Overview of physical storage media:

The storage media is classified by the speed with which data can be accessed cost per unit of data to buy the medium, and by the medium's reliability.

Cache memory:

The following are the features of a cache memory. They are

- ➔ It is the fastest and most costly form of storage.
- ➔ It is small in size
- ➔ It is managed by computer system hardware.

Main memory:

- The general purpose machine instructions operate on main memory.
- Although it contains mega bytes or even giga bytes, it is small for storing the entire database. Hence it is too expensive.
- The contents of main memory are usually lost if a power failure or system crash occurs.

Flash memory:

- It is also known as Electrically Erasable Programmable Read-Only Memory (EEPROM).
- The contents are not lost due to power failure.
- Reading data from flash memory takes less than 100 nano seconds (1/1000 of micro seconds).
- Writing data to flash memory is more complicated (i.e.) data can be written once that takes about 4 to 10 micro seconds and cannot be overwritten directly.
- If we want to overwrite, the entire memory has to be erased.
- The main drawback is it supports only limited no of erase cycles ranging from 10000 to 1 million.

Magnetic disk storage:

- This is the primary medium for the long term online storage of data.
- The entire database is stored on magnetic disk.
- The system must move the data from the disk to the main memory so that they can be accessed.

- If any modifications to the database it should be written to the magnetic disk.
- Power failures and system crashes doesn't affect the disk storage.
- It is referred to as "direct access storage" because it is possible to read data from any location on disk.

Optical storage:

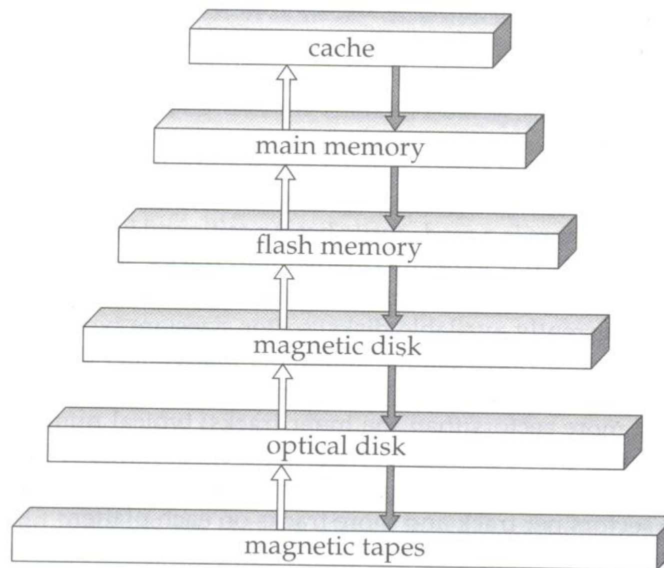
- The most popular forms of optical storage are compact disk (CD), which can hold upto 640 mega bytes of data, and the digital video disk (DVD), which can hold 4.7 or 805 giga bytes of data per side of the disk.
- For two sided disk upto 17 GB can be stored.
- Data are stored optically on a disk, and are read by a laser.
- The optical disks used in read only compact disks (CD-ROM), or read only digital video disks (DVD-ROM) cannot be written, but are supplied with data pre-recorded.
- There are "read once" version of compact disks called CD-R and digital video disk called DVD-R, which can be written only once. Such disks are often called Write once, read many (WORM) disks.
- There are also "multiple write" versions of compact disks called CD-RW and also digital video disks DVD-RW and DVD-RAM, which can be written multiple times.
- Juke box systems contain a few drives and numerous disks that can be loaded into one of the drives automatically.

Tape storage:

- Tape storage is used for backup and archival data. It is much cheaper than disks but data access is much slower because the tape must be accessed sequentially from the beginning. Hence it is referred to as "sequential access storage".
- Tapes have a high capacity and can be removed from the tape drive.
- Hence they are well suited for archival data.
- Tape juke boxes are used to hold exceptionally large collection of data from satellites, such as remote sensing data from satellites.

→The various storage media can be organized in a hierarchy according to their speed and their cost. The higher levels are expensive, but are faster. Down the hierarchy the cost per bit decreases whereas the access time increases.

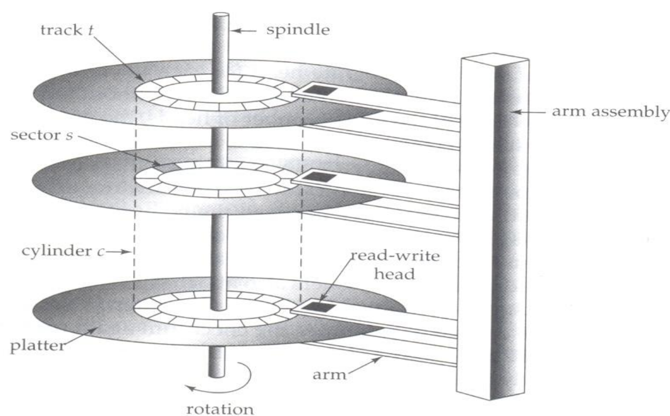
→The fastest storage media like cache and main memory are referred to as primary storage. The media in the next level like magnetic disk are referred to as secondary or online storage. The media in the lowest level of hierarchy like magnetic tapes and optical disks and juke boxes as referred to as tertiary storage or offline storage.



Storage volatility: → when the power to the device is removed it looses its contents. This is known as volatile storage. The cache and main memory are volatile storage. Other storage devices are non volatile storage.

Magnetic disk:

It is used for storing large amount of data. The most basic unit is bit of information. Bits are grouped into bytes or characters. Byte sizes from 4 to 8 bits. The capacity of the disk is measured by the no of bytes it can store.



Physical characteristics of Disks:

- Each disk platter has a flat circular shape.
- Its two surfaces are covered with a magnetic material, and information is recorded on the surfaces.
- Platters are made from rigid metal or glass and are covered with magnetic recording material. Such magnetic disks are called hard disks.
- The drive motor spins the disk at a constant high speed like 60, 90,120 or even 250 revolutions per second.
- There is a read write head positioned just above the surface of the surface of the platter.
- The disk surface is logically divided into tracks, which are subdivided into sectors.
- A sector is the smallest unit of information that can be read from or written to the disk.
- The inner tracks are of smaller length, around 200 sectors per track in the inner tracks and around 400 sectors per tracks in the outer tracks.
- A block is a contiguous sequence of sectors from a single track of one platter.
- Each side of the platter of a disk has a read-write head, which moves across the platter to access different tracks.
- A disk contains many plotters and the read write heads of all tracks are mounted on a single assembly called a disk arm, and move together.
- The disk plotters mounted on a spindle and the heads mounted on the disk arm are together known as head-disk assemblies.
- Since the head on all the platters move together, when the head on the platter is on the i th track, the heads on all other plotters are also on the i th track of their respective platters.
- Hence the i th tracks of all the platters together are called the i th cylinder.
- The read-write heads are kept as close as possible to the disk surface to increase the recording density.
- The head typically floats or flies only microns from the disk surface.
- The spinning of the disk creates small breeze, and the head assembly is shaped so that the breeze keeps the head floating just above the disk surface.
- The platters should me machined carefully to be flat because the head floats close to the surface and it may cause head crashes.
- The fixed head disk has a separate head for each track.
- This allows the computer to switch from track to track quickly without having to move the head assembly, but because of large number of heads, it's expensive.
- A disk controller interfaces between the computer system and the actual hardware of the disk drive.
- Disk controller also attaches checksums to each sector that is written, the checksum is computed from the data written to the sector.
- When the sector is read back, the controller computes the checksum again from the retrieved data and compares it with the stored checksum.
- If the data are corrupted, with a high probability the newly computed checksum will not match the stored checksum.
- If such an error occurs, the controller will retry the read several times.
- If the error continues to occur the controller will signal a read failure.

Performance measures of disks:

The main measures of the qualities of a disk are capacity, access time, data transfer rate, and reliability.

→Access time:

It is the time when a read or write request is issued to when data transfer begins.

→Seek time:

To access the data on a given sector of a disk, the arm first must move so that it is positioned over correct track, and then wait for the sector to appear under it as the disk rotates. The time for repositioning the arm is called the seek time.

→The average seek time:

The average seek time is the average of the seek times, measured over a sequence of random requests.

→Latency time:

Once the seek has started, the time spent waiting for the sector to be accessed to appear under the head is called the rotational latency. The average latency time of the disk is one-half the time for a full rotation of the disk.

→Data transfer rate:

It is the rate at which data can be retrieved from or stored to the disk.

→Mean time to failure:

It is the reliability of the disk. The mean time to failure of a disk is the amount of time the system runs without any failure.

RAID:

A variety of disk organization techniques, collectively called redundant array of independent disks. It is proposed to achieve improved performance and reliability. RAID systems are used for high reliability and higher performance rate.

Improvement of reliability via redundancy:

The solution to the problem of reliability is to introduce redundancy. The simplest but most expensive approach to introducing redundancy is to duplicate every disk. This technique is called Mirroring. A disk then consists of two physical disks, and every write operation is carried on both disks. If one of the disks fails then the data can be read from the other disk. Data will be lost only if the second disk fails before the first disk is repaired.

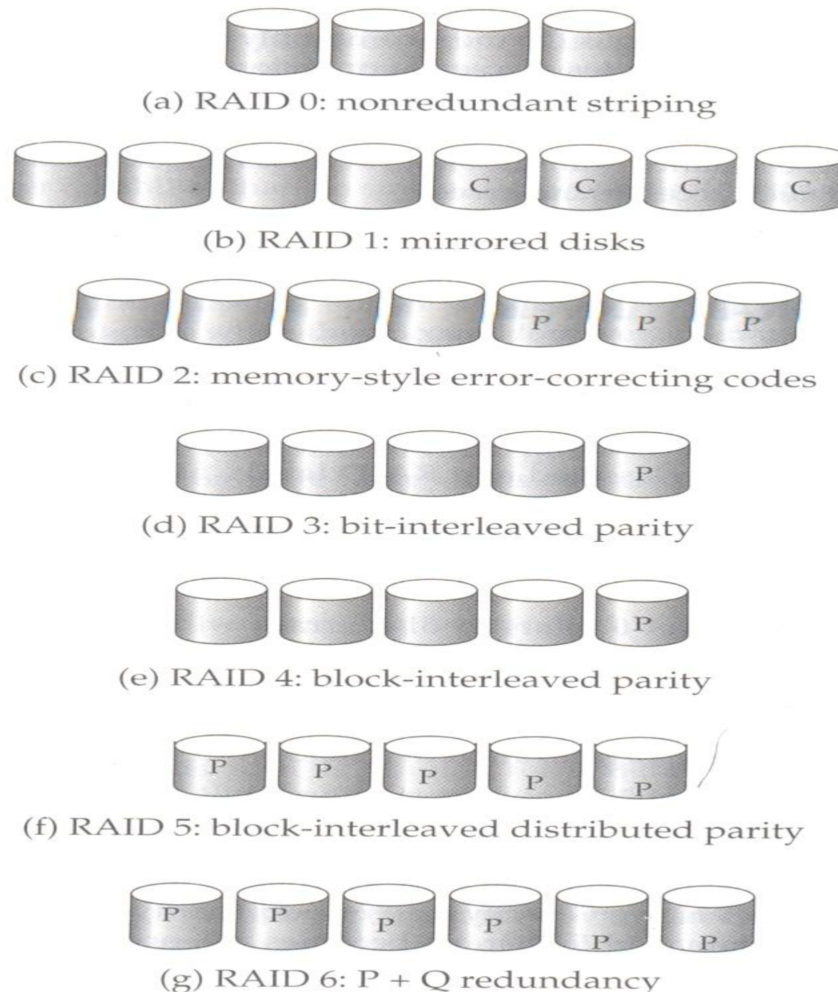
Improvement of performance via parallelism:

With disk mirroring, the rate at which read requests can be handled is doubled, since the read request can be sent to either disks. The transfer rate of each read is the same as in a single disk system, but the umber of reads per unit time has doubled. With

multiple disks we can improve the transfer rate as well by stripping data across multiple disks. Data stripping consist of splitting the bits of each byte across multiple disks. Such stripping is called bit-level stripping. Block –level stripping stripes blocks across multiple disks. It treats the array of disks as a single large disk, and logical numbers are given to each block.

There are two main goals of parallelism in a disk system:

- Load-balance multiple small access, so that the throughput of such accesses increases.
- Parallelize large access so that the response time of large access is reduced.



(a) RAID 0: nonredundant striping

(b) RAID 1: mirrored disks

(c) RAID 2: memory-style error-correcting codes

(d) RAID 3: bit-interleaved parity

(e) RAID 4: block-interleaved parity

(f) RAID 5: block-interleaved distributed parity

(g) RAID 6: P + Q redundancy

RAID Levels:

Mirroring provides high reliability, but it is expensive. Stripping provides high data transfer rates, but does not improve reliability. Various alternative schemes provide redundancy at lower cost by combining disk stripping with parity bits. These schemes are classified into RAID levels.

Level 0:

Refers to disk arrays stripping at level of blocks, but without redundancy.

Level 1:

Refers to disk mirroring with block stripping.

Level 2:

Known as memory-style error correcting code organization employs parity bits. Memory systems have long used parity bits for error detection and correction. Each byte in a memory system may have a parity bit associated with it, that records whether the numbers of bits in the byte that are set to 1 is even (parity=0) or odd (parity=1) .If one of the bit in the byte gets damaged, (either a 1 becomes a 0 or vice versa), the parity of the byte changes and thus do not match the stored parity. The idea of error-correcting codes can be used directly in disk arrays by striping bytes across disks.

Level 3:

Bit interleaved parity organization. The idea behind is that the disk controllers can detect whether a sector has been read correctly or not , so a single parity bit can be used for both error detection and correction. If one of the sectors gets damaged, the system knows exactly which sector it is and for each bit in the sector, the system can figure out whether it is a 1 or 0 by computing the parity of the corresponding bits from sectors in other disks. If the parity of the remaining bit is equal to the stored parity, the missing bit is equal to 0 or it is 1.

Benefits:→ less expensive compared to level 2.

→it needs only one parity disk for several regular disks and thus reduces storage space

→the transfer rate is N times faster compared to level 1 and 2.

Level 4:

Block interleaved parity organization. Uses block level stripping like level 0. Keeps the parity block on a separate disk. A block read accesses only one disk, allowing other request to be processed by the other disks. Thus the data transfer rate for each access is slower, but multiple read access can proceed in parallel leading to higher overall I/O rate. The transfer rate for large read and write is high due to parallelism.

Level 5:

Block interleaved distributed parity. It partitions data and parity among all N+1 disks instead of storing data in N disks and parity in one disk. Hence all disk can participate in read request. The parity block cannot store parity for blocks in the same disk because in case of disk failure would result in loss of data as well as parity.

Level 6:

It stores extra redundant information to guard against multiple disk failures. Instead of using parity, it uses error correcting codes such as reed Solomon

codes. 2 bits of redundant data are stored for every 4bits of data and the system can tolerate 2 disk failures.

Choice of RAID Levels:

The factors to be taken into account when choosing a RAID Level are

- ➔ Monetary cost of extra disk storage requirements.
- ➔ Performance requirements in terms of number of I/O operations
- ➔ Performance when a disk has failed
- ➔ Performance during rebuild (i.e.) while the data is failed disk is being rebuilt on a new disk)

Hot Swappable:

Disks can be removed and replaced by new ones without turning power off. Hot swappable reduces mean time to repair.

File organization:

A file is organized logically as sequence of records. These records are mapped onto disk blocks. Files are provided as basic construct in operating system. One approach to mapping the database to files is to use several files, and to store records of only one fixed length in any given file. Another way id to structure the files do that it can accommodate multiple lengths of records. Files of fixed length records are easier to implement than variable length records.

Fixed length records:

Consider a file of account records for bank database. Each record of this file is defined as

```
Type deposit = record
        Accno: char (10);
        Branchname: char (22);
        Balance: real
    End;
```

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Figure 11.6 File containing *account* records.

If we assume that each character occupies 1 byte and that a real occupies 8 bytes, then the record account is of length 40 bytes. A simple method is to use the first 40 bytes for the first record and the next 40 bytes for the second record, and so on. There are 2 problems with this approach. They are

- It is difficult to delete a record from this structure. The space occupied by the record to be deleted must be filled with some other record of the file, or there must be a way of marking deleted records so that they can be ignored.Unless the block size happens to be multiples of 40 some records will cross the block boundaries. (i.e.) part of the record will be in one block and another part in another block.

- When a record is deleted, the records that come after it are moved into the free space occupied by the deleted record. This will cause moving large no of records. Hence instead of this method the final record can be moved to the free space occupied by the deleted record. But this will cause additional block access. Hence it is acceptable to leave open the space occupied by the deleted record and wait for subsequent insertion. A simple marker can be given to the deleted record but it is not enough because it is hard to find the available space while insertion. Hence an additional structure is required.

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Figure 11.7 File of Figure 11.6, with record 2 deleted and all records moved.

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 8	A-218	Perryridge	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600

Figure 11.8 File of Figure 11.6, with record 2 deleted and final record moved.

At the beginning of the file, certain no of bytes are allocated as a file header. The header will contain the information about the file. The address of the first record whose contents are deleted should be stored in the header.

The first record should contain the address of the second available record. Since they point to the location of a record it is also referred to as pointers. The deleted records thus forms a link list, which is referred to as free list. If no space is available, the new record is added to the end of file.

header				
record 0	A-102	Perryridge	400	
record 1				
record 2	A-215	Mianus	700	
record 3	A-101	Downtown	500	
record 4				
record 5	A-201	Perryridge	900	
record 6				
record 7	A-110	Downtown	600	
record 8	A-218	Perryridge	700	

Figure 11.9 File of Figure 11.6, with free list after deletion of records 1, 4, and 6.

Variable –Length records:

Variable length records arise in database in several ways.

- Storage of multiple record types in a file.
- Record types that allow variable lengths for one or more fields.
- Record types that allow variable repeating fields.

Consider a different representation of the account information stored in the file in which one variable length record is used for each branch name and for all account information for that branch.

```
Type account-list = record
                        Branchname: char (22);
                        Account-info: array [1….infinity] of
                              Record;
                                    Accno: char (10)
                                    Balance: real;
                              End;
                        End;
```

There is no limit of how large a record can be.

Byte- String representation:

A simple method for implementing variable length records is to attach a special "end of record" symbol to the end of each record. Each record can be stored as string of consecutive bytes. Another type of byte string representation stores the record length at the beginning of each record, instead of using end of record symbol.

0	Perryridge	A-102	400	A-201	900	A-218	700	⊥
1	Round Hill	A-305	350	⊥				
2	Mianus	A-215	700	⊥				
3	Downtown	A-101	500	A-110	600	⊥		
4	Redwood	A-222	700	⊥				
5	Brighton	A-217	750	⊥				

Figure 11.10 Byte-string representation of variable-length records.

Disadvantage:

- It is not easy to reuse the space occupied by a deleted record.
- There is no space for records to grow longer. If a variable length record becomes longer, it must be moved. Movement is costly if pointers to record that are stored elsewhere in the database.

A modified form of byte string representation called “slotted page” structure is commonly used for organizing records within a single block. The slotted page structure has a header at the beginning of each block, containing

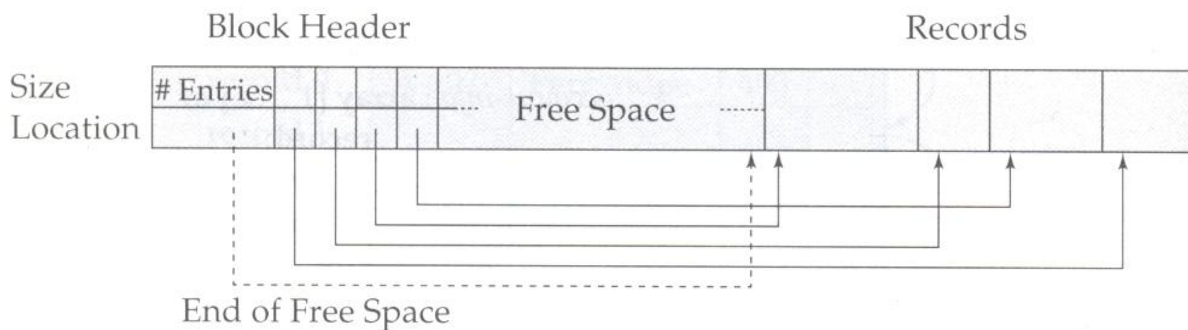


Figure 11.11 Slotted-page structure.

- The no of record entries in the header.
- The end of free space in the block.
- An array whose entries contain the location and size of each record.

The actual records are allocated contiguously in the block, starting from the end of the block. The free space in the block is contiguous, between the final entry in the header array, and the first record. If a record is inserted, space is allocated for it at the end of free space, and an entry containing its size and location is added to the header.

If a record is deleted the space that is occupied is freed and its entry is said to deleted. The records followed by the deleted records are moved further so that the free

space created by the deletion gets occupied and all free space is again between final entry in the header array and the first record. The end of free space pointer in the header is updated as well. In this method the pointer points to the entry in the header that contains that contains the actual location and not to the records directly.

Fixed-Length Representation:

Another way to implement variable length records efficiently in a file system is to use one or more fixed length records to represent one variable length record. There are two ways of doing this:

1. Reserved space:
   If there is a maximum record length that is never exceeded, we can use fixed length records of that length. Unused space is filled with a special null, or end of record, symbol.

2. List representation:
   We can represent variable length records by lists of fixed length records, chained together by pointers. If we choose to apply reserved-space method to account example, then select maximum length record with a maximum of 3 accounts per branch. Those branches with less than 3 accounts then they will have null fields, that field can be represented using end of record symbol.

The reserved space method is useful when most records have a length close to the maximum. Otherwise a significant amount of space may be wasted. Some branches may have many more accounts. Hence linked list method is used. To represent this, a pointer field is added.

0	Perryridge	A-102	400	A-201	900	A-218	700
1	Round Hill	A-305	350	⊥	⊥	⊥	⊥
2	Mianus	A-215	700	⊥	⊥	⊥	⊥
3	Downtown	A-101	500	A-110	600	⊥	⊥
4	Redwood	A-222	700	⊥	⊥	⊥	⊥
5	Brighton	A-217	750	⊥	⊥	⊥	⊥

Figure 11.12 File of Figure 11.10, using the reserved-space method.

The file structures of figures both use pointers. The difference is that, in fig 1 we use pointers to chain together only deleted records. Whereas in fig 2, we chain together all records pertaining to the same branch.

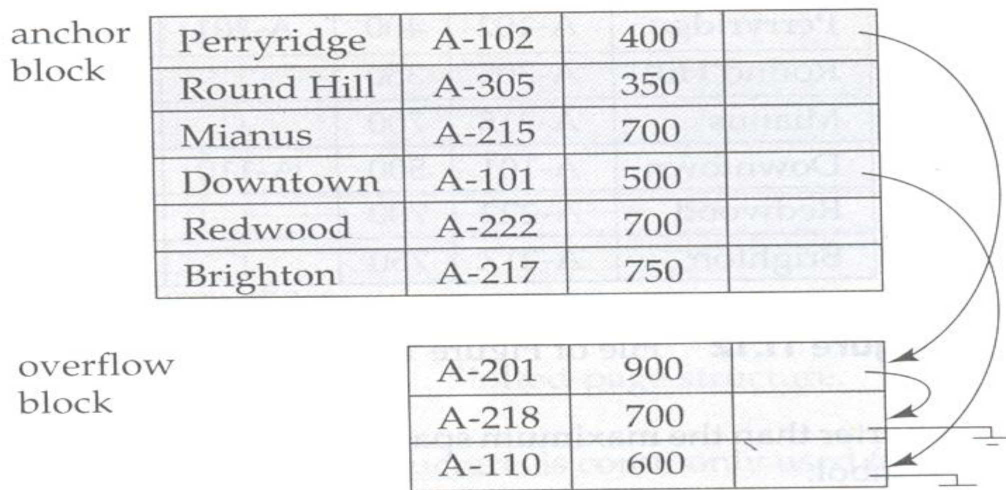
0	Perryridge	A-102	400
1	Round Hill	A-305	350
2	Mianus	A-215	700
3	Downtown	A-101	500
4	Redwood	A-222	700
5		A-201	900
6	Brighton	A-217	750
7		A-110	600
8		A-218	700

Figure 11.13 File of Figure 11.10 using linked lists.

A disadvantage to the structure of fig 2 is that the waste space in all records except the first in a chain. The first record needs to have the branch name value but subsequent records do not. We need to include a field for branch name in all records, lest he records not be of fixed length. To deal with this problem we allow to kinds of blocks in our file

1. Anchor blank→ contains the first record of a chain.

2. Overflow block:→ contains records other than those that are the first record of the chain. Thus all the records within a block have the same length, even though not all records in the file have the same length.



anchor block

Perryridge	A-102	400
Round Hill	A-305	350
Mianus	A-215	700
Downtown	A-101	500
Redwood	A-222	700
Brighton	A-217	750

overflow block

A-201	900
A-218	700
A-110	600

Figure 11.14 Anchor-block and overflow-block structures.

Organization of Records in Files:

Several ways:

1. Heap file Organization:
   - Any record can be placed anywhere in the file where there is space for the record
   - There is no ordering of records.
   - There is a single file for each relation.

2. Sequential file Organization:
   - Records are stored in sequential order according to the values of a "search key" of each record

3. Hashing File Organization:
   - Hash function is computed on some attribute of each record.
   - Result of the hash function specifies in which block of the file the record should be placed.
   - A separate file is used to store the records of each relations.
   - An clustering file organization, records of several different relations are stored in the same file.

4. Sequential file organization:
   - It's designed for efficient processing of records in sorted order based on some search-key.
   - Search key is any attribute or set of attributes it need not be the primary key or even a super key.
   - To permit fast retrieval of records in search –key order, we chain together records by pointers.
   - The pointer in each record points to the next record in search-key order.

A1	aa	1000	
A2	bb	2000	
A3	cc	3000	
A4	dd	4000	

A5	ee	5000	
A6	ff	6000	
A7	gg	7000	
A8	hh	8000	

Sequential File for Account Records:

- Sequential file organization allows records to be read in sorted order, that can be useful for display purposes.
- It's difficult to maintain physical sequential order as records are inserted and deleted. Since it's costly to move many records as a result of a single insertion or deletion. We can manage deletion by using pointer chains.

Rules:

1. Locate the record in the file that comes before the record to be inserted in search – key order.
2. If there is a free record within the same block as this record, insert the new record there.
    - Insert the new record in an overflow block.
    - The file after the insertion of the record(North Town, A888,800).
    - Few records need to be stored in overflow block.
    - Below search-key order and physical order may be totally lost.
    - Sequential processing will become much less efficient. At this point, the file should be reorganized.
3. Clustering File Organization:
    - Many Relational-database systems store each relation in a separate file, so that they can take full advantage of the file system that the
    - operating system provides.
    - Tuples of relation can be represented as fixed length records.
    - This simple implementation of a relational database system is well suted to low cost database implementation.
    - Many large-scale database system do not rely directly on the underlying operating system for the file Management.
    - One large os file is allocated to the database systems.
    - Database system stores all relations in this one file and manages the file itself.

- Advantages ofstoring many relation in one file, SQL Query for the bank database.
  Select account-number ,customer_name,customer_street, customer_city, from depositor, customer.
  Where depositor_customer_name=Customer.customer_name;
  - Join two relations. Depositor and customer.

Depositor Relation

Customer_name	Account_no
A A	A1
A A	A2
AA	A3
B B	A4

Customer Relation

Customer_name	Customer_street	Customer_city
AA	Main	Brooklyn
BB	Putnam	Stamford

- Clustering file organization stores related records of 2 or more relations in each block such a file organization allows us to read records that would satisfy the join condition by using one block read.
- Clustering has enhanced processing of a particular join,(depositor and customer) but it results in slow processing of their types of query.

Clustering File Structure:

AA	MAIN	Brooklyn
AA	A1	
AA	A2	
AA	A3	
BB	Putnam	stamford
BB	A4	

Clustering File Structure with pointer chains

AA	MAIN	Brooklyn
AA	A1	
AA	A2	
AA	A3	
BB	putnam	stamford
BB	A4	

STRUCTURE FOR FILES:

Two Basic kind of Indices:

1. Ordered Indices: Based on a sorted ordering of the values.

2. Hash Indices: Based on a uniform distribution of values across a range of buckets.

- The bucket to which a value is assigned is determined by a function called hash function.
- Each technology must be evaluated on the basis of these factors.

1. Access Types:
   - Types of access that are supported efficiently.
   - It can include finding records with a specified attributes values and finding records whose attributes values in a specified range.
2. Access time:
   - The time it takes to find a particular data items or set of items using the technology in question
3. Insertion time:
   - The time it takes to insert a new data items.
   - This value includes the time it takes to find correct palce to insert the new data as well as the time it takes to update the index structure.
4. Deletion time:
   - The time it taken to delete a data item, this value includes the time it taken to find the itmes to be deleted, as well as the time it takes to update the index structure.
5. Space overhead:
   - The additional space occupied by an index structure provided that the amount of additional space is moderate it's usually worth-while to sacrifice the space to achieve improved performance.

Indexing And Hashing:

An index for a file in a database system works in much the same way as the index in a book. Database system indices play the same role as book indices or card catalogs in library. There are two basic kinds of indices:

Ordered indices: This is based on a sorted ordering of the values.

Hash indices:

This is based on a uniform distribution of values across range of buckets. The bucket to which a value is assigned is determined by a function, called a hash function.

There are several techniques for both the indices, but none of these are the best. Each technique is suited to only particular database applications. Each technique must be evaluated on the basis of the following factors,

Access types:

Finding records with a specific attribute value and finding records whose attribute values fall in a specified range.

Access time:

The time it takes to find a particular data item, or set of items, using the technique.

Insertion time:

The time it takes to insert a data item. This value includes the time it takes to find the correct place to insert a new data item, as well as the time it takes to update the index structure.

Deletion time:

The time it takes to delete the data item. This value includes the time it takes to find the item to be deleted as well as the time it takes to update the index structure.

Space overhead:

The additional space occupied by an index structure. An attribute or a set of attributes used to look up records in a file is called a search key.

Ordered indices:

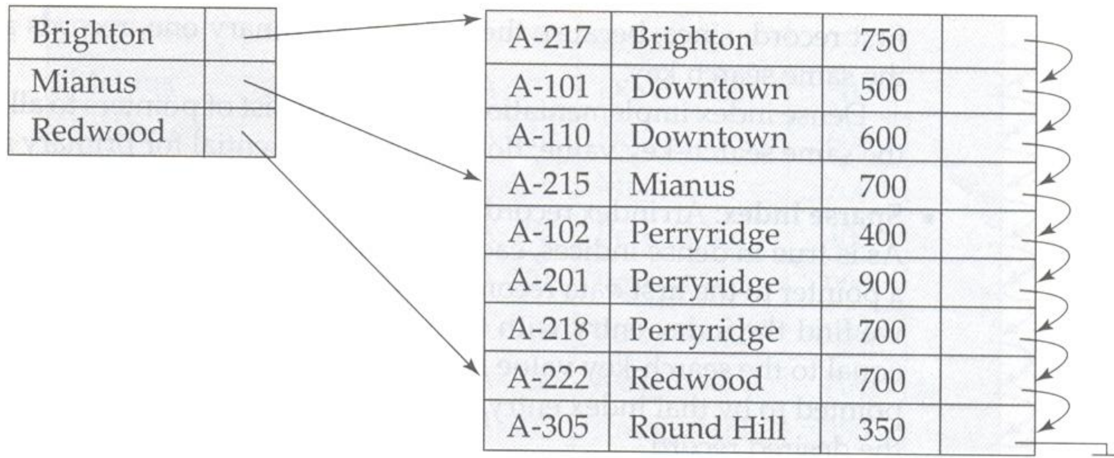
To gain a fast random access to records in a file, an index structure is used. Each index structure is associated with a particular search key. The records in the index file may themselves be stored in some sorted order. A file may have several indices, on a different search keys. If the file containing the records is sequentially ordered then a primary index is used.(index on primary key) primary index are also called as clustering indices. Indices whose search key specifies an order different from the sequential order of the file are called secondary indices or non clustering indices.

Primary indices:

These indices are designed for both sequential processing of entire file and random access to individual access. The figure below shows the records are stored in search key order with branch name used as search key.

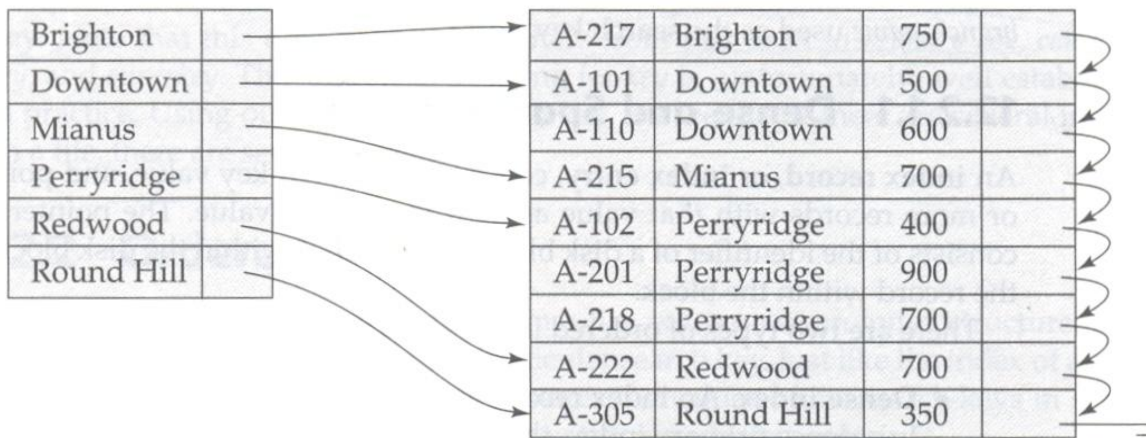
Dense and sparse indices:

An index record or index entry consists of a search key value and pointers to one or more records with that value as their search key value. The pointer to a record consists of the identifier of a disk block and a n offset within the disk block to identify the record within the block. There are two types of ordered indices



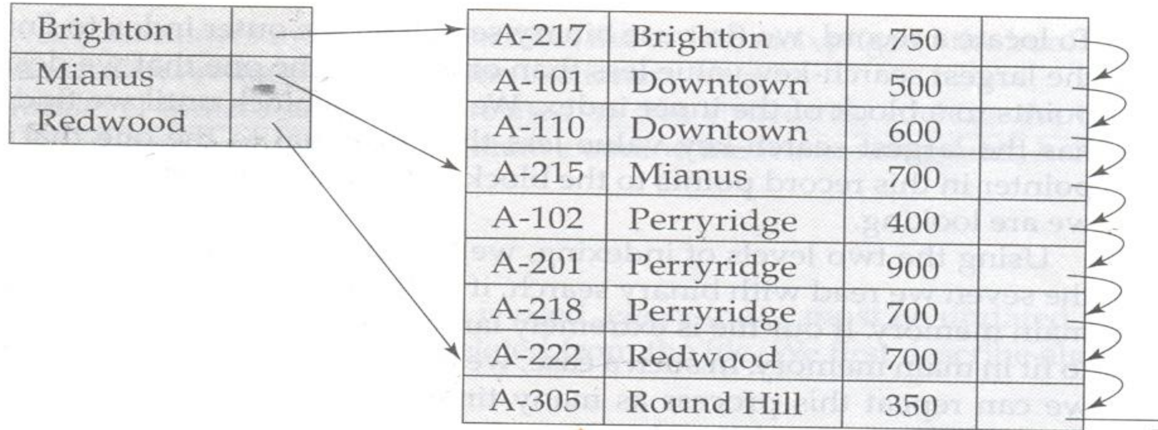
Dense index:

- An index record appears for every search key value in the file.
- The index record contains the search key value and a pointer to the first data record with that search key value.
- The rest of the records with the same search key value are stored sequentially after the first record.



Sparse index:

- An index records appears for only some of the search key values.
- 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, find the index entry with the largest key value that is less than or equal to the search key value.



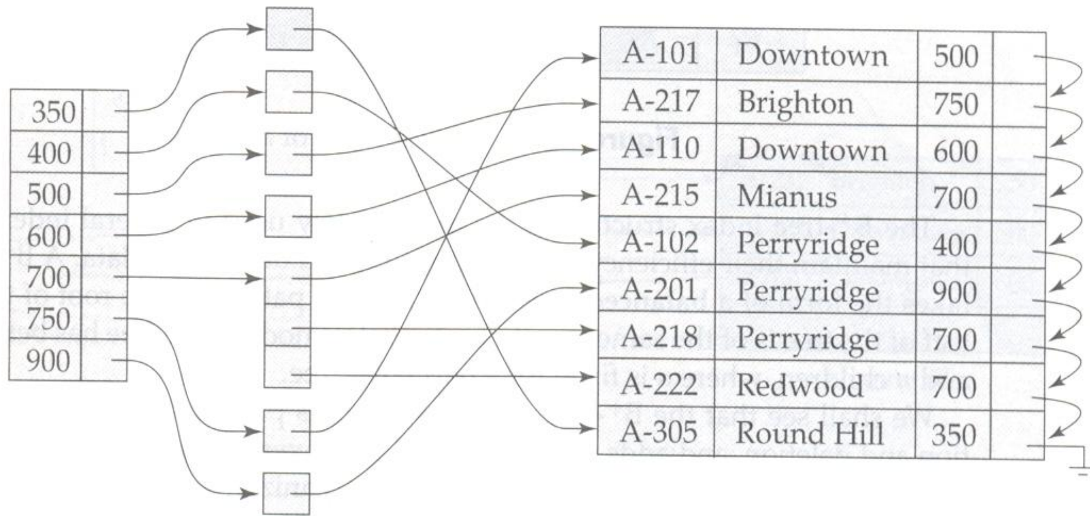
Secondary indices:

Secondary indices may be dense, with an index entry for every search value, and a pointer to every record in the file. A secondary index on a candidate key looks just like a dense primary index, except that records pointed to by successive values in the index are not stored sequentially.

However secondary indices may have a different structure from primary indices. If the search key of the primary indices is not a candidate key, it suffices if, the index points to the first record with a particular value for the search key, since the other records can be fetched by a sequential scan of the file.

In contrast, if the search key of a secondary index is not a candidate key, it is not enough to point to just the first record with each search key value. The remaining records with the same search key value could be anywhere in the file, since the records are ordered by search key of the primary index, rather than by the search key of the secondary index. Therefore the secondary index must contain pointers to all the records.

An extra level of indirection can be used to implement secondary indices on search keys that are not candidate keys. The pointers in such a secondary index do not point directly to the file. Instead each points to a bucket that uses an extra level of indirection on the account file, on search key balance.



Multilevel indices:

Indices with two or more levels are called multilevel indices. Searching for records in multilevel index requires significantly fewer I/O operations than searching for records by binary search.

If an index is sufficiently small to be kept in main memory, the search time to find an entry is low. If the index is so large that it must be kept on disk, a search for an entry requires several disk block reads. If the index occupies b blocks, binary search requires as many as [log2 (b)] blocks to be read. For 100 blocks index, binary search requires seven block reads.

To deal with this problem, two level index structure is used. To locate a record, we first use binary search on the outer index to find the record for the largest search key value less than or equal to the one that we desire. The pointer points to the block of inner index. We scan the book until we find the record that has the largest search key value less than or equal to the one that we desire. The pointer in this record points to the block of the file that contains the record for which we are looking.

Assume that the outer index is already in memory, then using the two levels of indexing, only one index block can be read, rather than the seven read with binary search. In such case another level of index can be created.

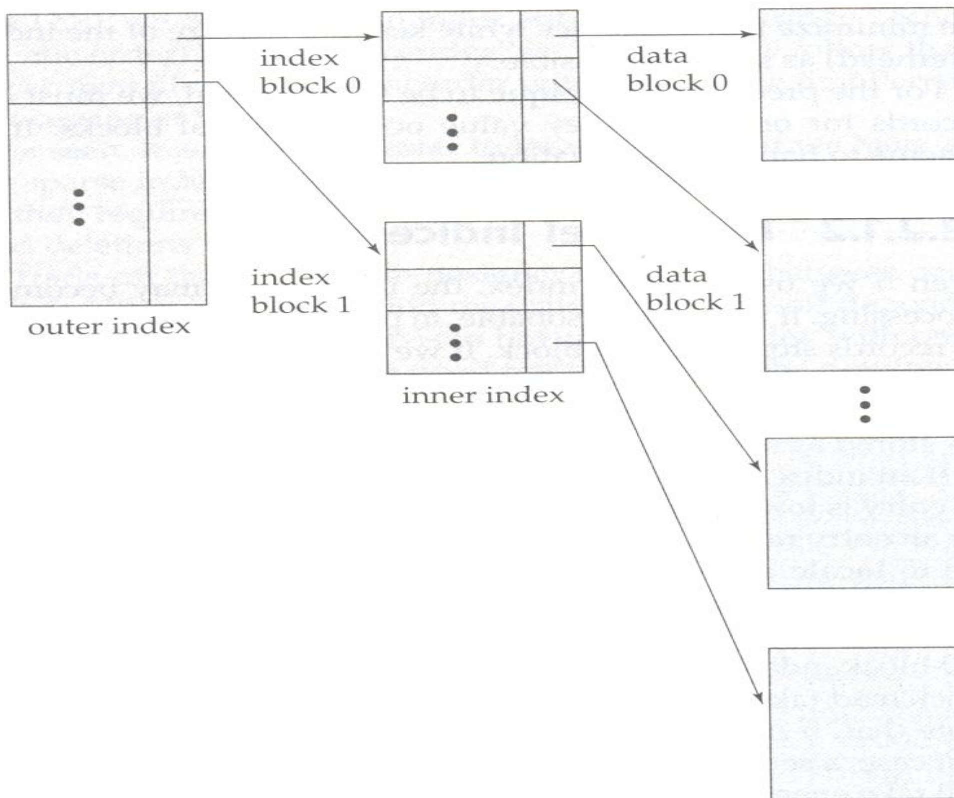


Figure 12.4 Two-level sparse index.

Hashing Technique:

One disadvantage of sequential file organization is that, to locate data either an index structure should accessed or binary search should be used. This results in more I/O operations. File organization based on the technique of hashing allow us to avoid accessing an index structure. Hashing also provides a way of constructing indices. There are two types of hashing techniques.

- Static hashing
- Dynamic hashing

Static hashing:

In a hash file organization, the address of the disk block containing a desired record can be obtained directly by computing a function on a search key value of a record. The term bucket is used to denote a unit of storage that can store one or more records. A bucket is typically a disk block but could be chosen to be smaller or larger than a disk block.

Let k denote the set of all search key values, and let b denote the set of all bucket addresses. A hash function h is a function from k to b. Let h denote hash function.

To insert a record with search key k, h (ki) gives the address of the bucket for that record. If there is space in the bucket to store the record then the record is stored in the bucket.

To perform a lookup on the search key value ki, then the bucket is searched using h (ki) address. Suppose that two search keys k5 and k7 have same hash value and when a lookup is performed then h (k5) will contains records with search key values k5 and k7. Thus the search key value of every record in the bucket should be checked to verify that the record is one that we want.

In case of deletion, the search key value of the records to be deleted is ki, then the bucket for that record is searched using h (ki) and the record is deleted.

Hash function:

A hash function distributes the stored keys uniformly across all the buckets, so that every bucket has the same no of records.

* The distribution is uniform → the hash function assigns each bucket the same no of search key values from the set of all possible search key values.

* The distribution is random → each bucket will have the nearly the same no of values assigned to it regardless of the actual distribution of search key values. The hash value will not be correlated to any externally visible ordering by the length of the search key values, such as alphabetic ordering or ordering by length of the search keys, the hash function will appear to be random.

Handling of bucket overflows:

When a record is inserted, the bucket to which it is mapped has space to store the record. If the bucket does not have enough space, a bucket overflow is said to occur. Bucket overflow can occur for several reasons:

Insufficient buckets:

No of buckets > total no of records that will be stored / the no of records that will fit in a bucket.

The total no of records will be known when the hash function is chosen.

Skew:

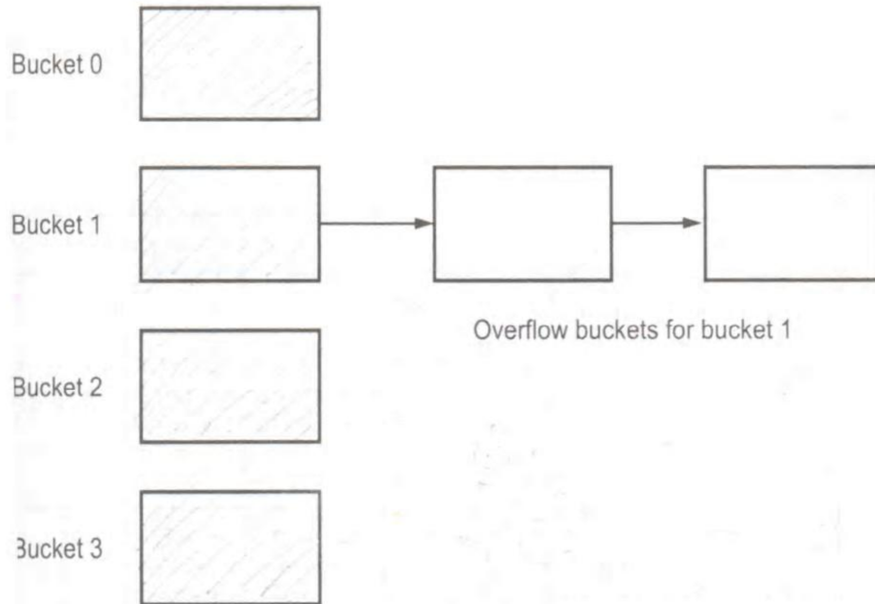
Some buckets are assigns more records than others, so a bucket may overflow even when other buckets still have space. This situation is called bucket skew. Skew can occur for two reasons,

→ Multiple records may have the same search key.
→ The chosen hash function may result in non uniform distribution of search keys.

Bucket overflow can be handled using overflow buckets. If a record must be inserted into a bucket b, and b is already full, the system provides overflow bucket for b, and inserts the record into the overflow bucket. If the overflow bucket is also full, the system provides another overflow bucket and so on. All the overflow buckets of a given

bucket are chained together in a linked list as shown in the fig. overflow handling using such a linked list is called overflow chaining.

To handle this overflow chaining the lookup algorithm should be changed. As before the system uses the hash function on the search key to identify a bucket b. the system must examine all the records in bucket b to see whether they match the search key as before. In addition, if bucket b has overflow buckets the system must examine the records in all the overflow buckets also.



This form of hash structure is sometimes referred to as closed hashing. In an alternative approach called open hashing, the set of buckets is fixed, and there are no overflow chains. Instead if a bucket is full, the system inserts records in some other bucket in the initial set of buckets b.

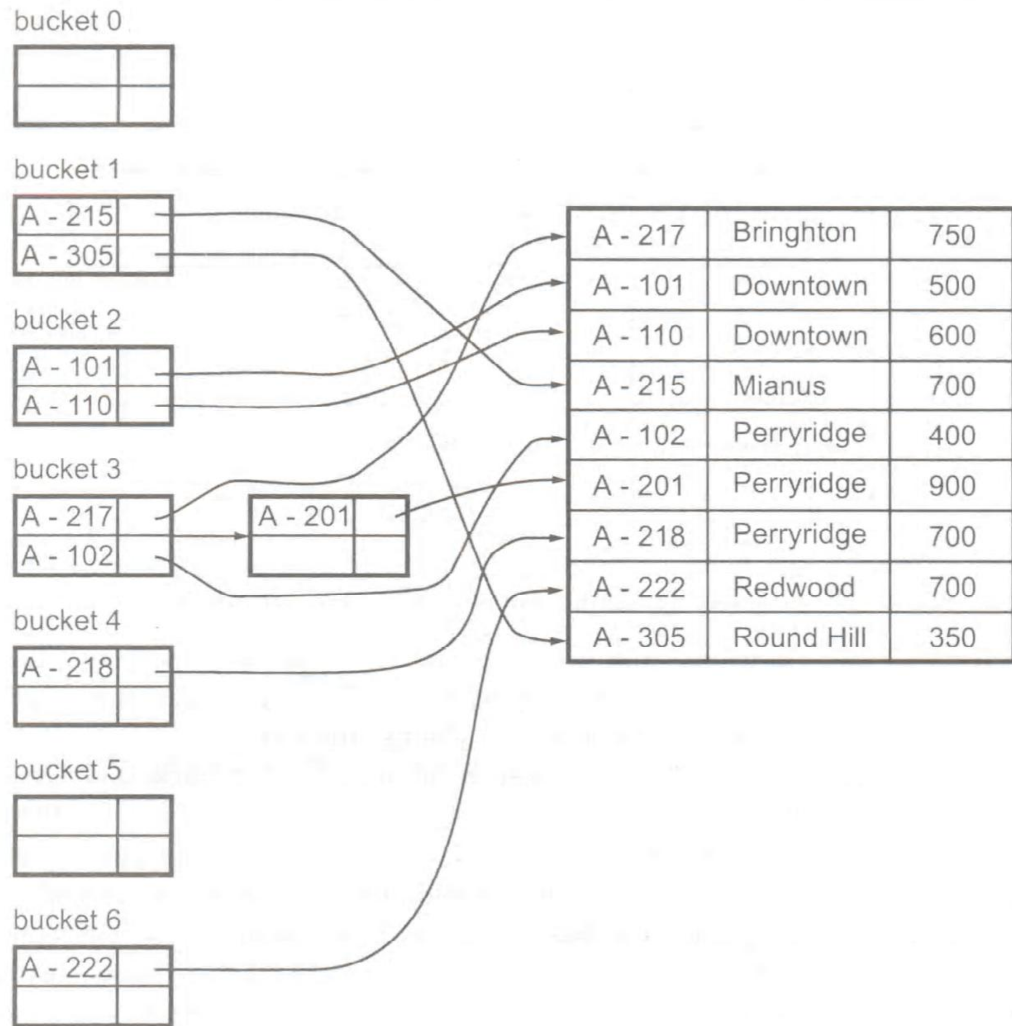
One policy is to use the next bucket in cyclic order that has space. This policy is called linear probing. Open hashing has been used to construct symbol tables for compilers and assemblers, but closed hashing is preferable for database because the deletion under open hashing is troublesome.

Hash indices:

Hashing can be used for index structure creation. A hash index organizes the search keys, with their associated pointers into a hash file structure. A hash index is constructed as follows.

- Apply hash function on a search key to identify a bucket.
- Store the key and its associated pointers in the bucket or in overflow buckets.

The fig below shows as secondary hash index on the account file, for a search key account number. The hash function in the fig computes the sum of the digits of the account number module 7. The hash index has 7 buckets, each of size 2. One of the buckets has 3 keys mapped to it, so it has an overflow bucket.



Dynamic hashing:

Most databases grow longer overtime. If we use static hashing for such databases then we have 3 options:

- Choose a hash function based on the current file size. This option will result in performance degradation as the database grows.
- Choose a hash function based on the anticipated size of the file at some point in the future. Although performance degradation is avoided, a significant amount of space may be wasted initially.
- Periodically reorganize the hash structure in response to the file growth. Such reorganization involves choosing a new hash function, recomputing the hash function on every record in the file, and generating new bucket assignments. This reorganization is massive and time consuming operation.

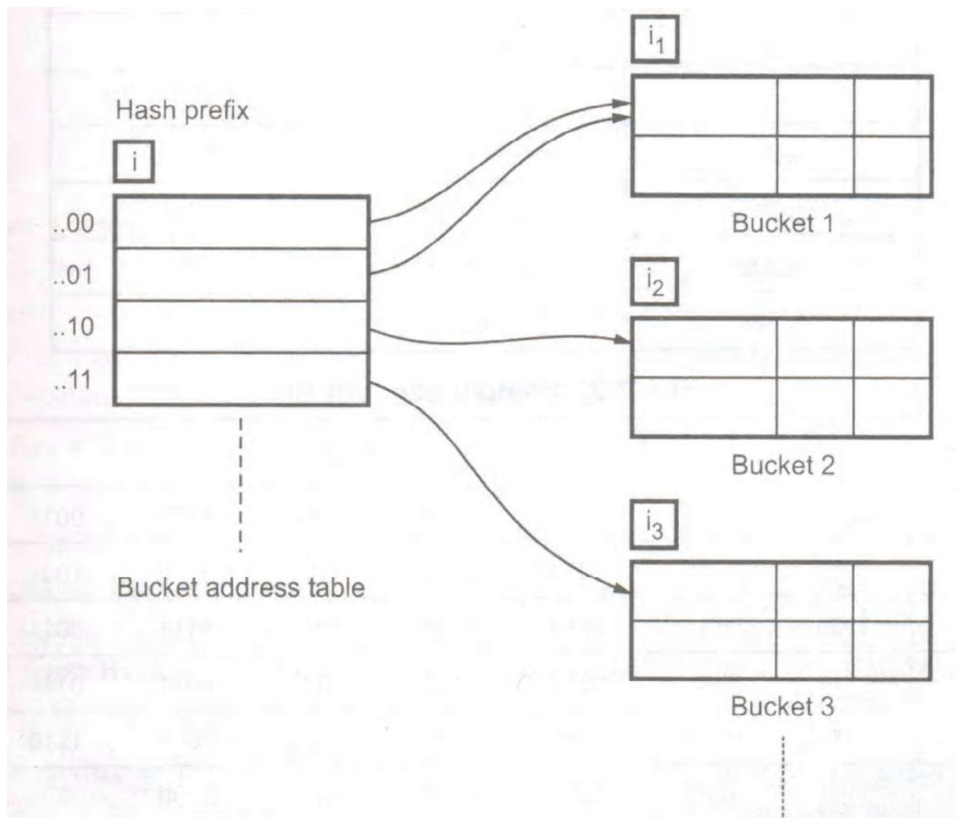
These problems can be avoided using dynamic hashing. Dynamic hashing technique allows the hash function to be modified dynamically to accommodate the

growth or shrinkage of the database. One of the forms of dynamic hashing is extendable hashing.

Extendable hashing:

It handles the changes in the database size by splitting and coalescing buckets as the database grows and shrinks. As a result space efficiency is retained. Since the reorganization is performed on only one bucket at a time, the resulting performance overhead is acceptably low.

With extendable hashing, the hash function h is chosen with desirable properties of uniformity and randomness. We do not create a bucket for each hash value instead buckets are created on demand, as records are inserted into the file. The entire b bits of the hash value are not used initially. At any point we use i bits, where $0 < i <= b$. These i bits are used as an offset into an additional table of bucket addresses. The value of i grows and shrinks with the size of the database.



The fig shows extendable hash structure. The i appearing above the bucket address table indicates I bits of the hash value h (k) are required to determine the correct bucket for K. this number will change as the file grows.

Advantages:

- The performance does not degrade as the file grows.
- It saves space.

Disadvantage:

- Lookup involves an additional level of indirection, since the system access the bucket address table before accessing the bucket itself.
- Implementation of extendable hashing involves additional complexity.

B + tree index files:

The main disadvantage of the index sequential file organization is that performance degrades as the file grows, both for index lookups and for sequential scans through the data.

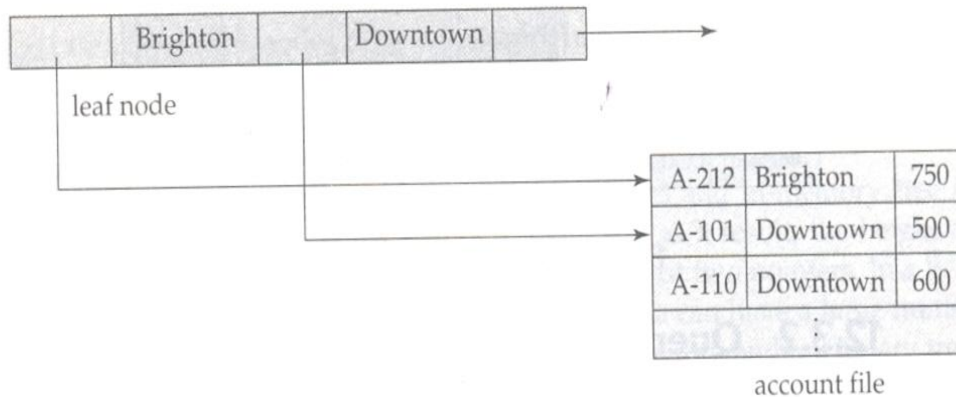
The B+ tree index structure is the most widely used for several index structures that maintain their efficiency despite insertion and deletion of data. A B+ tree index structure takes a form of balanced tree in which every path from the root of the tree to a leaf of the tree is of the same length. Each of non leaf node in the tree has between [n/2] and n children where n is fixed for a particular tree.

Structure of a B+ - Tree:

Typical node of B+ tree is shown below. It contains up to n-1 search key values K1, K2…Kn-1 and n pointes P1, P2 ….Pn. the search key values within a node are kept in sorted order.

P1	K1	P2	K2	……..	Pn-1	Kn-1	Pn

The figure below shows leaf node of a B+ tree for an account file, in which we have chosen n to be 3, and the search key is branch name. Since the account file is ordered by branch name, the pointers in the leaf node points directly to the file.



Each leaf can hold upto n-1 values. We allow leaf nodes to contain as few as [(n-1)/2] values. The range of values in each leaf does not overlap. Thus if Li and Lj are leaf nodes and i<j, then every search key value in Li is less than every search key value in Lj. If a B+ index is to be a dense index, every search key value must appear in some leaf node. Since there is a linear order on the leaves based on the search key values that they

contain, we use Pn to chain together the leaf nodes in search key order. This ordering allow for efficient sequential processing of the file.

The non-leaf nodes of the B+ tree form a multilevel (sparse index) on the leaf nodes. A non-leaf node may hold upto n pointers, and must hold atleast [n/2] pointers. The number of pointers in a node is called the fan-out of the node.

The root node can hold fewer than [n/2] pointers. However it must hold atleast two pointers, unless the tree consists of only one node. It is always possible to construct a B+ tree for any n, that satisfies the preceding requirements. The fig below shows a complete B+ tree for an account n=3.

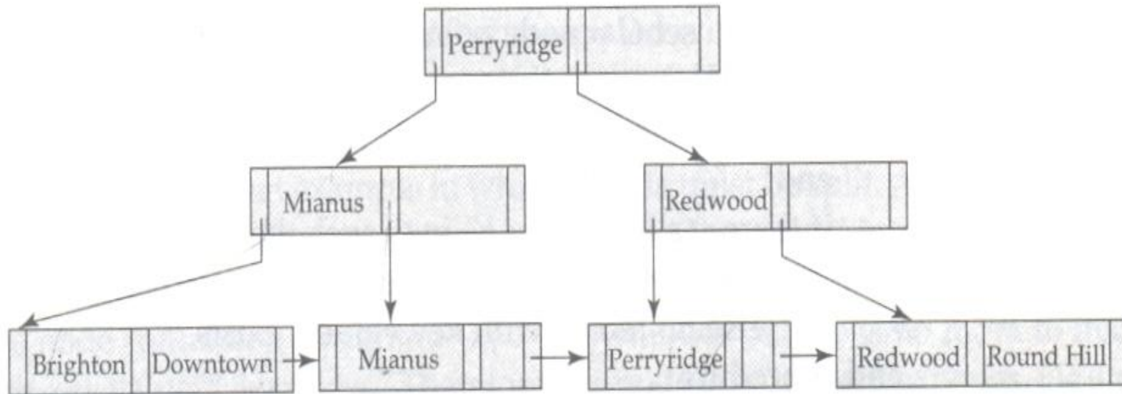


Figure 12.8 B^+-tree for *account* file ($n = 3$).

The fig below shows a B+ tree for n=5. These examples of B+ tree are all balanced. That is the length of every path from the root to a leaf node is the same. This is a requirement for a B+ tree that ensures good performance for lookup, insertion and deletion.

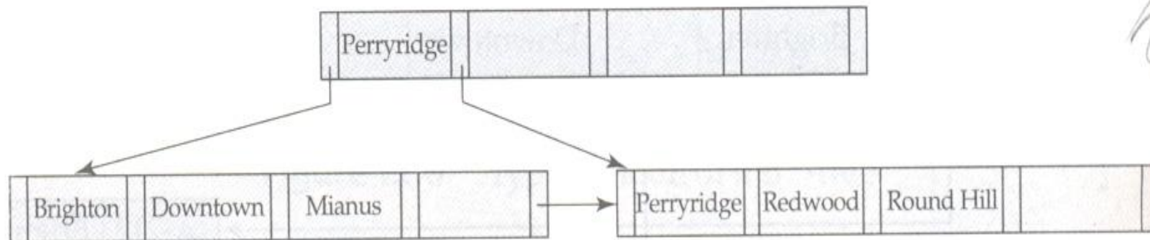
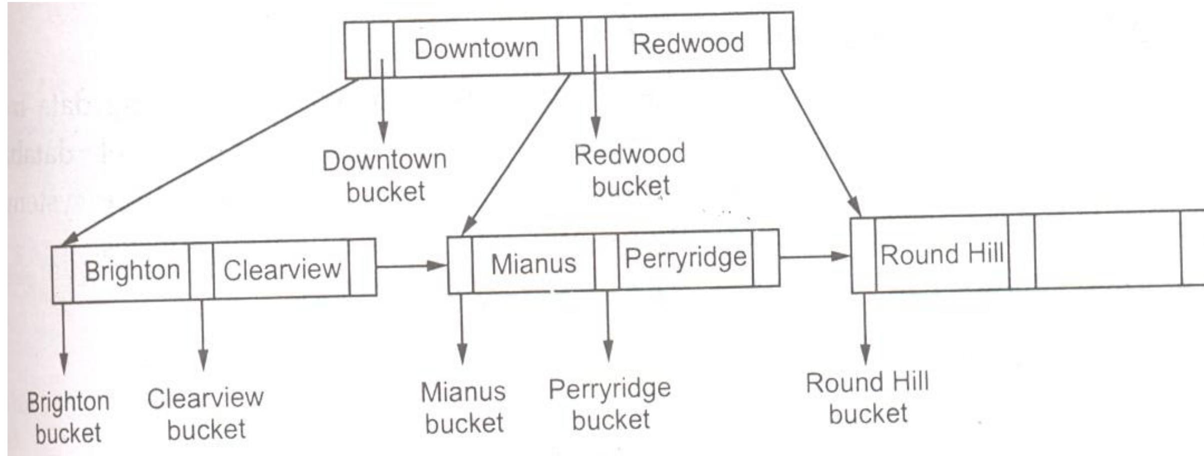


Figure 12.9 B^+-tree for *account* file with $n = 5$.

B trees index files:

B tree indices are similar to B+ tree indices. The primary distinction between the two approaches is that a B- tree eliminates the redundant storage of search key values. In the B+ tree shown in the fig 12.8 , the search keys “downtown”, “Mianus”, ”Redwood” and “Perryridge” appear twice.

A B-tree allows search key values to appear only once. Since search keys that appear in non leaf nodes appear nowhere else in the B tree, we are forced to include an additional pointer field for each search key in a non-leaf node. These additional pointers point to either file records or buckets for the associated search-key. A generalized B-tree leaf node appears in the fig. a non leaf node appears in the fig below.



The number of nodes accessed in a lookup in a B tree depends on where the search key is located. A lookup on the B+ tree requires traversal of a path from the root of the tree to some leaf node. In contrast, it is sometimes possible to find the desired value in a B tree before reaching a leaf node. The fact that fewer search keys appear in a non-leaf B tree node compared to B+ tree node implies that the B- tree has a smaller fan-out and therefore may have depth greater than that of the corresponding B+ tree. Thus lookup in a B-tree is faster for some search keys but slower for others.

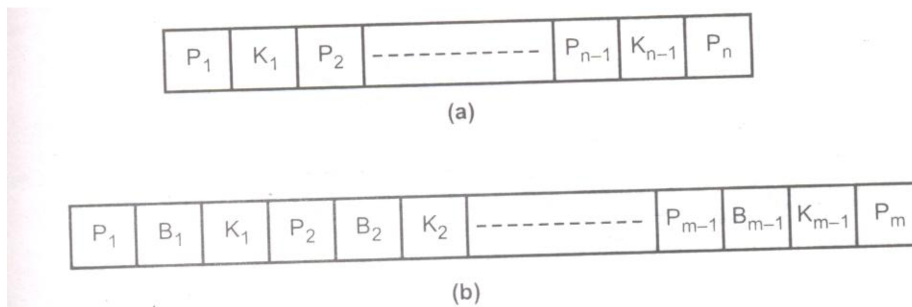


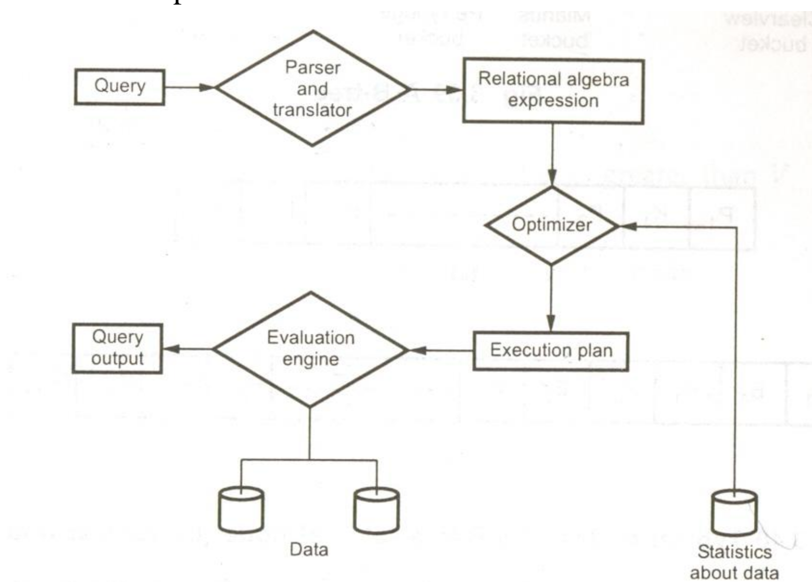
Fig. 3.40 Typical nodes of a B-tree (a) Leaf node (b) Nonleaf node

Deletion in B-trees is more complicated. In a B+ tree, the deleted entry always appears in a leaf. In a B+ tree, the deleted entry may appear in a non-leaf node. The proper value must be selected as a replacement from the sub-tree of the node containing the deleted entry. Specifically if search key Ki is deleted, the smallest search key appearing in the sub-tree of pointer Pi+1 must be removed to the field formerly occupied by Ki. Further actions need to be taken if the leaf node now has too few entries. In contrast, insertion in a B-tree is only slightly more complicated than is insertion in a B+ tree.

Query Processing

Overview of Query Processing

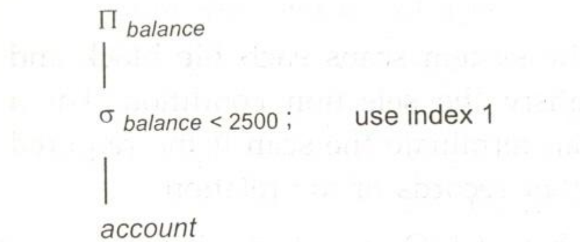
- Query processing refers to the range of activities involved in extracting data from a database.
- The activities include translation of queries in high-level database languages; into expressions that can be used tat the physical level of the file system, a variety of query – optimizing transformations, and actual evaluation of queries.
- The steps involved in processing a query appear in Fig 3.41
- The basic steps are:



1) Parsing and translation
2) Optimization
3) Evaluation.

- The first action, the system must take in query processing is to translate a given query into its internal form. This translation process is similar to the work performed by the parser of a compiler.
- In generating the internal form of the query, the parser checks the systax of the user's query, verifies that the relation names appearing in the query are names of the relations in the database, and so on.
- The system constructs a parse – tree representation of the query, which it then translates into a relational – algebra expression.

- Furthermore, the relations – algebra representation of a query specifies only partially how to evaluate a query.
- As a illustration, consider the query:
    - Select balance
    - From account
    - Where balance < 2500
- This query can be translated into either of the following relational – algebra expressions:
    - Balance < 2500 (Π balance (account))
    - Balance (σ balance < 2500 (account))
- To implement the preceding selection, we can search every tuple in account to find tuples with balance less than 2500.
- If a B+ -tree index is available on the attribute balance, we can use the index instead to locate the tuples.
- To specify fully how to evaluate a query, we need to provide not only the relational – algebra expression, but also to annotate it with instructions specifying how to evaluate each operation.



- A relational – algebra operation annotated with instructions on how to evaluate it is called an evaluation primitive.
- A sequence of primitive operations that can be used to evaluate a query is a query – execution plan or query – evaluation plan.
- Fig 3.42 illustrates an evaluation plan for our example query, in which a particular index (denoted in the fig. as “index 1”) is specified for the selection operation.
- The query – execution engine takes a query – evaluation plan, executes that plan, and returns the answers to the query.
- The different evaluation plans for a give query can have different costs.

Measures of Query Cost:

- The cost of query evaluation can be measured in terms of a number of different resources, including disk accesses, CPU time to execute a query, and, in distributed or parallel database system, the cost of communication.
- The response time for a query – evaluation plan (that is, the clock time required to execute the plan) could be used as a good measure of the cost of the plan.
- In large database systems, however, disk accesses are usually the most important cost, since disk accesses are slow compared to in – memory operations.
- Most people consider the disk – access cost a reasonable measure of the cost of a query – evaluation plan.
- The number of block transfers from disk is also used as a measure of the actual cost.
- We also need to distinguish between reads and writes of blocks, since it takes more time to write a block to disk than to read a block from disk.
- For more accurate measure find out:
  1) The number of seek operations performed,
  2) The number of blocks read,
  3) The number of blocks written,

And then add up these numbers after multiplying them by the average seek time, average transfer time for reading a block, and average transfer time for writing a block, respectively.

Selection Operation

- Consider a selection operation on a relation whose tuples are stored together in one file.
- Two scan algorithms to implement the selection operation are given below.

Basic Algorithms

- A1 (linear search) : In a linear search, the system scans each file block and tests all records to see whether they satisfy the selection condition. For a selection on a key attribute, the system can terminate the scan if the required record is found, without looking at the other records of the relation.
- The cost of linear search, in terms of number of I/O operations, is b_r , where b_r denotes the number of blocks in the files. Selection on key attributes have an average cost of $b_r/2$ but still have a worst – case cost of b_r.

Advantages:

- The linear search algorithms can be applied to any file.

Disadvantage:

- It is slower.
- A2 (binary search) : If the file is ordered on an attribute, and the selection condition is an equality comparision on the attribute, we can use a binary search to locate records that satisfy the selection.
- The system performs the binary search on the blocks of the file.
- The number of blocks that need to be examined to find a block containing the required records is $(\log_2 (b_r))$, where b_r, denotes the number of blocks in the file .
- If the selection is on a nonkey attribute, more than one block may contain required records, and the cost of reading the extra blocks has to be added to the cost estimate.

Selection using Indices

- Search algorithms that use an index are referred to as index scans.
- Search algorithms that use in index are:
- A3 (primary index, equality on key): For an equality comparision on a key attribute with a primary index, we can use the index to retrieve a single record that satisfies the corresponding equality condition.
- If a B+ -tree is used, the cost of the operation in terms of I/O operations, is equal to the height of the tree plus one I/O to fetch the record.
- A4 (Primary index, equality on nonkey) : We can retrieve multiple records by using a primary index when the selection condition specifies an equality comparision on a nonkey attribute, A. The cost of the operation is proportional to the height of the tree, plus the number of blocks containing records with the specified search key.
- A5 (Secondary index, equality): Selection specifying an equality condition can use a secondary index. This strategy can retrieve a single record if the equality condition is on a key; multiple records may get retrieved if the indexing field is not a key.
- In the first case, only one record is retrieved, and the cost is equal to the height of the tree plus one I/O operation to fetch the record. In the second case, each record may be resident on a different block, which may result in one I/O operation per retrieved record.

Selections Involving Comparisions

- Consider a selection of the form σ A < v(Γ).
- We can implement the selection either by using a linear or binary search or by using indices in one of the following ways:
- A6 (Primary index, comparison): A primary ordered index (for example, a primary B+ - tree index) can be used when the selection condition is a comparision.
- For comparison conditions of the form A>v or A> , a primary index on A can be used to direct the retrival of tuples, as follows.
- For A>v, we look up the value v in the index to find the first tuple in the file that has a value of A = v.
- A file scan starting from that tuple up to the end of the file returns all tuples that satisfy the condition.
- For A > v, the file scan starts with the first tuple such that A>v.
- For comparisions of the form A < v or A <v, an index lookup is not required.
- For A < v, we use a simple file scan starting from the beginning of the file, and continuing up to the first tuple with attribute A = v.
- The case A < v is similar, except the scan continues up to the first tuple with attribute A>v.
- In either case index is not useful
- A7 (Secondary index, comparison) : We can use a secondary ordered index to guide retrivel for comparision conditions involving <, <,>, or >.
- The lowest – level index blocks are scanned, either from the smallest value up to v(for < and <), or from v up to the maximum va;ue (for > and >).

Implementation of complex selections

- We now consider more complex selection predicates.
- Conjunction: A conjuctive selection is a selection of the form
- Disjunction: A disjunctive selection is a selection of a form
- Negation : The result of selection σ θ® is the set of tuples of r for which the condition θ evaluates to false.
- We can implement a selection operation involving either a conjunction or a disjunction of simple conditions by using one of the following algorithms:

- A8 (Conjunctive selection using one index) : We first determine whether an access path is available for an attribute in one of the simple conditions.
- If one is, one of the selection algorithms A2 through A7 can retrieve records satisfying that condition.
- A9 (Conjunctive selection using composite index). An appropriate composite index (that is, an index on multiple attribute) may be available for some conjunctive selections.
- If the selection specifies an equality condition on two or more attributes, and a composite index exists on these combined attribute fields, then the index can be searched directly. The type of index determines which of algorithms A3, A4 or A5 will be used.
- A10 (Conjunctive selection by intersection of identifiers): Another alternative for implementing conjunctive selection operations involves the use or record pointers or record identifiers.
- This algorithm requires indices with record pointers, on the fields involved in the individual conditions.
- The algorithm scans each index for pointers to tuples that satisfy an individual condition. The interection of all the retrieved pointers is the set of pointers to tuples that satisfy the conjunctive condition.
- The algorithm then uses the pointers to retrieve the actual records.
- The cost of algorithm A10 is the sum of the costs of the individual index scans, plus the cost of retrieving the records in the interaction of the retrieved lists of pointers.
- This cost can be reduced by sorting the list of pointers and retrieving records in the sorted order. Thereby,
  1) All pointers to records in a block come together, hence all selected records in the block can be retrieved using a single I/O operation, and
  2) Blocks are read in sorted order, minimizing disk arm movement.
- A11 (Disjunctive selection by union of identifiers) : If access path are available on all the conditions of a dinjunctive selection, each index is scanned for pointers to tuples that satisfy the individual condition. The union of all the retrieved pointers yields the set of pointers to all tuples that satisfy the disjunctive condition.
- We then use the pointers to retrieve the actual records.

- However, if even one of the conditions does not have an access path, we will have to perform a linear scan of the relation to find tuples that satisfy the condition.

Sorting

- We may build an index on the relation, and then use the index to read the relation in sorted order. May lead to one disk block access for each tuple.
- For relations that fit in memory, techniques like quick sort can be used. For relations that don't fit in memory, external sort-merge is a good choice.

External Sort-Merge

Let M denotes the memory size

1. Create sorted runs
   Let i be 0 initially
   Repeatedly do the following till the end of the relations
   a) Read M blocks of relation into memory
   b) Sort the in memory blocks
   c) Write sorted data to run R_i and increment i.
      Let the final value of i be N.
2. Merge the runs(N-ways merge)
   Assume that N<M
   1. Use N blocks of memory to buffer input runs, and 1 block to buffer output. Read the first block of each run into its buffer page.
   2. Repeat
      i) Select the 1st record among all buffer pages.
      ii) Write the record to the output buffer. If the output buffer is full write it to disk.
      iii) Delete the record from its input buffer page

If the buffer page becomes empty then read the next block(if any)of the run into the buffer.

3.Until all input buffer pages are empty.

>If, several merge passes are required.

a. In each pairs, contiguous groups of M-1 runs are merged.

b. A pass reduces the number of runs by a factor of M-1, and creates runs longer by the same factor.

For eg, if M=11, and there are 90 runs, one pass reduces the number of runs to 9, each 10 times the size of the initial runs.Repeated passes are performed till all runs have been merged into one.

Example:External sorting using sort-merge.

Join Operation

- In this section, we study several algorithms for computing the join of relations, and we analyze their respective costs
- Consider the expression:
    - Depositor customer
- We assume the following information about the two relations:
- Number of records of customer: n customer=10,000
- Number of blocks of customer : b customer = 400
- Number of records of depositor: n depositor = 5000
- Number of blocks of depositor: b depositor = 100

Nested – Loop Join

- This algorithm is called the nested – join algorithm, since it basically consists of a pair of nested for loops.
- Relation r is called the outer relation and relation s is called inner relation of the join
- The algorithm uses the notation tr.ts, where tr and ts are tuples; tr . ts denotes the tuple constructed by concatenation the attribute values of tuples tr and ts.
- The nested – loop algorithm requires no indices.
- It is expensive, since it examines every pair of tuples in the two relation.
- Consider the cost of the nested – loop join algorithm. The number of pairs of tuples to be consider is nr * ns, where nr, denotes the number of tuples in r,and ns denotes the number of tuples in s. For each record in r, we have to perform a complete scan on s.
- In the worst case, the buffer can hold only one block of each relation, and a total of nr * bs + br block accesses would be required, where br and bs denote the number of blocks containing tuples of r and s respectively.
- Now consider the natural join of depositor and customer.
- Assume that depositor is the outer relation and customer is the relation in the join.
- We will have to examine 5000 * 10000 = 50 * 10 6 pairs of tuples.
- In the worst case, the number of block accesses is 5000 * 400 + 100 = 2,000,100.

- In the best – case scenario, however, we can read both relations only once, and perform the computation.
- This computation requires at most 100 + 400 = 500 block accesses.

```
For each tuple tr in r do begin
    For each tuple ts in s do begin
            Test pair(tr,ts) to see if they satisfy the join condition θ
                    If they do, add tr.ts to the result
    End
End
```

Block Nested – Loop Join

- Block nested – loop join, which is a variant of the nested – loop join which every block of the inner relation is paired with every block of the outer relation.
- Within each pair of block, every tuple in one block is paired with every tuple in the other block, to generate all pairs of tuples.
- As before, all pairs of tuples that satisfy the join condition are added to the result.

```
For each block Br of r do begin
    For each block Bs of s do begin
            For each tuple tr in Br do begin
                    For each tuple ts in Bs do begin
                            Test pair (tr, ts) to see if they satisfy the join condition
                                    If they do, add tr . ts to the result
                            End
                    End
            End
    End
```

- The primary difference in cost between the block nested – loop join and the basic nested – loop join is that, in the worst case, each block in the inner relation s is read only once for each block in the outer relation, instead of ounce for each tuple in the outer relation.
- Thus, in the worst case, there will be a total of br * bs + br block accesses, where br and bs denote the number of blocks containing records of r and s respectively.
- In the worst case, we have to read each block of customer once for each block of depositor
- Thus, in the worst case, a total of 100 * 400 +100=40,100 block accesses are required.
- The number of block accesses in the best case remains the same namely, 100 + 400 = 500.

- The performance of the nested loop and block nested – loop procedures can be further improved:
    - o If the join attributes in a natural join or an equi – join form a key on the inner relation, then for each outer relation tuple the inner loop can terminate as soon as the first match is found.
    - o If memory has M blocks, we read in M – 2 blocks of the outer relation at a time, and when we read each block of the inner relation we join it with all the M – 2 blocks of the outer relation. This change reduces the number of scans of the inner relation from br to[br/ M -2) where br is the number of blocks of the outer relation. The total cost is then [br (M-2)]*bs+br.
    - o Reuse the blocks stored buffer, which reduces the number of disk accesses needed.
    - o If an index is available on the inner loop's join attribute, we can replace file scans with more efficient index lookups.

Indexed Nested - Loop Join

- In a nested – loop join, if an index is available on the inner loop's join attribute, index lookups can replace file scans.
- For each tuple tr in the outer relation r, the index is used to look up tuples in s will satisfy the join condition with tuple tr.
- This join method is called an indexed nested – loop join; it can be used with existing indices.
- For example, consider depositor customer.
- Suppose that we have a depositor tuple with customer – name "John".
- Then the relevant tuples in s are those that satisfy the selection "customer – name=John"
- The cost of an indexed nested – loop join can be computed as follows: for each tuple in the outer relation r a lookup is performed on the index for s, and the relevant tuples are retrieved.
- Then br disk accesses are needed to read relation r, where br denoted the number of block containing records of r.
- For each tuple in r, we perform an index lookup on s.
- Then, the cost of the join can be computed as br+nr * c, where br is the number of records in relation r, and c is the cost of a single selection on s using the join condition.
- For example, consider an indexed nested-loop join of depositor customer , with depositor as the outer relation.

- Suppose also that customer has a primary B^+-tree index on the join attribute customer_name, which contains 20 entries on an average in each index node.
- Since customer has 10,000 tuples, the height of the tree is 4, and one more access in needed to find the actual data.
- Since n depositor is 5000, the total cost is 100 + 5000 *5=25,100 disk accesses.
- This cost is lower than the 40,100 accesses needed for a block nested – loop join.

Merge join

- The merge join algorithm (also called the sort – merge join algorithm) can be used to compute natural join and equi-joins.
- Let r (R) and s (S) be the relations whose natural join is to be computed, and let R S denote their common attributes.

```
Pr:=address of first tuple of r;
Pr:=address of first tuple of s;
While (ps =null and pr=null ) do
Begin
        Ts:=tuple to which ps points;
        Ss:={ts};
        Set ps to point to net tuple of s;
        Done:=false
        While(not done and ps = null) do
        Begin
        Ts:=tuple to which ps points;
        If (ts[Join Attrs]=ts[Join Attrs])
        Then begin
        Ss:=Ss U {t`s};
        Set ps to point to next tuple of s;
        End
Else
        Done:=true;
End
tr:= tuple to which pr points;
while (pr null and tr[Join Attrs] < ts[Join Attrs]) do
begin
        set pr to point to next tuple of r;
        tr:=tuple to which pr points;
end
```

```
        while(pr null and tr[Join Attrs]=ts[Join Attrs]) do
        begin
                for each ts in Ss do
                begin
                        add ts  tr to result;
                end
        set pr to point to next tuple of r;
        tr:=tuple to which pr points;
        end
end
```

Merge Join

- In the algorithm, Join Attrs refers to the attributes in R n S, and t_r $t_{s`}$ where t_r and t_s are tuples that have the same values for JoinAttrs, denotes the concatenation of the attributes of the tuples, followed by projecting our repeated attributes.
- The merge join algorithm associates one pointer with each relation.
- These pointers point initially to the first tuple of the respective relation.
- As the algorithm proceeds, the pointer moves through the relation. A group of tuples of one relation with the same value on the join attributes is read into S_s.
- The algorithm requires that every set of tuples S_s fit in memory.
- If the relations are in sorted order, the number of block accesses is equal to the sum of the number of blocks in both files, $b_r + b_{s`}$.

Hash Join

- The hash join algorithm can be used to implement natural joins and equi – joins.
- In the hash join algorithm, a hash function h is used to partition tuples of both relations.
- The basic idea is to partition the tuples of each of the relations into sets that have the same hash value on the join attributes
- We assume that
  - It is a hash function mapping JoinAttrs values to {0,1,….nh}, where JoinAttrs denotes the common attributes of r and s used in natural join.
  - H_r, denote partitions of r tuples, each initially empty. Each tuple t_r £ r is put in partition, where i=h(t_r[JoinAttrs]).
  - H_s, denote partitions of s tuples, each initially empty. Each tuple t_s s is put in partition, where i=h(t_s[JoinAttrs]).
- The idea behind the hash join algorithm is this: Suppose that a r tuple a s tuple satisfy the join condition; then, they will have the same value for the join attributes.

- If that value is hashed to some value i, the r tuple has to be in H_{ri} and the s tuple in H_{si}.
- Therefore, r tuples in H_{ri} need only to be compared with s tuples in H_{si}; they do not need to be compared with s tuples in any other partition.
- For example, if d is a tuple in depositor, c is a tuple in customer, and h is a hash function on the customer-name attribute of the tuples, then d and c must be tested only if h(c)= h(d), we must test c and d to see whether the values in their join attributes are the same, since it is possible that c and d have different customer – names that have the same hash value.
- Fig 3.46 shows the details of the hash join algorithm to compute the natural join of relations r and s.

```
/* Partition s */
For each tuple ts in s do begin
        I:= h(ts [JoinAttrs])
        His:= His U {ts};

End
/* Partition r* /
For each tuple tr in r do begin
        I:= h(tr[JoinAttrs])
        Hri:=HRi U {tr};
End
/* Perform join on each partition */
For i=0 to nn do begin
        Read H si and build an in – memory hash index on it
        For each tuple tr in Hri do begin
        Probe the hash index on H si to locate all tuples ts
        Such that ts [JoinAttrs] = tr [JoinAttrs]
        For each matching tuple t s in H si do begin
                Add tr    ts to the result
        End
End
End.
```

Database tuning

- It describes a group of activities used to optimize and homogenize the performance of a database. It usually overlaps with query tuning, but refers to design of the database files, selection of the database management system (DBMS), operating system and CPU the DBMS runs on.
- The goal is to maximize use of system resources to perform work as efficiently and rapidly as possible. Most systems are designed to manage work efficiently, but it is possible to greatly improve performance by customizing settings and the configuration for the database and the DBMS being tuned.

I/O tuning

- I/O tuning is placing database transaction logs, files associated with temporary work spaces, and table and index file storage to optimize and balance reads and writes against these files. I/O is generally the most expensive operation in database work, and is typically the first bottleneck in database performance encountered.
- Hardware and software configuration of disk subsystems are examined: RAID levels and configuration, block and stripe size allocation, and the configuration of disks, controller cards, storage cabinets, and external storage systems such as a SAN. Transaction logs and temporary spaces are heavy consumers of I/O, and affect performance for all users of the database. Placing them appropriately is crucial.
- Frequently joined tables and indexes are placed so that as they are requested from file storage, they can be retrieved in parallel from separate disks simultaneously. Frequently accessed tables and indexes are placed on separate disks to balance I/O and prevent read queuing.

DBMS tuning

- DBMS tuning refers to tuning of the DBMS and the configuration of the memory and processing resources of the computer running the DBMS. This is typically done through configuring the DBMS, but the resources involved are shared with the host system.
- Tuning the DBMS can involve setting the recovery interval (time needed to restore the state of data to a particular point in time), assigning parallelism (the breaking up of work from a single query into tasks assigned to different processing resources), and network protocols used to communicate with database consumers.
- Memory is allocated for data, execution plans, procedure cache, and work space. It is much faster to access data in memory than data on storage, so maintaining a sizable cache of data makes activities perform faster.
- The same consideration is given to work space. Caching execution plans and procedures means that they are reused instead of recompiled when needed. It is important to take as much memory as possible, while leaving enough for other processes and the OS to use without excessive paging of memory to storage.
- Processing resources are sometimes assigned to specific activities to improve concurrency. On a server with eight processors, six could be reserved for the DBMS to maximize available processing resources for the database.

Question Bank

2 MARK QUESTIONS

1. Differentiate between Volatile and Non Volatile Storage.
2. Define Disk Pack.
3. Define the terms Seek Time and Latency.
4. What is ordering key?
5. When the Overflow of file occurs?
6. What is the difference between file organization and Access Methods?
7. Discuss the Mechanism used for read the data from or write the data to disk.
8. What are the differences between Static File and Dynamic File?
9. What is the use of Mixed File?
10. What is the Technique used for allowing a Hash file to expand and Shrink
11. Dynamically?
12. Why are the Disks, tapes used to store for Online- Database Files?
13. Define the term INTERBLOCK GAP?
14. How does the Mirroring Helps in improving Reliability?
15. Define Internal and External Hashing with an Examples?
16. Define the term BLOCK ANCHOR & DENSE INDEX?
17. Define the following terms:
    (a) Indexing Field,
    (b) Primary key
    (c) Clustering field
18. How does the Multilevel Indexing improve the Efficiency of searching an index
19. File?
20. How a B-Tree does differs from W Tree? Explain it.
21. Describe the structure of B-Tree.
22. Describe the structure of B+-Tree.
23. What are the differences between primary/secondary/clustering indexes?
24. What is measures of the quality of a disk?
25. What are the 2 types of ordered indices?

16 MARK QUESTIONS

1. Describe about RAID levels.
2. Describe the structure of B+-tree and give the algorithm for search in the B+-tree
3. with example.
4. Explain the comparison between ordered indexing and hashing.
5. Explain in brief about query processing.
6. Compare and contrast B+- and B--tree. Explain it with neat sketch.
7. List the secondary storages devices. Explain each in detail.
8. Explain the index structure for files and its types in detail.
9. Explain different hashing methods in detail.
10. List the various types of indices. Explain each in detail.
11. Explain different file organizations.