

Advanced Database (ADBMS)

By

Bhupendra Singh Saud

For B.Sc. CS/IT 7th Semester TU

Course of Contain

Unit 1: The Relational Model of Data and RDBMS Implementation Techniques [5 Hrs.]

Theoretical concepts, Relational model conformity and Integrity, Advanced SQL programming, Query optimization, Concurrency control and Transaction management, Database performance tuning, Distributed relational systems and Data Replication, Security considerations.

Unit 2: The Extended Entity Relationship Model and Object Model [6 Hrs.]

The ER model revisited, Motivation for complex data types, User defined abstract data types and structured types, Subclasses, Super classes, Inheritance, Specialization and Generalization, Constraints and Characteristics of specialization and Generalization, Relationship types of degree higher than two, Relational database design by EER- to relational mapping, basic concepts on UML.

Unit 3: Emerging Database Management System Technologies [18 Hrs.]

Object Oriented Database concepts: object identity, structure, and type constructors; encapsulation of operations, methods, and persistence; type and class hierarchies and inheritance; structures and unstructured complex objects and type extensibility; polymorphism, multiple inheritance and selective inheritance, versions and configurations; Object Relational Database concepts: overview of SQL and its object-relational features (the SQL standard and its components, object-relational support in SQL-99); evolution and current trends of database technology (with respect to the features of the Informix Universal Server and Oracle8); implementation and related issues for extended type systems; the nested relational model; Active database concepts: Generalized model for active databases and oracle Triggers; design and implement issues for active databases potential applications for active databases; Temporal database concepts: Time representation, calendars, and time dimensions; incorporating time in relational databases using tuple versioning, incorporating time in object-oriented databases using attribute versioning, time series data; Multimedia Databases: The nature of multimedia data and applications; spatial database concepts and architecture, introduction to multimedia database concepts; Deductive databases and Query

processing: Prolog/Data log notations, clausal form and horn clauses; interpretations of rules; Mobile Databases: Mobile computing architecture, characteristics of mobile environments, data management issues; Geographic Information Systems: GIS applications, data management requirements of GIS, specific GIS data operations;

Unit 4: New database applications and environments [8 Hrs.]

Data Mining: Overview of data mining technology (associated rules, classification, clustering), applications of data mining; Data Warehousing: Overview of data warehousing, typical functionality of a data warehouse;

Unit 5: Database Related Standards [8 Hrs.]

SQL standards, SQL 1999, SQL 2003, Object Data Management Group (ODMG) version 3.0 standards (ODL, OQL), Standards for interoperability and integration e.g. Web Services, SOAP, XML related specifications, e.g. XML Documents, DTD, XML Schema, X-Query, XPath.

Prerequisite:

- Be familiar with at least one OO Programming language such as .Net or C++ or Java,
- Fundamentals of DBMS, SQL

Reference Books:

1. Elmasri and Navathe, Fundamentals of Database Systems, Pearson Education
2. Raghu Ramakrishnan, Johannes Gehrke, Database Management Systems, McGraw-Hill
3. Korth, Silberchatz, Sudarshan, Database Systems, Design, Implementation and Management, Thomson Learning
4. C.J. Date & Longman, Introduction to Database Systems, Pearson Education

Computer Usage:

Windows or Linux based PC or workstation, Commercial OODBMS software package and MVC software development framework installed at the server.

Unit 1

Theoretical Concept of overall database

Data

The collection of raw facts is called data.

File

A file is the collection of related groups of data for example, payroll file of a company consists of the salary detail and all of these records have the same heads (e.g.; basic pay, HRA, FA etc.).

Records

A file may be further divided into more descriptive subdivisions, called, records. In other words a record is a collection of related data items as a single unit. It is also called column of table.

Tuples

The row of a table is called tuples. It is also called value of a table.

Fields

The column of a table is called fields.

Database

A database is a collection of related data necessary to manage an organization. By data, we mean known facts that can be recorded and have implicit meaning. For example, consider names, telephone numbers and addresses of the people. We may have recorded this data in an indexed address book, or we may have recorded on the hard drive, using a personal computer and software such as MS-Access, or Excel. This is the collection of related data with an implicit meaning and hence is a database. A database is logically coherent collection of data with some inherent meaning. A database is designed, built and populated with data for specific purpose. It excludes transient data such as: input documents, reports and intermediate results obtained during processing.

DBMS

A database management system is a set of procedures that manages the database and provide access to the database in the form required by any application program. It effectively ensures that necessary data in the desired form is available for diverse applications for different departments in an organization. A DBMS is hence a general purpose software system that facilitates the processes

of defining, constructing, manipulating, and sharing database among various users and applications. Fig 1 illustrate the difference between database and database management system.

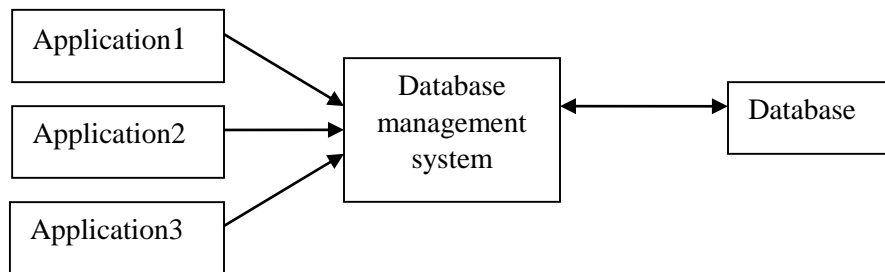


Fig 1 illustrates the distinction between a DBMS and a Database.

A database management system (DBMS) is a collection of programs that control the database. The primary goal of DBMS is store and manages the data conveniently and efficiently.

Why data management is Important?

One of the most valuable resources of any business is the data stored by the business. The efficient management of data is a key business activity. A systematic approach to data management allows data to became information

When data becomes information

When data is organized in a meaningful way, called information that may not have been easily observable becomes apparent for example total sales buying trends for February 2002 etc.

- Effective data management enables raw data to become useful information.
- Data + semantics(Rules) = information
- Any random collection of data is not a database.

Example uses of DBMS

- Airlines – reservations and schedules etc.
- Banking – customer information and accounts etc.
- Universities – student information, grades etc.
- Government – taxes, budgets, census etc.
- Sales – inventory, customer info etc.
- Credit cards – transactions, customer info etc.
- Newspapers – track subscribers, advertising revenue.
- Financial – stock prices, portfolio info.
- Telecommunications – records of calls made.

History

- Integrated data store, first general purpose DBMS, early 1960s, Charles Bachman, general electrics.
- Information management system (IMS), late 1960s, IBM.
- Relational database model, proposed in 1970 Edgar Codd, IBM's san Jose laboratory.
- Structured Query language (SQL) standardized in the late 1980s.
- More powerful query language, complex analysis of data, support for new data types late 1980s to 1990s.
- Packages which came with power customizable application layers.
- Internet.

Objectives of Database Management

1. Sharing of data

The main motivation for designing a system to manage a database in shareability of data. Data is considered as an important resource of an organization to be shared by a number of application .It gives the sharing of data.

2. Data integrity

The term integrity means to the validity of data contained in database. Data integrity can be reduced in many ways, including input typing errors, hardware malfunctions and data transmission errors. To avoid data integrity errors, database program should use validation procedures, which define acceptable input ranges for each field in the record.

3. Data independence

The term data independence refers to the storage of data in such way that it is not locked into use by the particular application. To ensure data independence, it is important to avoid software that uses file storage techniques.

4. Avoid data redundancy

The system should identify existence of common data and avoid duplicate recording. The process of avoiding duplicate data is called data redundancy. Selective redundancy is sometimes allowed to improve performance or for better reliability.

5. Data Security

This is concerned with protecting access to data. Protection is needed at many levels for access, modification, deletion or access.

6. Data maintenance

Good database management involves having system in place for data maintenance. This system includes procedure for adding records, updating records and deleting records.

7. Management control

As the dependence of an organization on the database increase, positive management controls should be exercised over addition, deletion, change and disposition of data. Data must be protected to satisfy legal, accounting and auditing requirements.

Purpose of database systems

Using Database system, we can easily insert, delete and update data from the database. We can easily retrieve required result. In a file processing system, permanent records are stored in various files, and different application programs are written to extract records from, and to add records to, the appropriate files. Before DBMS system, organization used to store information using file processing system. The main purpose of database is to remove the difficulties of file processing system, which are given below.

Drawbacks of using file systems to store data

1. Data redundancy and inconsistency

The few same data files are often included in many different files, in file processing system. The redundancy in storing the same data multiple time leads to several problems. First, there is the need to perform a single logical update- such as entering data on a new student multiple time: once for each file where student data is recorded. This leads to duplication of effort. Second, storage space is wastage when the same data is stored repeatedly and this problem may be serious for large database. Third files that represent the same data may become inconsistent. This may happen because an update is applied to some of the files but not to others. Even if an update adding a new student is applied to all the appropriate files, the data concerning the student may still be inconsistent because the updates are applied independently by each user group. For example, one user group may enter a student's birth date erroneously as JAN-19- 1984, whereas the other user groups may enter the correct value of JAN – 29 -1984.

2. Integrity problems

Most database applications have certain integrity constraints that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints. The data in a database satisfy certain types of the consistency constraints. For example the balances of the bank account never fall below Rs. 500. The file system does not have this facility.

3. Data isolation

In file processing system, data are scattered in various files, and files may be in different formats. It is difficult to write new application programs to retrieve the appropriate data.

4. Atomicity problem

In computer, any electrical or mechanical system may fail. When failure has occurred, it is ensured that has been detected and restored to the consistent state that existed prior to the failure. This problem cannot solved by file processing system for example a program to transfer Rs1000 from account A to B. if the system failure occurs during the execution of program, it is possible that Rs1000 was removed from account A but was not credited to account B, resulting in an inconsistent database state. It is essential to database consistency that either both the credit and debit occur or that neither occurs i.e. funds transfer must be atomic. It must happen in its entirety or not at all .it is difficult to ensure this property in a file processing system.

5. Difficulty in accessing data

Database systems must provide capabilities for efficiently executing queries and updates. Because the database is stored on disk the DBMS must provide specialized data structures to speed up disk search for the desired records. Auxiliary files called indexes are used for this purpose. Using DBMS, we can access the required result from the whole database. File processing environment do not allow needed data to be retrieved in a convenient and efficient manner.

6. Concurrent access anomalies

When the number of systems runs concurrently, the result may be inconsistent data. For example, suppose account, A, containing Rs5000, if two customers withdraw Rs400 and Rs500 at about the same time the result of concurrent executions may leave the account in an incorrect state. Both balances may read the amount Rs5000 and Rs4500, respectively these types of problem is solved by database.

7. Security problem

In file processing system there is no security method. But in database system unauthorized person cannot see the data. For example in bank account, there may be number of accounts and only access the information about particular customers.

Advantages of DBMS approach

- Controlled redundancy.
- Restricting unauthorized access.
- Providing persistent storage for program objects.
- Providing storage structures for efficient Query processing.
- Providing backup and recovery.

- Providing multiple user interfaces.
- Representing complex relationships among data.
- Enforcing integrity constraints.

When a DBMS is needed?

- To integrate efficiently and correctly large volumes of related data
- Supports sharing and discovering of information
 - Data mining, deductive databases, active databases
- DBMS ensure true QoS (Quality of Services)

When a DBMS is Inappropriate?

Disadvantages:

- Expensive to buy
- Expensive to maintain (need administrator)
- “Expensive” to run (needs significant resources)

Hence, it is “over-kill” when

- The database is small
- The database has simple structure
- The application is simple and not expected to change
- Can tolerate failures
- Do not require multiple, concurrent users

DBMS Languages

- **Data Definition Language (DDL):** Used by the DBA and database designers to specify the conceptual schema of a database. In many DBMSs, the DDL is also used to define internal and external schemas (views). In some DBMSs, separate storage definition language (SDL) and view definition language (VDL) are used to define internal and external schemas.
- **Data Manipulation Language (DML):** Used to specify database retrievals and updates.
 - DML commands (data sublanguage) can be embedded in a general-purpose programming language (host language), such as COBOL, C or an Assembly Language.
 - Alternatively, stand-alone DML commands can be applied directly (query language).

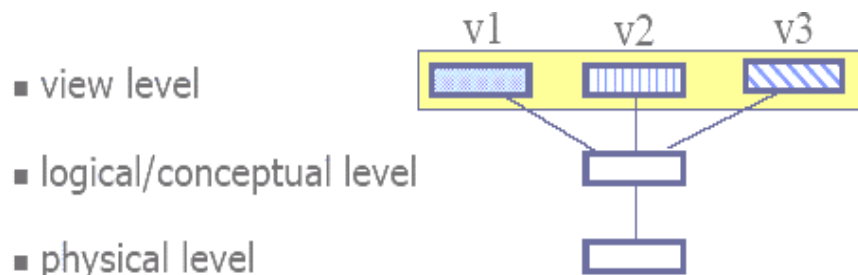
- **High Level or Non-procedural Languages:** e.g., SQL, are set-oriented and specify what data to retrieve than how to retrieve. Also called declarative languages.
- **Low Level or Procedural Languages:** record-at-a-time; they specify how to retrieve data and include constructs such as looping.

View of data

The main purpose of a database is to provide users with an abstract view of the data, i.e. the system hides certain details of how the data are stored and maintained. This is called data abstraction.

Three level architecture (ANSI / SPARC Architecture)

- View level.
- Logical / conceptual level.
- Physical level.



1. View level (external level)

It is the highest level of data abstraction. It describes only part of the entire database. Different users may view the data in different ways. This may involve viewing data in different formats or derived values which are not stored persistently in the database only the data relevant to the particular user need be included in the user's view. eg .CS majors, math major.

2. Logical level (conceptual level);

It describes what data are stored in database and relationship among these data. The entire database schema is described in terms of small number of relatively small structures. The logical structures of the entire database described by the conceptual schema. It includes all data entities attributes and relationship, integrity constraints are defined. All external view are derivable from conceptual schema it is independent of any storage details e.g.

Course (course no, course name, credits, dept)

Student (student id Lname, Fname, class, maior)

Grade-report (students ID, course No, Grade, tem

3. Physical level (Internal level)

It is the lower level of abstraction, describes how data are stored. It describes the physical storage characteristics of the data. Such as storage space allocation, indexes, It is independent of any particular application software. It also describes how the tables are stored, how many bytes / attributes it allocated etc.

Advantages of the 3-schema representation

- Each user/application should have their customized view of the data and should be able to change this view without affecting other users.
- Change to internal level should not affect user views.
- Change to physical storage device should not affect other levels.
- Changes to conceptual level should only affect those users/ applications accessing the changed data.

Database schema

It is the structure of the DB that captures data types, relationships, constraints on the data. It is independent of any application program. It may change frequently. It is similar to types and variables in programming language.

Database instances or state

It is the actual data in the database at a particular moment in time. It is analogous to the value of a variable in programming language.

Mapping

Conceptual / Internal Mapping

It defines the correspondence between the conceptual view and the stored database. It specifies how conceptual records and file are represented at the internal level. If a change is made to the storage structure definition then the conceptual / internal mapping must be changed accordingly so that the conceptual schema can remain invariant. In other words, the effect of such changes must be isolated below the conceptual level, in order to preserve physical data independence.

External / conceptual mapping

It defines the correspondence between a particular external view and the conceptual view. For example, fields can have different data types, fields and records name can be changed several

conceptual fields can be combined into a single external fields and so on. Any number of external views can exist at the same time any number of users can share a given external view, different external views can overlap.

Data independence

When a schema at a lower level is changed, only the mapping between this schema and higher level schemas need to be changed in a DBMS that fully supports data independence. The higher level schemas themselves are unchanged. Hence, the application programs need be changed since they refer to the external schemas logical data independence. The three schema architecture can be used to further explain the concept of data independence, which can be defined as the capacity to change the schema at one level of database system without having to change the schema at the next higher level. There are two types of data dependence.

Types of data independence

Logical data independence

The ability to modify the logical schema without changing the view level and application program depends on the logical schema. We may change the conceptual schema to expand the database (by adding record type or data item), to change constraints, or to reduce database (by removing record type or data item).

Physical data independence

The ability to modify the physical schema without changing the logical schema is called physical data independence. Hence the external schema need not be changed as well. Changes to the internal schema may be needed because some physical files had to be recognized. For example, by creating additional access structure – to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

Logical data independence is more difficult to achieve than in physical data independence, since the application programs are heavily dependent on the logical structure of data that they access.

The concept of data independence is similar in many respects to the concept of abstract data types in modern programming language. Both hide implementation details from the users to concentrate on the general structure, rather than on low level implementation details.

Data dictionary

The result of compilation of DDL statement is a set of tables that is stored in a special file called data dictionary. A data dictionary is a file that contains metadata that is data about data. This file is consulted before actual data are read or modified in a database system.

Data model

A collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints is called data model. The various data model that have been proposed fall into three different groups.

- Object based logical models.
- Record based logical models.
- Physical model

Object based logical model

Object based logical models are used in describing data at the logical and view levels. They are characterized by the fact that they provide fairly flexible structuring capability and allow data constraints to be specified explicitly. There are many different object based logical models.

- The entity relationship model.
- The object oriented model.
- The semantic data model.
- The functional data model.

The Entity Relationship model

The entity relationship model(ER model) is based on a perception of a real world that consists of a collection of basic objects, called entities, and of relationship among these objects and characteristics of an entity.

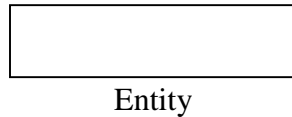
Entity-Relationship (E-R) diagram

The diagrammatic representation of entities attributes and their relationship is described by E-R diagram. The E-R diagram is an overall logical structure of a database that can be expressed graphically. It was developed to facilitated database design and the simplicity and pictorial clarity of this diagramming technique have done great help in the designing part of database. The basic components of ER diagram are given below.

Entity:

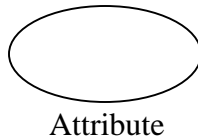
An entity is a “thing” or “object” in the real word that is distinguishable from other objects. For example each person is an entity. An entity has a set of properties and the values for some set of

properties may uniquely identify an entity. For example, if student is an entity, is identified by registration number. It is represented by rectangle.



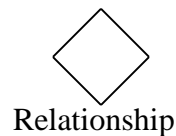
Attribute:

Attributes are properties possessed by an entity or relationship. For example stu_no, stu_name, stu_sub are the attributes of the entity student. Attribute is represented by ellipse.



Relationship:

A relationship is an association among several entities and represents meaningful dependencies between them. For example the association between teachers and students is teaching. It is represented by diamond.



The Object Oriented Model

Like E-R model object oriented model is based on a collection of objects. An object contains values stored in instance variables within an object. An object also contains bodies of code that operate on an object. The bodies of code are called methods.

Objects that contain the same types of values and the same methods are grouped together into classes. A class may be viewed as a definition for objects. This combination of data and methods comprising a type definition is similar to programming language abstract data type.

Record Based Logical Model

Record Based Logical Models are used in describing data at logical and view levels. In contrast to object-based data models, they are used both to specify the overall logical structure of the database and to provide a higher level description of the implementation.

Record based models are so named because the database is structured in fixed format records of several types. Each record type defines a fixed number of fields or attributes, and each field is usually of a fixed length. The record based logical models are given below.

1. Relational model

In June 1970, Dr. E.F. Codd published a paper entitled, A relational model of data for large shared data banks. This relational model of data for large shared data banks. This relational model, sponsored by IBM, then came to be accepted as the definitive model for relational database management system (RDBMS).

The relational model uses a collection of tables to represent both data and the relationship among those data. Each table has multiple columns, and each column has a unique name. The relational database is relatively new. The first database system was based of either a network model or hierarchical model. This model greatly improves the flexibility of the database management system. It has established itself as a primary data model for commercial data processing application.

Table 1. Teacher

Tea_id	Tea_name	Tea-address
T1	Rajesh	Balaju
T2	Gopal	Naxal
T3	Govinda	Asan
T4	Gopi	Maipi

Table 2: Student

Stu-id	Stu-name	Tea-id
S1	Gita	T1
S2	Radha	T1
S3	Nita	T2
S4	Rajan	T3
S5	Rasendra	T4

A relational database management has following properties:

- Represent data in the form of table.
- Shows the relationship between two or more its physical tables.
- Does not require the user to understand its physical implementation.
- Supports in an iterative query language.
- Provides information about its content and structure in system table.
- Support the concept of null values.

2. Network Model

Data in the network model are represented by collection of records and the relationships among data are represented by links which can be viewed as pointers. The records in the database are organized as collection of arbitrary graphs.

It is an improvement of hierarchical model. Here multiple parent-child relationship is used. The network approach allows us to build up many to many correspondences that mean each child can have more than one parent. This model is more versatile and flexible than the hierarchical model.

A record is in many respects similar to an entity in the entity relationship model. Each record is a collection of fields (attributes). Each of which contains only one data value. A link is an association between precisely two records.

A data structure diagram is a schema representing the design of network database. Such a diagram consists of two basic components, boxes, which correspond to record types, and lines which correspond to links.

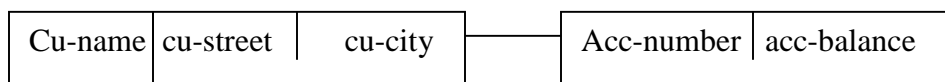
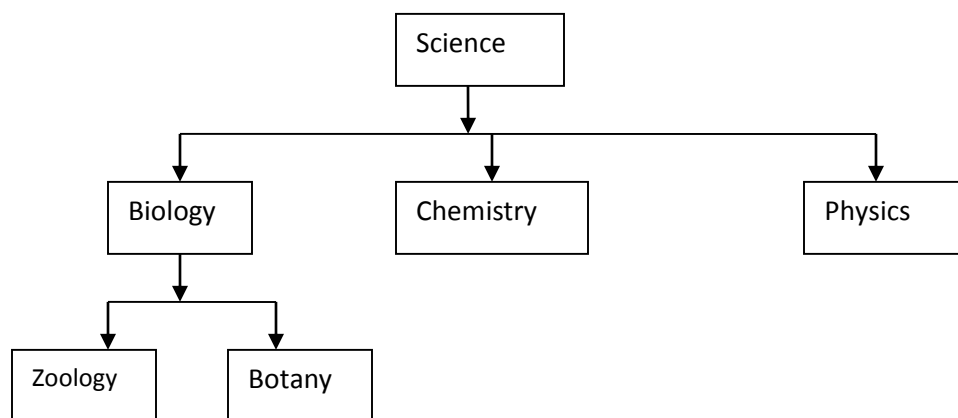


Fig: Data structure diagram

3. Hierarchical Model

The hierarchical model is similar to the network model in the sense that data and relationship among data are represented by records and links respectively. It differs from the network models in that the records are organized as collection of trees rather than arbitrary graphs.

This model is introduced in the information management system (IMS) developed by IBM in 1968. The top level of data is parent or root and others are sub-root or branches which may have subdivision.



Tree structure diagram is the schema for a hierarchical database. Such a diagram consists of two basic components, boxes which correspond to record types and lines, which correspond to links.

Disadvantage of Hierarchical Model

- This structure is not flexible enough to represent all the relationships that occur in a real world.
- It cannot have many-to-many relationships.
- The hierarchical model is used only when the concerned data has a clearly hierarchical character with a single route. E.g. DOS directory structures.

4. Physical data model

Physical data models are used to describe data at the lowest level. In contrast to logical data model, there are few physical data models such as unifying model and frame-memory model.

Transaction

A transaction is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database consistency constraints .i.e. if the database was consistent when the transaction started, the database must be consistent when the transaction successfully terminated. During the execution of a transaction it may be necessary temporarily to allow inconsistency. This temporary inconsistency is although necessary.

Storage management

- Database requires large amount of storage space. It is measured in gigabytes or terabytes.
- The large amount of data cannot store in main memory so data are moved from main memory to secondary memory as needed and vice versa. It is important that the database system structure the data so as to minimize the need to move data between disk and main memory.
- The goal of database system is to simplify and facilitate access to data. High level views help to achieve this goal.
- Storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for the interaction with the file manager.
- The raw data are stored on the disk using the file system, which is usually provided by a conventional operating system.
- The storage manager translates the various DML statements into low-level file system commands. Thus the storage manager is responsible for storing, retrieving and updating of data in the database.

Database Administrator (DBA)

DBA is a person who understands the data and the needs of the enterprise with respect to that data at a senior management level. Thus it is that a DBA's job to decide what data should be stored in the database in the first place and to establish policies for maintaining and dealing with the data.

once it has been stored. An example of such a policy might be, one that dictates who can perform what operators in what data in what circumstances, in other words security policy. Data administrator is a manager not a technical.

The DBA will typically have a staff of system programmers and other technical assistants. Thus the DBA is responsible for the overall control of the system at a technical level some tasks of DBA are as follows.

1. Schema definition

DBA creates the original database schema by writing a set of definitions that translated by the DDL compiler to a set of tables that is stored permanently in the data dictionary.

2. Storage structure and access method definition

By writing a set of definitions, DBA creates appropriate storage structure and access methods. Which is translated by the data storage and data definition language compiler.

3. Schema and physical organization modifications

DBA has the responsibility to modify schema and physical organization programmers accomplish the relatively modifications either to database schema or the description of the physical storage organization by writing a set of definitions that is used by either the DDL compiler or data storage and data definition language compiler to generate modifications to the appropriate internal system tables (e.g. data dictionary)

4. Granting of authorization for data access

The granting of different types of authorization allows the database administrator to regulate which parts of database various users can access. The authorization information is kept in a special system structure that is consulted by the database system whenever access to the data is attempted in the system.

5. Integrity constraint specification

The data values stored in the database must satisfy certain consistency constraints. For example the no of hours an employ may work in 1 week may not exceed a specified limit. Such a constraint must be specified explicitly by the database administrator. The integrity constraints are kept in a special system structure that is consulted by the database system whenever an update takes place in the system.

6. Defining dumps and reload policies

7. Monitoring performance and responding to changing requirement using.

DDL: data definition language.

SDL: storage definition language.

VDL: view definition language.

Database Users

The main goal of database system is to provide an environment for retrieving information from and storing new information into the database. These are four types of database users.

1. Application Programmers

Application programmers are persons who write the program in HLL Such as C, COBOL, PL/1, Pascal etc. They prepare program for bank, hospital, school etc.

2. Sophisticated users

They interact with system without writing programs. They form their requests in the database query language. Each such query is submitted to query processor. Whose function is to break down DML statement into instructions that the storage manager understands. The analyst who submits queries to explore data in a database is called sophisticated users.

3. Specialized Users

They are sophisticated users who write specialized database applications that do not fit into traditional data processing framework. Among these applications are computer aided design system, knowledge base and expert system. System that store data with complex data types eg graphic data, audio data etc.

4. Naïve Users

They are unsophisticated users who interact with system by invoking one of the permanent application programs that have been written previously. eg. a bank teller who needs to transfer \$50 from account A to account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from which the money to be transferred, and the account to which the money is to be transferred

Overall Database System

The functional components of a database system can be broadly divided into

- Storage Manager
- Query Processor Components

Storage Manager

Storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the following tasks.

- Interaction with the file manager.
- Efficient storing, retrieving and updating data.

The storage manager components includes:

1. Authorization and integrity manager

Which tests for the satisfaction of integrity constraints and checks the authority of users to access data

2. Transaction Manager

Which ensures that the database remains in a consistent (correct) state despite system failures, and that current transaction execution proceed without conflicting

3. File manager

Which manages the allocation of space on disk storage and that concurrent transaction execution proceed without conflicting.

4. Buffer Manager

Which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The manager is critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

The storage manager implements several data structures as part of the physical system implementation.

- **Data files**
Which store the database itself.
- **Data Dictionary**
Which stores metadata about the structure of the database in particular the schema of the database.
- **Indices**
Which provides fast access to data items that hold particular values.
- **Statistical data**
Which stores statistical information about data in the database. This information is used by query processor to select efficient ways to execute a query.

The Query processor

The query processor components include.

- **DDL Interpreter**
Which interprets DDL statements and records the definitions in the data dictionary.
- **DML Compile**
Which translates DML statements in a query language into an evaluation plan consisting of low level instructions that the query evaluation engine understand.
- **Embedded DML pre-compiler**
Which converts DML statements embedded in an application program to normal procedure call in the host language. The pre-compiler must interact with the DML compiler to generate the appropriate code.

- **Query Evaluation Engine**


Which executes low level instructions generated by DML compiler.

Relational Model

The relational model is today the primary model for commercial data processing applications. It has attained its primary position because of its simplicity as compared to earlier data models such as the network model or the hierarchical model. It is a lower level model that uses a collection of tables (also called relations) to represent both data and the relationship among those data. A table of values is called relation. A relation may be thought of as a **set of rows**. A relation may alternately be thought of as a **set of columns**. Each row represents a fact that corresponds to a real-world **entity** or **relationship**. Each row has a value of an item or set of items that uniquely identifies that row in the table. Sometimes row-ids or sequential numbers are assigned to identify the rows in the table. Each column typically is called by its column name or column header or attribute name. Each table has multiple columns and each column has a unique name.

Example: RDBMS

<u>Eid</u>	<u>Ename</u>	<u>Salary</u>	<u>City</u>	<u>Dno</u>
E01	Ajaya	23000	Kathmandu	D1
E02	Binek	35000	Kathmandu	D2
E03	Chandru	32000	Pokhara	D3
E04	Dina	27000	Biratnagar	D3
E05	Elina	18000	Dhanagadhi	D2



<u>Dno</u>	<u>Dname</u>	<u>Head</u>
D1	Finance	E04
D2	Management	E01
D3	IT	E02

Formal Definitions

A **Relation** may be defined in multiple ways.

The **Schema** of the form: $R(A_1, A_2 \dots A_n)$ is called relation, Relation schema R is defined over attributes $A_1, A_2 \dots A_n$.

For Example:

CUSTOMER (Cust-id, Cust-name, Address, Phone#)

Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has a **domain** or a set of valid values. For example, the domain of Cust-id is 6 digit numbers.

Tuple

A tuple is an ordered set of values. Each value is derived from an appropriate domain. Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values. $\langle 632895, \text{"John Smith"}, \text{"101 Main St. Atlanta, GA 30332"}, \text{"(404) 894-2000"} \rangle$ is a tuple

belonging to the CUSTOMER relation. A relation may be regarded as a set of tuples (rows or records).

Domain

A domain has a logical definition: e.g., “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S. A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as month name, date, year or yyyy-mm-dd, or dd mm, yyyy etc. An attribute designates the role played by the domain. E.g., the domain Date may be used to define attributes “Invoice-date” and “Payment-date”. The relation is formed over the Cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name. For example, attribute Cust-name is defined over the domain of strings of 25 characters. The role these strings play in the CUSTOMER relation is that of the name of customers.

Formally,

Given $R(A_1, A_2, \dots, A_n)$

$r(R) \subset \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$

R: schema of the relation

r of R: a specific "value" or population of R.

R is also called the intension of a relation

r is also called the extension of a relation

Example

Let $S_1 = \{0, 1\}$

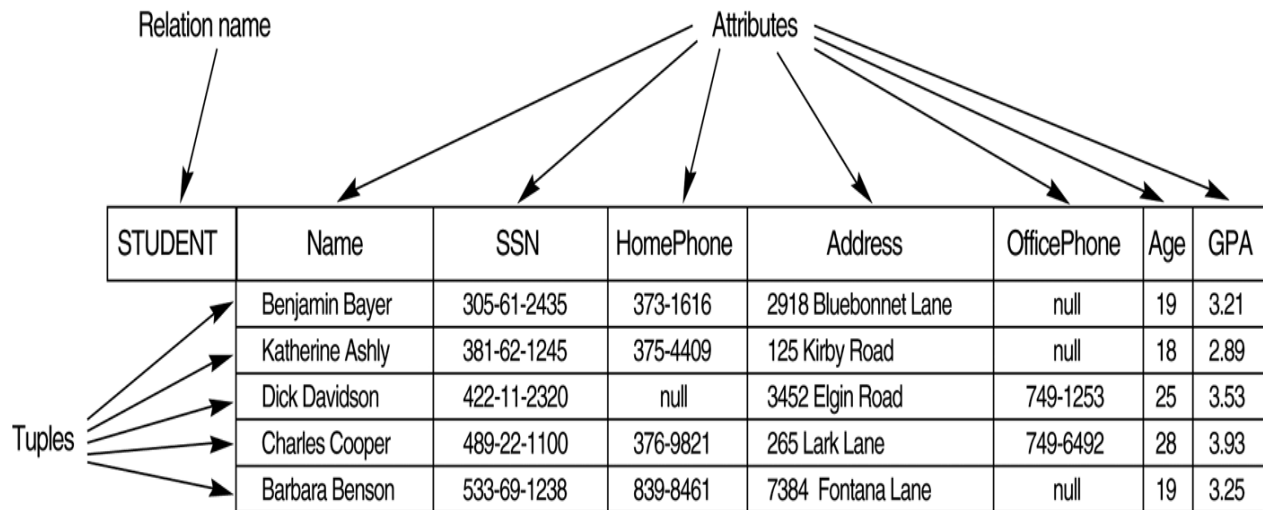
Let $S_2 = \{a, b, c\}$

Let $R \subset S_1 \times S_2$

Then for example: $r(R) = \{ \langle 0, a \rangle, \langle 0, b \rangle, \langle 1, c \rangle \}$ is one possible “state” or “population” or “extension” r of the relation R, defined over domains S_1 and S_2 . It has three tuples.

Definition Summary

Informal Terms	Formal Terms
Table	Relation
Column	Attribute/Domain
Row	Tuple
Values in a column	Domain
Table Definition	Schema of a Relation (Intension)
Populated Table	Extension



Data Integrity

The fundamental function of the DBMS is to maintain the integrity of the data. Data integrity ensures that the data in the database is consistent, accurate, correct, and valid. It ascertains that the data adhere to the set of rules defined by the database administrator and hence, prevents the entry of the invalid information into database. Data integrity is of four types, namely, *domain integrity*, *entity integrity*, *referential integrity*, and *semantic integrity*. Generally, domain integrity is applied on the attribute; entity integrity is applied on the tuple; referential integrity is applied on the relation; and semantic integrity ensures logical data in the database.

1. Entity Integrity

The entity integrity constraint states that no primary key (nor any part of the primary key) value can be null. This is because the primary key value is used to identify individual tuples in a relation. Having null value for the primary key implies that we cannot identify some tuples. This also specifies that there may not be any duplicate entries in primary key column.

2. Referential Integrity

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations (Parent and Child). Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. It is a rule that maintains consistency among the rows of the two relations. Examples of Referential integrity constraint in the Customer/Order database:

Customer (custid, custname)

Order (orderID, custid, OrderDate)

To ensure that there are no orphan records, we need to enforce referential integrity.

An orphan record is one whose foreign key value is not found in the corresponding entity – the entity where the PK is located.

3. Domain Integrity

A **domain** represents a set of values that can be assigned to an attribute. Domain integrity constraint is specified on the column (attribute) of a relation, so that correct values can be entered in the column for each record. The domain integrity states that every element from a relation should respect the type and restrictions of its corresponding attribute. For example, we can specify the value of 'Marks' attribute of "Student" table which must be greater than 0 and less than or equal to 100.

4. Semantic Integrity

To represent real world accurately and consistently, business rules and logical rules must be enforced in database. Such rules are derived from our knowledge of the application semantics and are called semantic integrity constraints. Semantic integrity ensures that data in the database is logically consistent and complete with respect to the real world. This type of integrity cannot be expressed by the model and contains integrity constraints like:

- Number of pages of a book cannot be zero
- A book is published by only one publisher
- An author cannot review his own book etc.

A constraint specification language may have to be used to express these rules. SQL-99 allows triggers and assertions to allow for some of these.

Advanced SQL

SQL stands for structured query language developed at IBM research for system R. It includes features of relational algebra and tuple relational calculus. It is standard for relational data access. It is DBMS independent. It is one commercially available query language. SQL can define the structure of data create table, index, view alter table etc. modify data in the data base such as select, update, delete, insert etc.

SQL Data type

- Numeric
- Character strings
- Bit strings
- Temporal data
- Null value

Numeric data

- **Exact number**

Two integer types with different ranges.

INTEGER (or INT) and SMALLINT

- **Approximation numbers**

Three floating point type:

FLOAT, REAL and DOUBLE PRECISION.

- User can define the precision for FLOAT
- The precision of REAL and DOUBLE PRECISION is fixed.
- Floating point numbers can be in decimal or scientific notation.

SQL Character strings

A Character string is a sequence of printable chars. In SQL, character string is denoted by enclosing it in single quote; e.g. 'Hello SQL'

Two types of character strings :

Fixed length n: CHAR (n) or CHARACTER (n)

Varying length of maximum n: VARCHAR (n) or CHARACTER (n).

SQL Bit Strings

Bit strings are sequences of binary digits or bits. Bit string types are:

Fixed length n: BIT (n)

Varying length of maximum n: VARBIT (n) or BITVARYING (n). The default value for n is 1.

SQL temporal data

Date data type

Date format is: YYYY-MM-DD

Time data type

Time format is HH: MM: SS

Interval data type

E.g. Admit date – discharge date.

Format is INTERVAL start field (p) [to end – field (fs)]

SQL components

1. Data definition language (DDL):

The SQL DDL provides commands for defining relation schema, deletion relations, creating indices, and modifying relation schemas.

2. Interactive data definition language (DML):

The SQL DML includes languages 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.

3. Embedded DML:

The embedded form of SQL is designed for use with in general purpose programming languages, such as PL/1, COBOL, PASCOL, FORTRAN and C.

4. View definition:

The SQL DDL includes commands for defining views.

5. Authorization:

The SQL DDL includes commands for specifying access rights to relations and views.

6. 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.

7. Transaction control:

SQL includes commands for specifying the beginning and ending of transactions. Several implementations also allow explicit locking of data for concurrency control.

Table creation

A new relation can be created using the CREATE TABLE command. The general syntax is as follows.

```
CREATE TABLE table_name [{column descriptors}];
```

e.g. CREATE TABLE DEPARTMENT;

Table Creation Example

```
CREATE TABLE employee
```

```
(  
    Emp-no      INT(6) NOT NULL,  
    Name        VARCHAR(20),  
    Gender      CHAR,  
    DOB         DATE,  
    Salary      DECIMAL (8, 2),  
    PRIMARY KEY (EMP-no)  
);
```

The relations for banking enterprise are given below:

Branch = (branch_name, branch_city, asserts)

Customer = (customer_name, customer-street, customer- city)
 Loan = (branch- name, loan number amount)
 Borrower = (customer- name, lone- number)
 Account = (branch_name, account_number, balance)
 Depositor = (customer- name, account_number)

Select Overall Form

```
SELECT    <attributes and function list >
FROM      < table list >
[WHERE    < condition >]
[GROUP BY <grouping attributes >]
[HAVING   <group condition >]
[ORDER BY < {attribute ASC/DESC} list >];
```

Basic Structure

The basic structure of an SQL expression consists of three clauses. SELECT, FROM and WHERE.

The SELECT clause corresponds to the project operation of the relational algebra. It is used to list the attributes desired in the result of a query. The FROM clause corresponds to the Cartesian product operation of relational algebra. It lists the relations to be scanned in the evaluation of the expression. The WHERE clause corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the from clause.

The SQL query is the form

```
SELECT A1 A2,.....,An
FROM R1, R2, .....Rn
WHERE P;
```

Where A1 A2,.....,An are attributes, R1, R2,Rn are relations and p is predicate i.e. condition.

The Rename operation

SQL allows renaming attributes.

Syntax

Old_name AS new_name

We can use as clauses in both select and from clause.

Example: loan_no is replaced by loan_id

```
SELECT customer_name, borrower.loan_number AS loan_id
FROM borrower, loan
WHERE borrower.loan_number = loan.loan_number AND branch_name = 'bagbazar';
```

Tuple Variables

The as clause is used in defining the notion of tuple variable, as is done in the tuple relational calculus. The tuple variable in SQL must be associated with a particular relation. Tuple variables are defined in the FROM clause.

#For all customers who have a loan from the bank, find their names and loan numbers.

```
SELECT DISTINCT customer_name, T.loan_number
FROM borrower AS T, loan AS S
WHERE T.loan_number=S.loan_number;
```

String operations

The commonly used operation on string is pattern matching using the operator “like”. We use two special characters:

Percent (%): it matches any substring.

Underscore (-): it matches any character.

Example: Find the names of all customers whose street address includes the substring "main" is:

```
SELECT customer_name
FROM Customer
WHERE street like "%main%"
```

Some examples are:

- “Perry%” matches any string beginning with “Perry”.
- “%edge%” matches any string containing “edge” as a sub string.
- “- - -” matches any string of exactly three characters.
- “- - -%” matches any string of the least three characters.

Ordering the display of tuples

The ORDER BY clause is used to list the tuples in sorted order.

Example: To list in alphabetical order all customers who have a loan at Bagbazar branch is:

```
SELECT DISTINCT customer_name
FROM borrower, loan
WHERE Borrower.loan_number = loan.loan_number AND Branch_name = 'bagbazar'
ORDER BY customer_name ;
```

We can specify DESC for descending order or ASC for ascending order.

E.g. List all attributes from loan relation amount by descending order and loan_number by ascending order.

```
SELECT *  
FROM loan  
ORDER BY amount DESC, loan_number ASC.
```

Set operations

Set operations are union, intersect and except.

The union operation

Find all customers having a loan, an account or both is:

```
(SELECT customer_name  
FROM depositor)  
UNION  
(SELECT customer_name  
FROM borrower)
```

The union operation automatically eliminates duplicates. For duplicate value we use union all in place of union.

```
(SELECT customer_name  
FROM depositor)  
UNION ALL  
(SELECT customer_name  
FROM borrower)
```

The intersection operation

To find all customers who have a both loan and an account at the bank is:

```
(SELECT distinct customer_name  
FROM depositor)  
INTERSECT  
(SELECT distinct customer_name  
FROM borrower)
```

The select operation automatically removes duplicates. If we want all duplicates, than intersect all in place of intersect.

E.g.

```
(SELECT customer_name  
FROM Depositor)
```

```
INTERSECT ALL
(SELECT customer_name
FROM borrower)
```

The except operation

To find all customers who have an account but no loan at the bank is:

```
(SELECT DISTINCT customer_name FROM depositor)
EXCEPT
(SELECT customer = name FROM borrower)
```

Except operation automatically removes duplicated. If we want to retain all duplicates then except is replaced by EXCEPT ALL.

```
(SELECT customer-name FROM depositor)
EXCEPT ALL
(SELECT customer_name FROM borrower)
```

Aggregate functions

Aggregate functions that take a collection of value as input and returns a single value. Aggregate functions are given below:

- Average : AVG
- Minimum : MIN
- Maximum : MAX
- Total : SUM
- Count : COUNT

To find the average account balance of Bagbazar branch.

```
SELECT AVG (balance)
FROM account
WHERE branch- name = 'Bagbazar'
```

For the group of tuples, we can use group by to find the average account balance at each branch is:

```
SELECT branch_name, AVG (balance)
FROM account
GROUP BY branch- name
```

If we want duplication is removed then **distinct** is used. To find the number of depositors for each branch is:

```
Select branch_name, count (distinct customer- name)
```

From depositor, account

Where depositor account- number = account_account_number

Group by branch_name.

For the condition in **group by** clause we use having clause

e.g. To list branch_name and average balance group by branch_name having average balance greater than 1200 is:

Select branch_name, **avg** (balance)

From account

Group by branch_name

Having avg (balance) > 1200

To find the average balance for all customers:

Select avg (balance)

From account

To find number of tuples in the customer relation:

Select count (*)

From customer

Find the averages balance for each customer who lives in lalitpur and has at least three accounts.

Select depositor.customer- name, **avg** (balance)

From depositor, account, customer

Where depositor.account_number = account.account_number **and**
depositor.customer_name = customer.customer_name **and** customer_city = "lalitpur"

Group by depositor.customer_name

Having count (distinct depositor.Account_number) > = 3

Null values

Null values are values that indicate absence of information about the value of an attribute.

To find the loan number with null values.

Select loan-number

From loan

Where amount is null;

Nested Sub queries

A sub query is nested inside another query called nested sub queries. A common use of sub queries is to perform tests for set membership, set companions and set cardinality.

Set membership

The **in** connective tests for set membership, where the set is a collection of values produced by a **SELECT** clause.

The **NOT IN** connective tests for the absence of set membership consider the query:

Find all customers who have both a loan and account at the bank. This is solved by union operation and another approach is finding all account holders at the bank who are members of the set of borrowers from the bank. This formation generates the same results as did the previous one but it leads us to write our query using **in** connective of SQL. We first by finding all account holders.

(Select customer_name

From depositor)

We then need to find those customers who are borrower from banks and who appear in the list of account holders obtained on the sub query. The result is

select distinct customer_name

form borrower.

Where customer_name **in** (**select** customer_name **from** depositor);

Find all customers who have both on account and loan at the Bagbazar branch.

(Select distinct customer_name

From borrower, loan

Where borrower . loan number = loan . loan-number.

and branch_name = "bagbazzar" **and** (branch_name, customer_name) **in** (**select**

branch_name, customer_name **from** depositor, account **Where**

depositor.account_number = account.account_number)

We also use **not in** clause

To find all customer who have a loan at a bank but don't have an account at the bank.

Select distinct customer_name

From borrower

Where customer_name **not in** (**select** customer_name

From depositor);

Set Comparison

We use comparison operators for set comparison operations. "Greater than at least one" is represented by **> some**.

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

```
Select branch_name
From branch
Where assets > some (select asserts From branch Where branch_city = 'Bagbazar');
```

SQL allows < some, < = some, =some, and < > some is not the same as not in.

Find the names of all branches that have asserts greater than that of each branch in Bagbazar.

```
SELECT branch_name
FROM branch
WHERE assets > ALL (SELECT asserts FROM branch WHERE branch_city = 'Bagbazar')
```

SQL also allows < all, <= all, > = all, and < > all comparisons

Find the branch that has the highest average balance

```
SELECT branch_name
FROM account
GROUP BY branch_name
HAVING AVG (balance) > = ALL (SELECT AVG (balance)
                                FROM account
                                GROUP BY branch_name);
```

Test for empty relations

SQL allows for testing whether a sub query has any tuples in its result. The EXISTS construct returns the value true if the argument sub query is non empty.

Find a customers who have both an account and a loan at the bank is:

```
SELECT customer_name
FROM borrower
WHERE EXISTS (SELECT * FROM depositor
              WHERE depositor.Customer_name = borrower.Customer_name);
```

There is nonexistence tuples to test in a sub query by using the non-exists construct. We can use the not exists construct to simulate the set containment (i.e. subset) operation. We can write "relation A contains relation B" as "not exists (B except A)"

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

```
SELECT DISTINCT s.customer_name
FROM depositor AS S
```



```
WHERE NOT EXISTS (SELECT branch_name FROM branch
WHERE branch_city = 'Bagbazar')
```

EXCEPT

```
(SELECT R.branch_name FROM depositor AS T, account AS R
WHERE T.account_number = R.account_number AND S.customer name = T.customer
name)
```

Derived Relations

The result of the relation can be renamed and the attributes also renamed by AS clause.

```
(SELECT branch_name, AVG (balance) FROM depositor
GROUP BY branch_name AS result (branch_name, avg – balance)
```

View

We can create view in SQL by CREATE VIEW command. The syntax is given below:

```
CREATE VIEW V AS < query expression >
```

We can define view for the names of customers who have either an account or a loan is:

```
CREATE VIEW all_customer AS
(SELECT branch_name, customer_name FROM depositor, account
WHERE depositor.account_number = account.Account_number )
```

UNION

```
(SELECT branch_name, customer_name
FROM borrower, loan
WHERE borrower.loan_number = loan.loan_number);
```

Modification of the Database

SQL allows deleting, inserting and modifying data in a database.

Deletion

SQL allows to delete only whole tuples. We cannot delete values on only particular attributes.

Syntax

```
DELETE FROM r
WHERE p;
```

Where r is the relation in which we have to delete values and p is the predicate condition

E.g. To delete Puspa's account records

```
DELETE FROM depositor
WHERE customer_name = 'puspa';
```

To delete all loan amounts between 4000 and 5000

```
DELETE FROM loan
WHERE amount BETWEEN 4000 AND 5000;
```

To delete all account at every branch located in Bagbazar

```
Delete from account
Where branch_name in (select branch_name from branch
Where branch_city = 'bagbazar');
```

To delete the records of all accounts with balance below the average at the bank.

```
Delete from account
Where balance < (select avg (balance) FROM account)
```

Insertion

We can insert tuples in the relation. The attributes values for inserted tuples must be members of the attributes domain. Tuples inserted must be of the correct arity.

To insert account_no 501 at the Bagbazar branch and the balance is 5000

```
INSERT INTO account VALUES ('Bagbazar', 501, 5000)
```

It is equivalent to:

```
INSERT INTO account (branch_name, account_number , balance)
VALUES ('Bagbazar', 501, 5000);
```

It is also equivalent to:

```
Insert into account (account_number, branch_name, balance)
Values (501, 'Bagbazar', 5000)
```

The insert statement considered only examples in which a value is given for every attribute in inserted tuples. It is possible for inserted tuples to the given values on only some attributes of the schema. The remaining attributes are assigned a null value denoted by NULL e.g.

```
Insert into account
Values (NULL, 'B-101', 1500)
```

We know that account B-101 has Rs 1500 but branch name is not known.

Updates

We can change a value in a tuple without changing all values in the tuple. For this, update statement can be used.

To increase the balance by 5 percent

```
UPDATE account
SET balance = balance * 1.05;
```

Account with balance over 10,000 receives 6 percent interest and other receives 5 percent.

```
UPDATE account
SET balance = balance * 1.06
WHERE balance > 10000
UPDATE account
SET balance = balance * 1.05
WHERE balance <= 10000;
```

Update of a view

We can create view by:

```
Create view branch_loan as
Select branch_name, loan_number
From loan;
```

We can update by:

```
Insert into branch_loan
Values ('pokhara', 305)
```

Data Definition Language

The SQL DDL allows the specification of set of relations and information about each relation. The information includes:

- The schema for each relation.
- The domain of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Schema Definition in SQL

We can define SQL relation using the create table command.

```
Create table r (A1 D1, A2 D2,.....,An Dn, < integrity_constraint 1>
.....
< integrity_constraint k >);
```

Where r is the name of the relation each A is the name of an attribute type in the values in the domain of attribute A_i.

Examples:

```
CREATE TABLE customer
(customer_name CHAR (20) NOT NULL,
customer_strees CHAR (30),
customer_city CHAR (30),
PRIMARY KEY (customer_name));
```

```
CREATE TABLE Branch
(branch_name CHAR(15) NOT NULL,
Branch- city CHAR (30),
Assets  INTEGER,
PRIMARY KEY (branch_name),
CHECK (assets >= 0);
```

```
CREATE TABLE account
(account_number CHAR(10) NOT NULL,
branch_name CHAR(15),
Balance INTEGER,
PRIMARY KEY (account_number),
CHECK (balance >= 0));
```

```
CREATE TABLE depositor
(customer_name CHAR(20) NOT NULL,
account_number CHAR(10) NOT NULL,
PRIMARY KEY (customer_name,
account_number)),
```

Delete Table

To remove a relation from SQL database. We use DROP TABLE command. It removes all information and table also. The command is

```
DROP TABLE r
```

Where r is the relation

Alter Table

We use alter table command in SQL to add attributes to the existing relation. All tuples in the relation are assigned **null** as the value for the new attribute. The form of alter table command is:

```
ALTER TABLE r ADD A D
```

Where r is the name of existing relation, A is the name of attribute to be added and D is the domain of the added attribute. We can drop attributes from a relation using a command:

```
ALTER TABLE r DROP A
```

Where r is the name of an existing relation and A is the name of attribute in a relation.

Integrity constraints

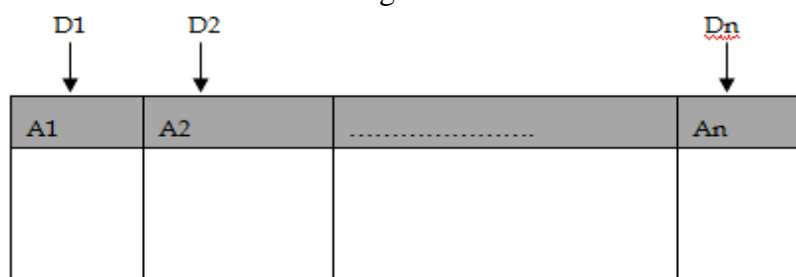
The term integrity refers to the accuracy or correctness of data in the database. Integrity constraint is a condition specified on a database schema which must hold on all of valid relation instances. Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database. Constraints are basically used to impose rules on the table, whenever a row is inserted, updated, or deleted from the table. Constraints prevent the deletion of a table if there are

dependencies. The different types of constraints that can be imposed on the table are domain constraints, referential constraints, trigger, assertions etc. The constraints related to domain constraints are NOT NULL, UNIQUE, PRIMARY KEY and CHECK constraints.

1. Domain constraints
 - NOT NULL constraints
 - UNIQUE constraints
 - PRIMARY KEY constraints
 - CHECK constraints etc.
2. Referential constraints
3. Triggers
4. Assertion

Domain constraint

Domains are used in the relational model to define the characteristics of the columns of a table. Domain refers to the set of all possible values that attribute can take. The domain specifies its own name, data type, and logical size. The logical size represents the size as perceived by the user, not how it is implemented internally. For example, for an integer, the logical size represents the number of digits used to display the integer, not the number of bytes used to store it. The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. We can define the valid values by listing them as a set of values (such as an enumerated data type in a strongly typed programming language), a range of values, or an expression that accepts the valid values. Strictly speaking, only values from the same domain should ever be compared or be integrated through a union operator. The domain integrity constraint specifies that each attribute must have values derived from a valid range



The **create domain** clause can be used to define new domains. For example, to ensure that age must be an integer in the range 1 to 100, we could use:

```
CREATE DOMAIN Ageval INTEGER
CHECK (VALUE >= 1 AND VALUE <= 100)
```

The domain can be restricted to contain only a specified set of values by using IN clause:

```
CREATE DOMAIN AccountType CHAR (10)
```

```
CONSTRAINT account-type-test
CHECK (VALUE IN ('Checking', 'Saving'))
```

SQL also provides drop domain and alter domain clauses to drop or modify domains that have been created earlier.

CHECK Constraints

CHECK constraint is added to the declaration of the attribute. The CHECK constraint may use the name of the attribute or any other relation or attribute name may in a sub-query. Attribute value check is checked only when the value of the attribute is inserted or updated. CHECK constraints allow users to prohibit an operation on a table that would violate the constraint. It is a local constraint.

Example: let's create a student table with attributes student id, student name, age and address. If we need to allow only those students in the table whose age must be an integer range 20 to 45, we could use the CHECK constraint during the creation of table as below:

```
CREATE TABLE Student
(
    sid    INTEGER,
    sname  VARCHAR(20),
    age    INTEGER,
    PRIMARY KEY (sid),
    CHECK (age >= 20 AND age <= 45)
)
```

In the above student table if we are trying to insert a new record as

```
INSERT INTO Student
VALUES (5, "Rajesh", 15);
```

We get insertion is rejected message since value of age attribute violated the check condition.

Referential Integrity

In the relational data model, associations between tables are defined through the use of foreign keys. The referential integrity rule states that a database must not contain any unmatched foreign key values. It is to be noted that referential integrity rule does not imply a foreign key cannot be null. There can be situations where a relationship does not exist for a particular instance, in which case the foreign key is null. A referential integrity is a rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null. Referential integrity ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation to establish the relationship between tables.

In relational model we use primary key and foreign key to establish relationships between two relations. Table containing primary key attribute is called master table and the table containing foreign key attribute is called child table. Referential integrity ensures that value appeared in foreign key attribute of child table must also appear in primary key attribute of corresponding master table. Defining referential integrity restricts database modification operations such as insert, delete, and update. To illustrate this consider following two relations:

Employee				Department		
Eid	Ename	Salary	Dno	<u>Dno</u>	Dname	Location
E01	Ramesh	26000	D1	D1	IT	Gwarko
E02	Sohan	19000	D1	D2	Finance	Satdobato
E03	Renuka	25000	D2	D3	HR	Balkumari
E04	Dina	22000	D3			

- **Insert:** We cannot insert new tuples containing value of foreign key attribute that do not appear in primary key attribute of master table. For example, we cannot insert new employee that works in D4 department because the department D4 does not exist in department table.
- **Delete:** We cannot delete tuples containing values of primary key attribute that also appear foreign key attribute of related table. For example, we cannot delete tuple containing value D2 of DNO from department table because there are employees working in department D2.
- **Update:** There are two cases of update operation
 - **Case1:** We cannot change the value of primary key attribute if the child table contains related values. For example, we cannot change value of DNO from D1 to D5 in department table because employee table contains employees working in department D1.
 - **Case 2:** We cannot change value of foreign key attribute if the primary key attribute does not contain modified value of foreign key attribute. For example, we cannot change value of DNO from D3 to D5 in employee table because department table does not contain department D5.

Referential Integrity in SQL

Primary keys and foreign keys can be specified as parts of the SQL create table statement as below:

CREATE TABLE Books

```
(
    ISBN VARCHAR (20),
    Title  VARCHAR (20),
    Category VARCHAR (15),
    Price  INTEGER,
    Pages  INTEGER,
    Year   DATE (10),
    Pid    VARCHAR (5),
    PRIMARY KEY (ISBN),
    FOREIGN KEY (Pid) REFERENCES Publisher (Pid) ON DELETE CASCADE ON
    UPDATE CASCADE
)
```

CREATE TABLE *Publisher*

```
(
    Pid VARCHAR (10),
    Pname VARCHAR (15),
    City  VARCHAR (20),
    Email VARCHAR (20),
    PRIMARY KEY (Pid)
)
```

- Alternatively, we can use on delete set null and on update set null.
- Also, we can use on delete set default and on update set default.

Assertions

Assertions are general purpose checks that allow the enforcement of any condition over the entire database. Similar to CHECK but they are global Constraints. When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion. This testing may introduce a significant amount of overhead; hence assertions should be used with great care. An assertion in SQL takes the form:

CREATE ASSERTION <assertion-name> CHECK <predicate>

Example: The department id of manager relation is always not null since each manager works at least one department.

CREATE ASSERTION Noallow CHECK

(NOT EXISTS (SELECT * FROM MANAGER WHERE DeptId IS NULL));

Above assertion ensures that there is no manager who is not assigned any department at any time.

Let's take a manager relation in which some records are inserted as

Manager

Mid	Mname	Address	DeptId
M01	Aayan	Pokhara	D11
M02	Bhupi	Lalitpur	D22
M03	Arjun	Kathmandu	D11
M05	Ramesh	Palpa	Null

In the above table the department id of manager 'Ramesh' is NULL due to which assertion is violated and we cannot further modify the database.

Assertions can be dropped using the DROP ASSERTION command.

```
DROP ASSERTION Noallow;
```

Example 2: The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```
CREATE ASSERTION sum-constraint
CHECK (NOT EXISTS (SELECT * FROM branch
                    WHERE (SELECT SUM (amount) FROM loan
                           WHERE loan.branch-name =branch.branch-name) >= (SELECT
                           SUM (amount) FROM account
                           WHERE loan.branch-name = branch.branch-name))));
```

Example 3: To specify the constraint that the salary of an employee must not be greater than the salary of the manager of the department that the employee works for in SQL, we can write the following assertion:

```
CREATE ASSERTION Salary_Constraint
CHECK (NOT EXISTS (SELECT * FROM Employee E, Employee M, DEPARTMENT D
                    WHERE E.Salary >M.Salary AND E.Dno=D.Dnumber AND D.Mgr_ssn =M.Ssn));
```

Difference between CHECK constraint and Assertion

A major difference between CREATE ASSERTION and the individual domain constraints and tuple constraints is that the CHECK clauses on individual attributes, domains, and tuples are checked in SQL only when tuples are inserted or updated. Hence, constraint checking can be implemented more efficiently by the DBMS in these cases. The schema designer should use CHECK on attributes, domains, and tuples only when he or she is sure that the constraint can only be violated by insertion or updating of tuples. On the other hand, the schema designer should use CREATE ASSERTION only in cases where it is not possible to use CHECK on attributes, domains, or tuples, so that simple checks are implemented more efficiently by the DBMS.

Triggers

A trigger is a procedure (statement) that is automatically invoked by the DBMS in response to specified changes to the database. A database that has a set of associated triggers is called an active database. Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met. It is the most practical way to implement routines and granting integrity of data. Unlike the stored procedures or functions, which have to be explicitly invoked, these triggers implicitly get fired whenever the table is affected by the SQL

operation. For any event that causes a change in the contents of a table, a user can specify an associated action that the DBMS should carry out. Trigger follows the Event-Condition-Action scheme (ECA scheme). To design a trigger mechanism, we must meet following three requirements:

- Event: A change to the database that activates the trigger.
- Condition: Trigger performs some action only if a specified condition matches at the occurrence of the event
- Action: A procedure that is executed when the trigger is activated and its condition is true.

General form of trigger:

```
CREATE TRIGGER <trigger-name>
    <Time events>
    ON <list-of-tables>
    WHEN <Predicate>
    <Action-name>
```

Need of Triggers

Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met.

For example, for every pre-paid account whose balance is less than or equal to 0, the account is automatically marked as “blocked”.

```
CREATE TRIGGER overdraft AFTER UPDATE ON pre-paid
    REFERENCING NEW ROW AS nrow
    FOR EACH ROW
    WHEN nrow.balance <= 0
    UPDATE pre-paid
    SET blocked = 'T';
```

Let's take a pre-paid relation as

Pre-paid

PNO	Balance	Blocked
1	4000	False
2	5000	False
3	7000	False
4	2000	False

PNO	Balance	Blocked
1	4000	T
2	5000	False
3	7000	False
4	2000	T

```
UPDATE TABLE pre-paid
```

```
SET Balance = Balance-4000;
```

If we execute this query then we get following modified Pre-paid table

Query Processing and Optimization

Query Processing is a procedure of transforming a high-level query (such as SQL) into a correct and efficient execution plan expressed in low-level language. A query processing select a most appropriate plan that is used in responding to a database request. When a database system receives a query for update or retrieval of information, it goes through a series of compilation steps, called **execution plan**. In the first phase called **syntax checking** phase, the system parses the query and checks that it follows the syntax rules or not. It then matches the objects in the query syntax with the view tables and columns listed in the system table. Finally it performs the appropriate query modification. During this phase the system validates the user privileges and that the query does not disobey any integrity rules. The execution plan is finally execute to generate a response. So query processing is a stepwise process.

Q. What do you mean by query processing? What are the various steps involved in query processing? Explain with the help of a block diagram.

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

- Parsing and translation
- Optimization
- Evaluation

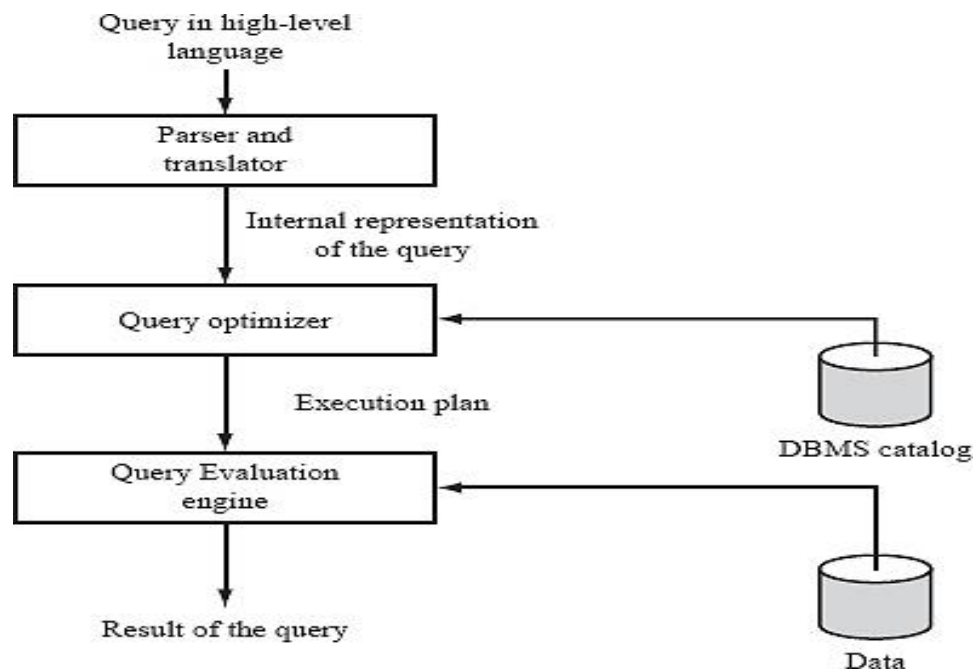


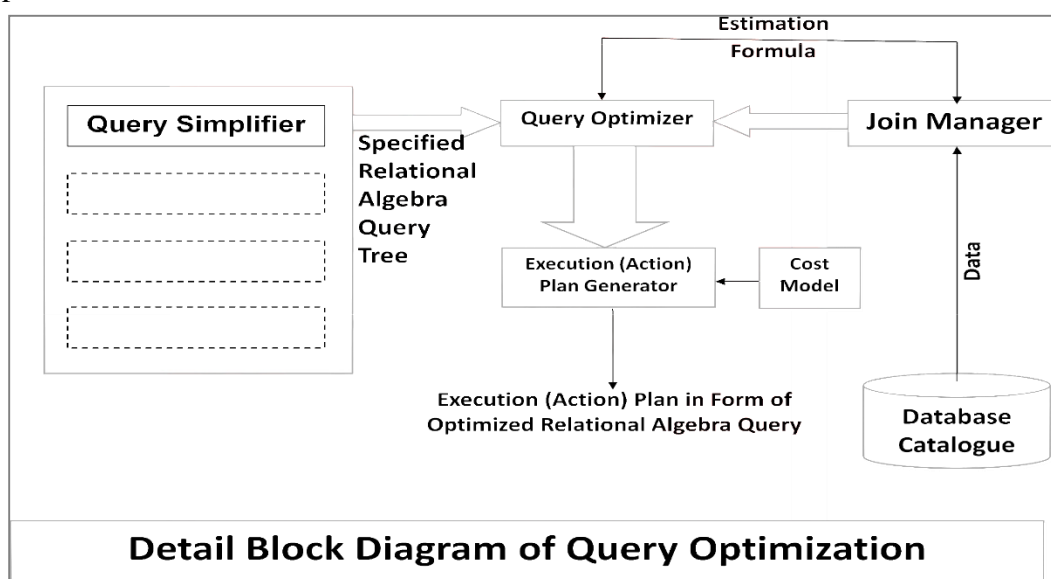
Fig: Query Processing steps

1. Parsing and translation

Check syntax and verify relations. Translate the query into its internal form. This is then translated into relational algebra.

2. Optimization

The primary goal of query optimization is of choosing an efficient execution strategy for processing a query. The query optimizer attempts to minimize the use of certain resources (mainly the number of I/O and CPU time) by selecting a best execution plan (access plan). A query optimization start during the validation phase by the system to validate the user has appropriate privileges. Simply, generate an optimal evaluation plan (with lowest cost) for the query plan is called optimization.



3. Evaluation

The query-execution engine takes an (optimal) evaluation plan, executes that plan, and returns the answers to the query.

Query Optimization

The primary goal of query optimization is of choosing an efficient execution strategy for processing a query. DBMS provides two different approaches to query optimization: rule based and cost-based. With the rule-based approach, the optimizer chooses execution plans based on heuristically ranked operations. However, the rule-based approach is being phased out in favor of the cost-based approach, where the optimizer examines alternative access paths and operator algorithms and chooses the execution plan with the lowest estimated cost. The estimated query cost is proportional to the expected elapsed time needed to execute the query with the given

execution plan. The optimizer calculates this cost based on the estimated usage of resources, such as I/O, CPU time, and memory needed. The goal of cost-based optimization is to minimize the elapsed time to process the entire query.

Example: Transformation of an SQL-query into an RA-query and find optimized query

Employee (Eno, Ename, Title)

Project (Eno, Pno, Salary, Duration)

Query: Find the names of employees who are managing a project of duration greater than 5 years?

High level query:

SELECT Ename

FROM Employee E, Project P

WHERE E.Eno=P.Eno AND Duration>5;

Two possible transformations of the query are:

Expression 1: $\Pi_{\text{Ename}} (\sigma_{\text{E.Eno=P.Eno} \wedge \text{Duration}>5} (\text{Employee} \times \text{Project}))$

Expression 2: $\Pi_{\text{Ename}} (\text{Employee} \bowtie (\sigma_{\text{Duration}>5} (\text{Project})))$

Expression 2 avoids the expensive and large intermediate Cartesian product, and therefore typically is better.

Transaction Management

A transaction is a collection of several operations on the database appears to be a single unit from the point of view of the database user. For example, a transfer of funds from a checking account to a savings account is a single operation from the customer's standpoint; within the database system, however, it consists of several operations.

Database transaction is collection of SQL queries which forms a logical one task. For transaction to be completed successfully all SQL queries has to run successfully. Database transaction executes either all or none. For example, if your database transaction contains 4 SQL queries and one of them fails then change made by other 3 queries will be rolled back. This way your database always remain consistent whether transaction succeeded or failed.

Transaction is implemented in database using SQL keyword TRANSACTION, COMMIT and ROLLBACK.

- COMMIT writes the changes made by transaction into database
- ROLLBACK removes temporary changes logged in transaction log by database transaction.

Example:

Operation (Money Transfer)	T1
Read balance of account A1	Read (A1)
Subtract 20,000 from A1	$A1 = A1 - 20000$
Update balance of A1	Write (A1)
Read balance of account A2	Read (A2)
Add Rs 20,000 to A2	$A2 = A2 + 20000$

Table: Money Transfer Transaction

Why transaction is required in database?

Database is used to store data required by real life application e.g. Banking, Healthcare, Finance etc. All your money stored in banks is stored in database. In order to protect data and keep it consistent any changes in this data needs to be done in transaction so that even in case of failure data remain in previous state before start of transaction. Consider a Classical example of ATM (Automated Tailor Machine); we all use to withdraw and transfer money by using ATM. If you break withdrawal operation into individual steps you will find:

- Verify account details.
- Accept withdrawal request
- Check balance
- Update balance
- Dispense money

Suppose your account balance is 1000\$ and you make a withdrawal request of 900\$. At fourth step your balance is updated to 900\$ and ATM machine stops working due to power outage. Once power comes back and you again tried to withdraw money you surprised by seeing your balance just 100\$ instead of 1000\$. This is not acceptable by any person in the world) so we need transaction to perform such task.

Properties of Transaction

There are four important properties of database transactions these are represented by acronym ACID and also called ACID properties or database transaction where:

- **Atomicity:** Atom is considered to be smallest particle which cannot be broken into further pieces. Database transaction has to be atomic means either all steps of transaction completes or none of them.
- **Consistency:** Transaction must leave database in consistent state even if it succeed or rollback.

- **Isolation:** Two database transactions happening at same time should not affect each other and has consistent view of database. This is achieved by using isolation levels in database.
- **Durability:** Data has to be persisted successfully in database once transaction completed successfully and it has to be saved from power outage or other threats. This is achieved by saving data related to transaction in more than one places along with database.

Transaction States

Whenever a transaction is submitted to a DBMS for execution, either it executes successfully or fails due to some reasons. During its execution, a transaction passes through various states that are active, partially committed, committed, failed, and aborted.

- **Active state** - It is initial state. Transaction stays in this state while it is executing.
- **Partially committed state** - After the final statement has been executed, a transaction is in partially committed state.
- **Committed state** - After successful completion, a transaction is in committed state.
- **Failed state** - After the discovery that normal execution can no longer proceed, a transaction is in failed state.
- **Terminated State** – This state corresponds to the transaction leaving the system. The transaction information that is maintained in system tables while the transaction has been running is removed when the transaction terminates. Failed or aborted transactions may be restarted later – either automatically or after being resubmitted by the user – as brand new transactions.

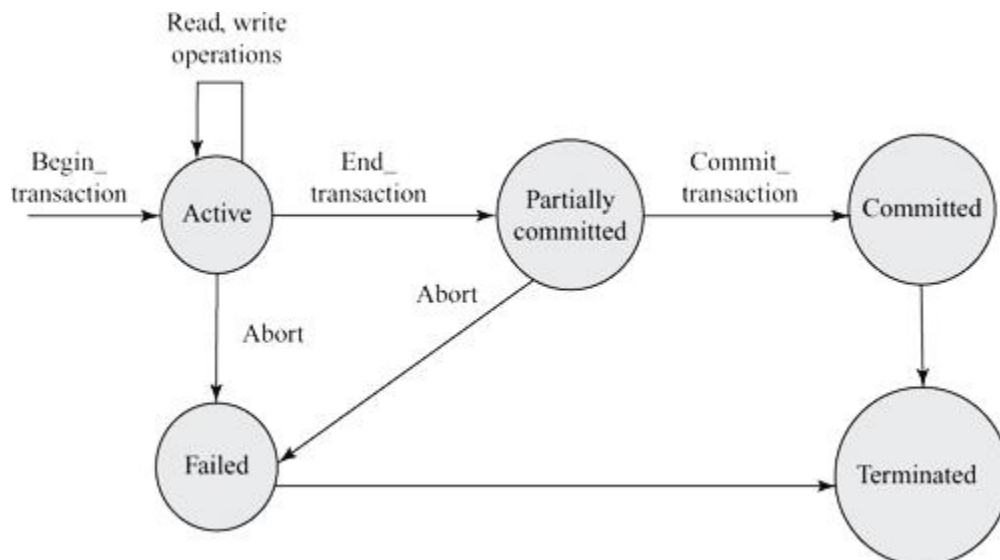


Fig: State diagram of Transaction

Concurrency Control

When multiple transactions are trying to access the same sharable resource, there could arise many problems if the access control is not done properly. There are some important mechanisms to which access control can be maintained. Earlier we talked about theoretical concepts like serializability, but the practical concept of this can be implemented by using Locks and Timestamps. Here we shall discuss some protocols where Locks and Timestamps can be used to provide an environment in which concurrent transactions can preserve their Consistency and Isolation properties.

Objectives of concurrency control mechanism can be list as below:

- To enforce Isolation (through mutual exclusion) among conflicting transactions.
- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.

Why concurrency Control needed?

If transactions are executed serially, i.e., sequentially with no overlap in time, no transaction concurrency exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Here are some typical examples:

- **The Lost Update Problem:** This problem occurs when two transactions that access the same database items have their operations interleaved in such a way that value of the data item written by one transaction is overlapped by another transaction and hence results to the incorrect value of the database item. Consider the example given below where two concurrent transactions try to update inventory of product P1 simultaneously.

Example: Assume initial value of data item P1 is 300

T1	T2	Value of Data Items
Read(P1)		T1.P1←300
P1=P1+100		T1.P1←300+100=400
	Read(P1)	T2.P1←300
	P1=P1+50	T2.P1←300+50=350
Write(P1)		T1 writes P1←400
	Write(P1)	T2 writes P1←350
Commit	Commit	

Table: Schedule LUP

This is incorrect value of P1. Problem was due to overwriting the value of P1 by T2. Correct value is 450.

- **The Dirty Read (Temporary Update) Problem:**

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is

changed back to its original value which causes transaction T2 to have dirty value of the data item. Again consider the example where two concurrent transactions try to update inventory of product P1 simultaneously. This example is same as above example except that its operations are interleaved in slightly different way and first transaction is failed.

Example: Assume initial value of data item P1 is 300

T1	T2	Value of Data Items
Read(P1)		T1.P1←300
P1=P1+100		T1.P1←300+100=400
Write(P1)		T1 writes P1←400
	Read(P1)	T2.P1←400
	P1=P1+50	T2.P1←400+50=450
		T1 is aborted (Undo)
Abort	Write(P1)	T1 writes P1←450
	Commit	

Table: Schedule DRP

This is incorrect value of P1. Problem was due to reading of dirty value of data item by T2. Correct value is 350.

Log based protocol for concurrency control

One way to ensure serializability is to access the data items in mutually exclusive manner. This means while one transaction is accessing data item, no other transaction should be allowed to access the data item. Lock variables are most commonly used approach for achieving mutual exclusion. Locks are of two kinds: Binary Locks and Shared/Exclusive Locks.

Binary lock

Binary lock is a variable that can be only in two states. It is either locked or unlocked. Normally, locked state is represented by value 1 and unlocked state is represented by value 0. A distinct lock is associated with each database item x. If the value of the lock on data item x is 1, item x cannot be accessed by a database operation that requests the item. If the value of the lock on x is 0, the item can be accessed when requested. Two operations, lock and unlock, are used with binary locking and these two operations must be implemented atomically.

If the simple binary locking scheme described above is used, every transaction must obey the following rules:

1. A transaction T must issue the operation lock(x) before performing any read(x) or write(x) operations.
2. A transaction T must issue the operation unlock(x) after finishing all read(x) and write(x) operations.
3. A transaction T will not issue a lock(x) operation if the data item x is already locked by it.
4. A transaction T will not issue an unlock(x) operation if the data item x is not locked by it.

Shared/Exclusive Lock

This type of lock is also called multiple mode lock. It is a variable that can be in any of three states: unlocked, read-locked (shared lock) and write-locked (exclusive lock). If a transaction acquires shared lock (read-lock) on data item x then other transactions can also acquire shared lock on data item x. But no transaction can acquire exclusive lock (write-lock) on data item x. On the other hand, if a transaction acquires exclusive lock (write-lock) on data item x then no other transactions can acquire shared/exclusive lock (read/write lock) on the data item.

When shared/exclusive locking scheme discussed above is used, the system must enforce the following rules:

1. A transaction T must issue the operation `read_lock(x)` or `write_lock(x)` before performing `read(x)` operation.
2. A transaction T must issue the operation `write_lock(x)` before performing `write(x)` operation.
3. A transaction T must issue the operation `unlock(x)` after finishing all its `read(x)` and `write(x)` operations.
4. A transaction T will not issue a `read_lock(x)` operation if it already holds a shared lock (read-lock) or exclusive lock (write-lock) on item x.
5. A transaction T will not issue a `write_lock(x)` operation if it already holds a shared lock (read-lock) or exclusive lock (write-lock) on item x.
6. A transaction T will not issue an `unlock(x)` operation if it does not hold a shared lock (read-lock) or exclusive lock (write-lock) on item x.

Two-Phase Locking Protocol

The two-phase locking protocol is used to ensure the serializability in Database. This protocol is implemented in two phase. *Growing Phase* and *Shrinking Phase*.

- **Growing Phase:** In this phase we put read or write lock based on need on the data. In this phase we does not release any lock. Remember that all lock operation must precede first unlock operation appeared in a transaction.
- **Shrinking Phase:** This phase is just reverse of growing phase. In this phase we release read and write lock but doesn't put any lock on data. Unlock operations can only appear after last lock operation.

For a transaction these two phases must be mutually exclusive. This means, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.

T1	T2	Data Items Values
Lock(x)		
Read(x)		x←50
x=x+100		x←150
Write(x)		T1 writes x←150
Lock(y)		
Unlock(x)		
	Lock(x)	
	Read(x)	x←150
	x=x*2	x←300
	<u>Write(x)</u>	T2 writes x←300
Read(y)		y=50
y=y+100		y=150
Write(y)		T1 writes y←150
Unlock(y)		
	Lock(y)	
	Unlock(x)	
	Read(y)	y←150
	y=y*2	y←300
	Write(y)	T2 writes y←300
	Unlock(y)	

Time-Stamp Based Protocol

The most commonly used concurrency protocol is time-stamp based protocol. This protocol uses either system time or logical counter to be used as a time-stamp. Lock based protocols manage the order between conflicting pairs among transaction at the time of execution whereas time-stamp based protocols start working as soon as transaction is created. Every transaction has a time-stamp associated with it and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transaction, which come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and priority may be given to the older one.

A timestamp can be implemented in two ways. The simplest one is to directly assign the current value of the clock to the transaction or the data item. The other policy is to attach the value of a logical counter that keeps incrementing as new timestamps are required.

The concurrency control algorithm must check whether conflicting operations violate the time stamp ordering in the following two cases.

Case I: Transaction T Issues Write(x) Operation

- If $ReadTS(x) > TS(T)$ or if $WriteTS(x) > TS(T)$, then abort and roll back T and reject the Operation. This should be done because some younger transaction with a time stamp greater than $TS(T)$ – and hence after T in the timestamp ordering – has already read or

written the value of item x before T had a chance to write x, thus violating the timestamp ordering.

- If the above condition is not satisfied then execute the Write(x) operation and set WriteTS(x) to TS(T).

Case II: Transaction T Issues Read(x) Operation

- If WriteTS (x) > TS(T), then abort and rollback and reject the operation. This should be done because some younger transaction have timestamp greater than TS (T) – and hence after T in the Timestamp ordering-has already written the value of item x before T had a chance to read x.
- If write TS(x) < = TS (T), then execute the Read(x) operation of T and set ReadTS(x) to the larger to TS(T) and the current ReadTS(x).

Example: Assume timestamps of T1 and T2 is 100 and 110 respectively and initial value of x is 500

T1	T2	Timestamps
Read(x)	Read(x)	ReadTS(x)←100 ReadTS(x)←110
x=x+200	x=x+200	
Write(x)	Write(x)	WriteTS(x)←110 WriteTS(x)>TS(T1) and hence Timestamp order violated
Abort T1		

Database performance tuning

Although newer relational databases and faster hardware run most SQL queries with a significantly small response time, there is always room for improvement. After a database is deployed and is in operation, actual use of the applications, transactions, queries, and views reveals factors and problem areas that may not have been accounted for during the initial physical design. Thus database tuning is a process of minimizing response time or improving performance by using various optimization techniques during designing, querying, transactions etc.

The goals of tuning are as follows:

- To make applications run faster.
- To improve (lower) the response time of queries and transactions.
- To improve the overall throughput of transactions.

Tuning a database involves dealing with the following types of problems:

- How to avoid excessive lock contention, thereby increasing concurrency among transactions.
- How to minimize the overhead of logging and unnecessary dumping of data.
- How to optimize the buffer size and scheduling of processes.

- How to allocate resources such as disks, RAM, and processes for most efficient utilization.

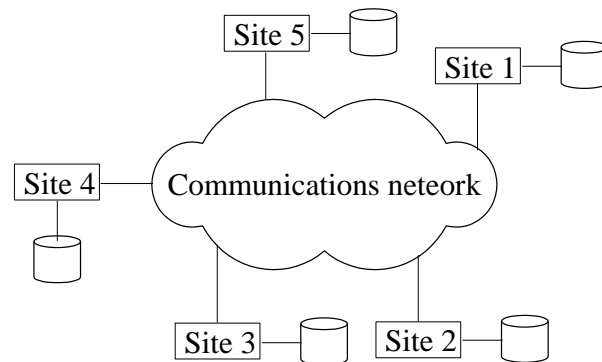
Tuning of database may occur in following sections:

- Tuning of Index
- Tuning of database design
- Tuning of query

Index tuning means reconstructing of index if overflow of data may occur in the table. Tuning of database design means reconstructing overall structure of database and keep in normal form if we need to insert additional requirements to the existing database system. And tuning of query means use less costly clauses during writing SQL of given statement.

Distributed Database

A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network. Distributed databases bring the advantages of distributed computing to the database management domain. It consists of a number of processing elements, not necessarily homogenous, that are interconnected by a computer network, and that cooperate in performing certain assigned tasks. As a general goal, distributed computing systems partition a big, unmanageable problem into smaller pieces and solve it effectively in a coordinated manner. It provides two major benefits: more computer power can be used to solve a complex task and each autonomous processing element can be managed independently and develop its own applications.



Distributed Database Management System

A distributed database management system (D-DBMS) is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users. It consists of a single logical database that is split into a number of fragments. Each fragment is stored on one or more computers under the control of a separate DBMS, with the computers connected by a communications network. Each site is capable of independently processing user requests that

require access to local data and is also capable of processing data stored on other computers in the network.

Functions of Distributed database system

- **Distribution and Network transparency**
Users do not have to worry about operational details of the network. There is Location transparency, which refers to freedom of issuing command from any location without affecting its working. Then there is naming transparency, which allows access to any names object (files, relations, etc.) from any location.
- **Replication transparency:** It allows to store copies of a data at multiple sites for better availability, performance, and reliability making the user unaware of the existence of copies.
- **Fragmentation transparency:** Allows to fragment a relation horizontally (create a subset of tuples of a relation) or vertically (create a subset of columns of a relation) making the user unaware of the existence of fragments.
- **Increased reliability and availability:** Reliability refers to system live time, that is, system is running efficiently most of the time. Availability is the probability that the system is continuously available (usable or accessible) during a time interval. A distributed database system has multiple nodes (computers) and if one fails then others are available to do the job.
- **Improved performance:** A distributed DBMS fragments the database to keep data closer to where it is needed most. This reduces data management (access and modification) time significantly.
- **Easier expansion (scalability):** Allows new nodes (computers) to be added anytime without chaining the entire configuration.
- **Autonomy:** Gives the users tighter control over their own local database.

Additional Functions

- **Keeping track of data:** The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDMS catalog.
- **Distributed query processing:** The ability to access remote sites and transmit queries and data among the various sites via a communication network.
- **Distributed transaction management:** The ability to devise execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data and maintain integrity of the overall database.

- **Replicated data management:** The ability to decide which copy of a replicated data item to access and to maintain the consistency of copies of a replicated data item.
- **Distributed database recovery:** The ability to recover from individual site crashes and from new type failures such as the failure of communication link.
- **Security:** Distributed transactions must be executed with the proper management of the security of the data and the authorization/access privileges of users.
- **Distributed directory (catalog) management:** A directory contains information (metadata) about data in the database. The directory may be global for the entire DDB, or local for each site. The placement and distribution of the directory are design and policy issues.

These above functions (in addition to those of a centralized DBMS) themselves increase the complexity of DDBMS over a centralized DBMS.

Data Fragmentation, Replication, and Allocation Techniques

Data Fragmentation: A process of splitting a relation into logically related and correct parts.

Fragmentation consists of breaking a relation into smaller relations or fragments, and storing the fragments (instead of the relation itself), possibly at different sites. In horizontal fragmentation, each fragment consists of a subset of rows of the original relation. In vertical fragmentation, each fragment consists of a subset of columns of the original relation. A relation can be fragmented in three ways:

- Horizontal Fragmentation
- Vertical Fragmentation
- Mixed Fragmentation.

Horizontal Fragmentation:

A horizontal fragment of a relation is a subset of the tuples in that relation. The tuples that belong to the horizontal fragment are specified by a condition on one or more attributes of the relation. Horizontal fragmentation divides a relation “horizontally” by grouping rows to create subsets of tuples, where each subset has a certain logical meaning. These fragments can then be assigned to different sites in the distributed system. Derived horizontal fragmentation applies the partitioning of a primary relation to other secondary relations which are related to the primary via a foreign key.

It is a horizontal subset of a relation which contains those of tuples which satisfy selection conditions specified in the SELECT operation of the relational algebra on single or multiple attributes. Consider the Customer relation with selection condition (sex= male). All tuples satisfy this condition will create a subset which will be a horizontal fragment of Customer relation.

Customer

<u>Customer id</u>	<u>Name</u>	<u>Area</u>	<u>Payment Type</u>	<u>Sex</u>
1	Nicky	Ktm	Cash	Female
2	Geeta	Lalitpur	Credit card	Female
3	Ruby	Ktm	Cash	Female
4	Aayan	Pokhara	Cash	Male
5	Aarav	Palpa	Credit card	Male
6	Mina	Pokhara	Cash	Female
7	Umesh	Chandani	Credit card	Male

Horizontal Fragmentation are subsets of tuples (rows)

$\sigma_{\text{sex}=\text{male}}(\text{customer})$

Fragment 1

<u>Customer id</u>	<u>Name</u>	<u>Area</u>	<u>Payment Type</u>	<u>Sex</u>
4	Aayan	Pokhara	Cash	Male
5	Aarav	Palpa	Credit card	Male
7	Umesh	Chandani	Credit card	Male

Fragment 2

$\sigma_{\text{sex}=\text{female}}(\text{customer})$

<u>Customer id</u>	<u>Name</u>	<u>Area</u>	<u>Payment Type</u>	<u>Sex</u>
1	Nicky	Ktm	Cash	Female
2	Geeta	Lalitpur	Credit card	female
3	Ruby	Ktm	Cash	female
6	Mina	Pokhara	Cash	female

Vertical Fragmentation:

Vertical fragmentation divides a relation “vertically” by columns. A vertical fragment of a relation keeps only certain attributes of the relation. It is a subset of a relation which is created by a subset of columns. Thus a vertical fragment of a relation will contain values of selected columns. There is no selection condition used in vertical fragmentation. All vertical fragments of a relation are connected by using PROJECT operation of the relational algebra.

Example:

<u>Customer_id</u>	<u>Name</u>	<u>Area</u>	<u>Payment_Type</u>	<u>Sex</u>
1	Nicky	Ktm	Cash	Female
2	Geeta	Lalitpur	Credit card	Female
3	Ruby	Ktm	Cash	Female
4	Aayan	Pokhara	Cash	Male
5	Aarav	Palpa	Credit card	Male
6	Mina	Pokhara	Cash	Female
7	Umesh	Chandani	Credit card	Male

Vertical fragmentation is subset of attributes

Fragment 1

<u>Customer_id</u>	<u>Name</u>	<u>Area</u>	<u>Sex</u>
1	Nicky	Ktm	Female
2	Geeta	Lalitpur	Female
3	Ruby	Ktm	Female
4	Aayan	Pokhara	Male
5	Aarav	Palpa	Male
6	Mina	Pokhara	Female
7	Umesh	Chandani	Male

Fragment 2

<u>Customer_id</u>	<u>Payment_Type</u>
1	Cash
2	Credit card
3	Cash
4	Cash
5	Credit card
6	Cash
7	Credit card

To combine all the vertically fragmented tables we need to perform join operation on the fragments.

```
SELECT customer_id, Name, Area, Sex, Payment_type
FROM Fragment 1 NATURAL JOIN Fragment 2;
```

Mixed (Hybrid) Fragmentation

We can intermix the two types of fragmentation, yielding a mixed fragmentation. The original relation can be reconstructed by applying UNION and OUTER UNION (or OUTER JOIN) operations in the appropriate order. In general a fragment of a relation can be specified by SELECT-PROJECT combination of operations which is represented by $\Pi_L(\sigma_C(R))$.

Data Replication and Allocation

Replication is useful in improving the availability of data. The most extreme case is replication of the whole database at every site in the distributed system, thus creating a fully replicated distributed database. This can improve availability remarkably because the system can continue to operate as long as at least one site is up. It also improves performance of retrieval for global queries because the results of such queries can be obtained locally from any one site; hence, a retrieval query can be processed at the local site where it is submitted, if that site includes a server module. The disadvantage of full replication is that it can slow down update operations drastically, since a single logical update must be performed on every copy of the database to keep the copies consistent. This is especially true if many copies of the database exist. Full replication makes the concurrency control and recovery techniques more expensive than they would be if there was no replication.

In partial replication some selected part is replicated to some of the sites. Data replication is achieved through a replication schema.

The process of assigning each fragment to a particular site in a distributed system is called data distribution (or **data allocation**). This is relevant only in the case of partial replication or partition. The selected portion of the database is distributed to the database sites. The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site. For example, if many updates are performed, it may be useful to limit replication.

Types of Distributed Database System

- Homogeneous distributed database system
- Heterogeneous distributed database system

If all servers (or individual local DBMSs) use identical software and all users (clients) use identical software, the DDBMS is homogenous; otherwise, it is heterogeneous.