UNIT I – DATABASE SYSTEMS AND DATA MODELING

Learning Objectives

At the end of the unit, the students will be able to

- Understand the concept of Database and Database Management System
- Compare various database models and database types
- List the Codd's rules of Relational model
- Explain the need for various keys in DBMS and create them
- Validate the database using constraints
- Draw E-R diagram
- Perform Normalization in 1NF, 2NF, 3NF
- Describe DBA tasks, tools and utilities
- Apply database backup and recovery

Introduction

Data is fundamental in Computer Engineering. Most of the Computer applications will read the data, process the data and produce required results. Data has to be maintained in a robust and efficient manner for any application to be productive. This necessitates the need for database and database management system.

In the early years of computing (1940's – 1950's), digital information is stored in punch card represented by the holes in predefined positions. With the evolution of hardware (1950's – 1960's), data are stored as flat files. Flat files are found inefficient and insecure in handling data. The concept of database evolved then. In 1960's – 1970's, the first type of DBMS, hierarchical DBMS, was developed in IBM for Apollo program, called as Information Management System (IMS). A network based database model called CODASYL (Conference on Data Systems Languages) was implemented in Honeywell. In 1972, the concept of relational DBMS was developed by E.F.Codd and this became the standard principle for database systems. In 1976,a new database model called Entity-Relationship, or ER, was proposed by P.Chen and was used in designing database. In 1980's, Structured Query Language (SQL) became the standard query language. In 1990's, object oriented approach is applied to databases called Object databases. In 2000's, XML databases were used as for its interoperability feature. A new type of database called "NoSQL" (Not Only SQL) is employed which handled database in means other than tabular relations used in relational databases. "NewSQL" is another class of modern RDBMS that provide NoSQL for
Online Transaction Processing(OLTP).

DATABASE SYSTEMS

Database Management System (DBMS)

Database is a collection of interrelated data stored in an organized manner. Database Management System is defined as a collection of interrelated data and a set of programs to create, access and maintain those data in an efficient manner. DBMS can be defined as a collection of databases and operations on the database.

Features of DBMS

(1) DBMS acts as a middle man between the user and application on one end and the database on the other end.
(2) It receives the request from the user, transfer it to the database, retrieves the required information from the database and send back the information to the user.
(3) It shares the database among multiple users and applications.

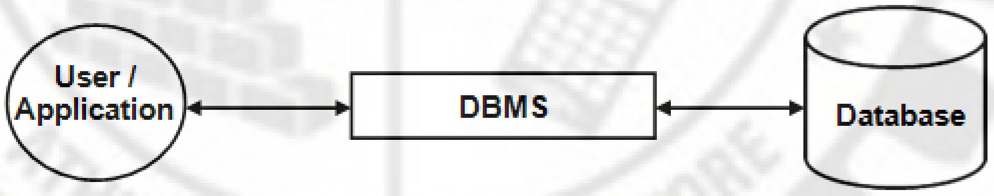


Fig. 1.1 - DBMS

Characteristics of Database

The important characteristics of database are

(1) Real World Entity
DBMS should be able to represent all kinds of data that exists in the real world as entities, attributes and relations. For example, a school database may use "student" as an entity and "name" as an attribute.

(2) Relation based Tables
DBMS should be able to relate entities or tables in the database by means of relation. That is, any two tables should be related.

(3) Isolation of data and application
Data and the application that access the data should be isolated to maintain data independence.

(4) Data Integrity
DBMS should be able to maintain the correctness and consistency of database, even during hardware and software malfunctions.

(5) Less Redundancy

Data should be stored in such a way that there is no repetition or minimum amount of repetition of data in the database.

(6) Query Language
DBMS should have a strong query language to retrieve and manipulate the data.

(7) Multi-user Access
Multiple users should be able to access the database without affecting other user.

(8) ACID properties
DBMS should support ACID properties, that is, Atomicity, Consistency, Isolation and Durability.

(9) Backup and Recovery
Database should be able to backup on separate medium using tools and techniques in DBMS. Database should be able to recover, if lost or damaged in unfortunate situation.

(10) Security
DBMS should provide security, so that data are protected from unauthorized users.

Components of Database

Themajor components of Database are

(1) User
(2) Database
(3) DBMS
(4) Database Application

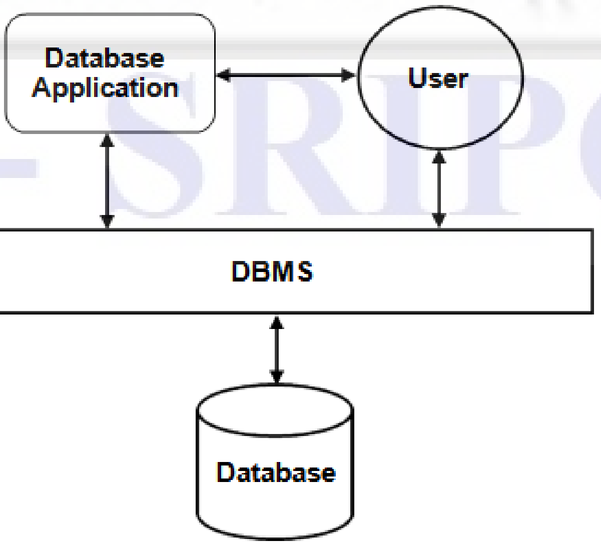


Fig. 1.2 – Components of Database

(1) User

User is the one who actually uses the database. User can be administrator, developer or the end user.

(2) Database

Database is the collection of related data stored in tables as rows and columns. There are two types of data

(i) User data
Data related to users which are stored, retrieved and manipulated.

(ii) Meta data
This is data about data. It stores information about how many tables, their names, how many columns, their names, keys, etc.

(3) DBMS

DBMS is the software that helps the user to interact with the database. It acts as middleman between the user and the database.

(4) Database Application

This is the application which helps the user to interact with the database by means of query language.

Functions of Database

The functions are database are given below

(1) Data Dictionary Management
DBMS stores meta-data in data dictionary. When end user needs a particular data, the DBMS uses the data dictionary and provides the data to the user.

(2) Data Storage Management
DBMS creates data structures required for data storage. The users are relieved from defining and implementing physical data structures.

(3) Security Management
DBMS protects the database by granting privileges to users. Unauthorized users are not allowed to access the database.

(4) Multi-user Access Control
This feature enables multiple users to access the database simultaneously without affecting the integrity of the database.

(5) Backup and Recovery Management
Backup is the process of copying the database into other storage medium. Recovery is the process of restoring the database from the backup.

(6) Data Integrity Management
Data integrity refers to the process of maintaining the correctness and consistency of data. DBMS enforce this through integrity constraints and rules.

(7) Database Access Language
DBMS use query language such as SQL (Structured Query Language) to access the database.

(8) Transaction Management
DBMS should guarantee all the transactions on the database should follow ACID (Atomicity, Consistency, Isolation and Durability) properties.

Applications of DBMS

DBMS is used in various applications. Some of them are listed below

(1) Banking
For maintaining information related to customers, accounts, loans, banking transactions, etc.

(2) Universities
For maintaining students records, staff records, course infor mation, fees, etc.

(3) Railways, Airlines
For maintaining train / flight information, timings, reservation details, etc.

(4) Telecommunication
To keep track of calls made, generate monthly bills, etc.

(5) Business
For storing and processing information related to products, sales, purchase, etc.

– DATABASE MODELS

Database Models

The logical design of the database is called “Database Model”. This is used to describe the data, their relationship, and how they are processed and stored inside the system.

Evolution of Database models

A database model is a specification that describes how a database is structured and used. Several database models have been proposed and evolved over the years. The common models are given below

(1) Flat database model
(2) Record based database model
(3) Object based database model

(1) Flat database model

The flat database model consists of a single, two-dimensional array of data elements. All the members of a given column are assumed to be similar values, and all the members of a row are assumed to be related to one another. For instance, database for username and password might consist of two columns, where the password is stored against the username.

(2) Record based database model

Record based models represent the structure of the database in fixed format records of several types. Each record type defines a fixed number of fields or attributes, and each field is of particular type and length.

The three most widely used record based data models are

- Hierarchical Model
- Network Model
- Relational Model

(3) Object based database model

In the 1990s, the object oriented programming paradigm was applied to database technology and calledas object based databases. Object databases utilize the key ideas of object programming, such as encapsulation and polymorphism, in the design of databases.

Types of Database models

The following are the commonly used types of database models.

(1) Hierarchical database model
(2) Network database model
(3) Relational database model

1.1.9.1 Hierarchical database model

This model represents database as a tree structure. In this model, the data are organized as records and records are stored in the forms of nodes of the tree from top to bottom. There is parent child relationship among various levels of the tree. The top most level record is called as root. Records are accessed by navigating downward from the root node using pointers.

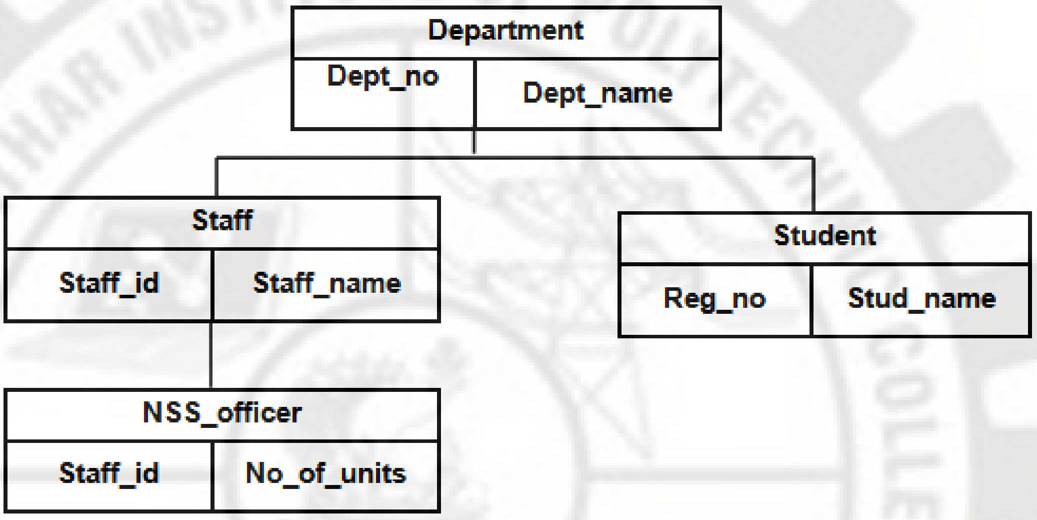


Fig 1.3 - Hierarchical database model

Advantages

(1) Simplicity

The design of the hierarchical model is simple.

(2) Data integrity

This model provides data integrity as there is parent child relationship between the records.

Disadvantages

(1) Maintenance is difficult.

(2) Programming is complex.

Network database model

This model represents the database as a graph. Each record is represented as a node. The nodes are connected according to their relationship. In this model, a record can have more than one parent. Records can be accessed through several paths.

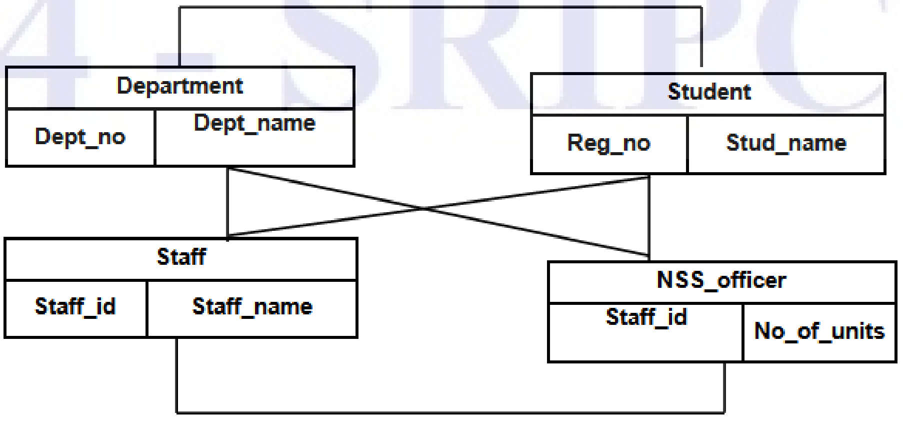


Fig 1.4 - Network database model

Advantages

(1) It is conceptually simple and easy to design.
(2) It supports one to many and many to many relationships.

Disadvantages

(1) The knowledge of the relationship between the records is required to design the database.
(2) There is no structural independence.

Relational database model

This model represents the database as a collection of two-dimensional tables called “relations”. The tables are related to each other. Each table is given a unique name and contains number of rows and columns of data. The column headings are called “attributes” or “fields” and rows are called “tuples” or “records”. The most common query language used with the relational model is the Structured Query Language (SQL).

Advantages

(1) Simplicity
It is easy to design, add, delete and change data.
(2) Structural Independence
The change in the structure of the database does not affect the data access.

Disadvantages

(1) It is difficult to store special data such as multimedia, digital data, spatial data, etc.
(2) If the number of tables in the database increases, performance decreases.

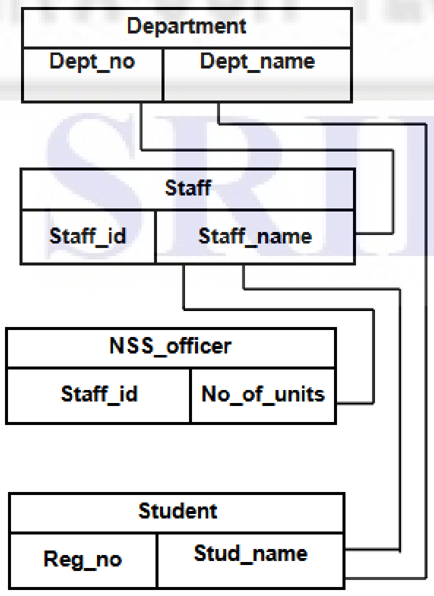


Fig. 1.5 - Relational database model

TYPES OF DATABASES

Transactional Database

A transactional database is a database management system (DBMS) that has the capability to roll back or undo a database transaction if it is not completed appropriately. The purpose of transactional database is to maintain the ACID (atomic, consistent, isolated and durable) principle in transactions.

1.2.1.1.Transactional Database Operations

Transactional Databaseuse SQL or a SQL-like language to conduct operations using the following pattern

Step 1: Begin the transaction.
Step 2: Execute the set of data queries or manipulations.
Step 3:If no errors occur, commit the transaction and end the operation.
Step 4: If errors occur, rollback the transaction and end the operation.

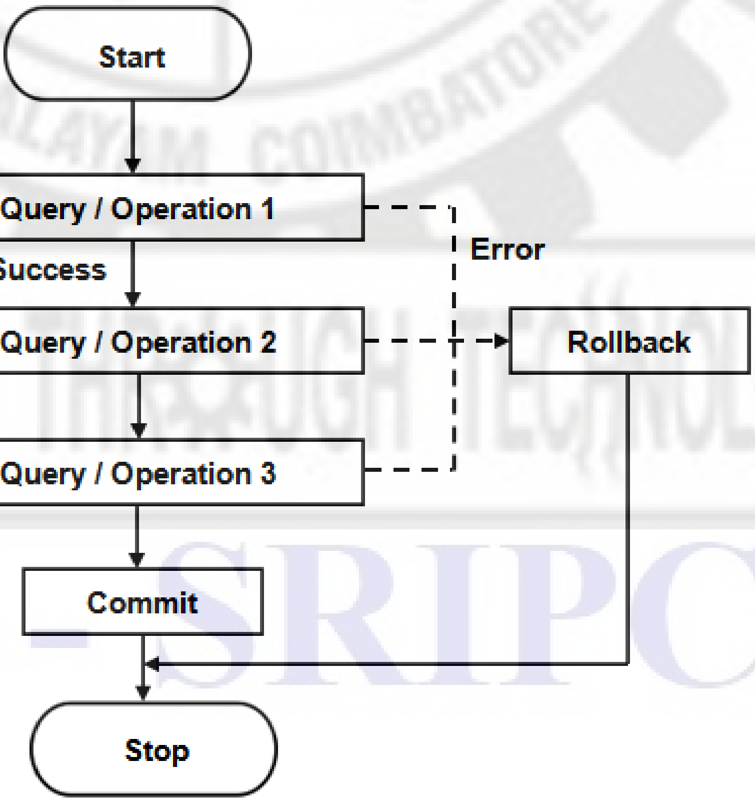


Fig. 1.6 - Transactional Database

Decision Support Database

Decision Support Database is a database from which data is extracted and analyzed statistically in order to inform business in taking decisions. For example, a decision support database might provide data to forecast which model of the product is fast moving in the market.

The two major technologies used in Decision support database are

1) Transaction Processing
2) Analytic Processing

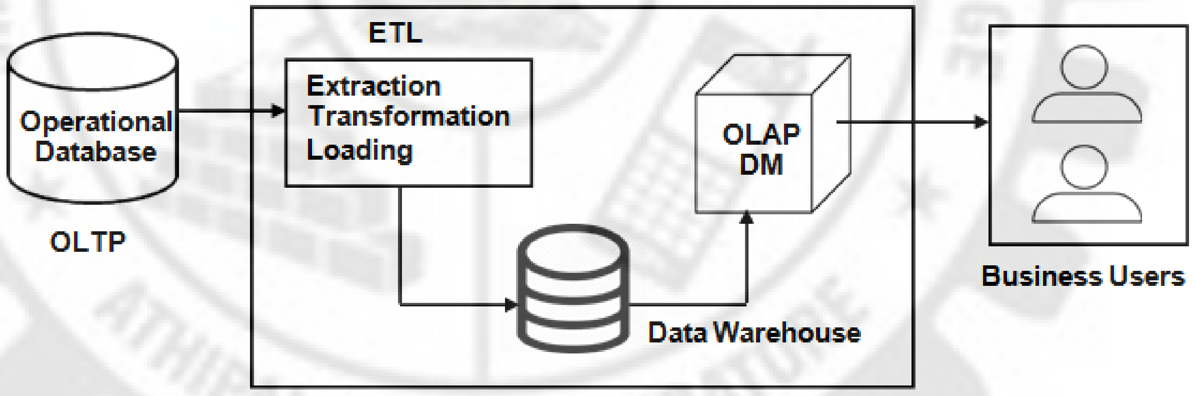


Fig. 1.7 - Decision Support Database

The data to be analyzed usually comes from operational database system through OLTP (Online Transactional Processing). Operational data is extracted, transformed and loaded into the data warehouse for further analyzing. OLAP (Online Analytical Processing) and Data Mining systems read these data, analyze them, produce useful reports and present them to end(business) users.

Hybrid Database

A hybrid database (HDB) is a database system that supports and uses both on-disk and in-memory data storage. It is a combination of in-memory database system (IMDB) and disk-based database system (DBMS). The hybrid database system use hard disks for long-term data storage and use in-memory for rapid access of dynamic data.

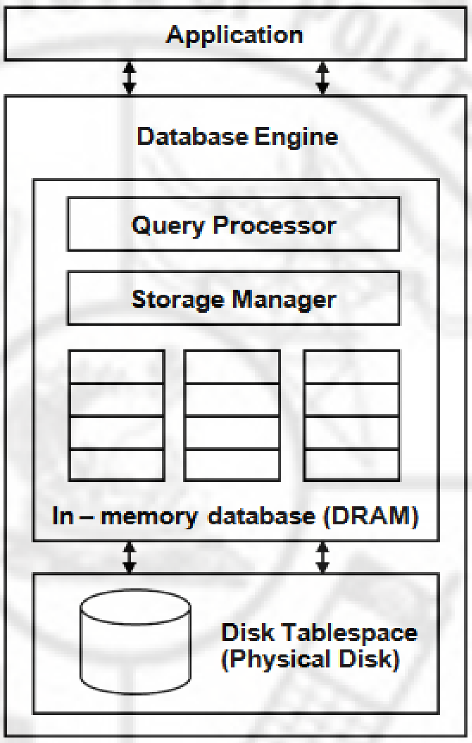


Fig. 1.8 - Hybrid Database

Open Source Database

An Open Source database is a database that includes Free and Open Source Software (FOSS). Open source software is software that makes the source code available to anyone. The user is allowed to implement, share and further develop the database software to suit various needs. Most commonly used open source database is MySQL. It runs on a majority of operating systems including UNIX, Linux, Mac and Windows. Some other open source database are SQLite, PostGreSQL, MongoDB, FireBird, etc.

– RELATIONAL DATA MODEL

CODD's Rules

E.F Codd was a Computer Scientist who invented Relational model for Database management. Codd proposed thirteen rules (numbered zero to twelve) and said that if a Database Management System meets these rules, it can be called as a Relational Database Management System. These rules are called as Codd's rules. They are

Rule Zero

The DBMS must be able to manage database entirely through relational capabilities.

Rule 1: Information rule

All information including metadata is to be represented as stored data in tables. Everything in a database must be stored in a table format.

.Rule 2: Guaranteed Access
Every single data element is guaranteed to be accessible with a combination of table-name, primary-key (row value), and attribute-name (column value).

Rule 3: Systematic Treatment of NULL Values
The NULL values in a database must be given a systematic and uniform treatment. NULL can be interpreted as one the following − data is missing / not known / not applicable.

Rule 4: Active Online Catalog
The structure of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users.

Rule 5: Comprehensive Data Sub-Language Rule
One well defined language must be there to provide all types of access to data. Example: SQL. If the data can be accessed without the help of this language, then it is considered as a violation.

Rule 6: View Updating Rule
All the views of a database, which can be theoretically updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule
A database must support high-level insertion, updation, and deletion. Set operations like Union, Intersection and minus should also be supported.

Rule 8: Physical Data Independence
Any change in the physical structure of the database must not affect the applications using it.

Rule 9: Logical Data Independence
Any change in logical structure of the database must not affect the applications using it.

Rule 10: Integrity Independence
Integrity constraints must be specified separately from application programs and stored in the catalog. All the integrity constraints should be independent of the application program.

Rule 11: Distribution Independence
A database should work properly regardless of its distribution across a network.
Rule 12: Non-Subversion Rule
If low level access is allowed to a system it should not be able to subvert or bypass integrity rule to change data.

Relational Database Management System (RDBMS)

Relational Database Management System (RDBMS) is a database management system based on relational model introduced by E.F.Codd. RDBMS stores data in the form of tables which are related to each other. Structured Query Language (SQL) is the language interface used in RDBMS for storing and retrieving data in the database. Some commonly used RDBMS software are MySQL, Oracle, MS SQL Server, DB2, etc.

Difference between DBMS and RDBMS

S.N	DBMS	RDBMS
1	In DBMS, data are stored as a collection of records, in the form of files.	In RDBMS, data are stored in the form of related tables.
2	DBMS does not support distributed databases.	Most of the RDBMS support distributed databases.
3	In DBMS, security levels are minimal.	In RDBMS, there are multiple levels of security.
4	Most of the DBMS support only single user.	RDBMS supports multiple users.
5	It supports 3 rules of E.F.Codd.	It supports minimum 6 rules of E.F.Codd.
6	Example : MS Access, dBase	Example : MySQL, Oracle

Components of RDBMS

The relational model is comprised of the following three main components

1) Data Structure
2) Data Integrity
3) Data Manipulation

Data Structure deals with the format for organizing and storing data in an efficient manner. Data Definition Language (DDL) commands are used to define the data structure.

Data integrity refers to the overall completeness, accuracy and consistency of data.

Data manipulation is the process of selecting, inserting, deleting and updating data in a database. The language or query used for data manipulation is called Data Manipulation Language (DML).

Table Structure

Table is a structure that contains many rows and columns in which the data are stored in RDBMS. A table is also called "relation" since the data in the table are related to each other.

Example
Table Name : employee

EMPNO	NAME	DEPT	SALARY
10	arun	sales	10000
20	babu	production	12000
30	chitra	accounts	14000

Record (or) Row (or) Tuple

The body of the table contains a number of rows with related data values. Each row of the table is called "record" or "tuple". For example, the table given below has 3numbers of tuples.

Example
Table Name : employee

EMPNO	NAME	DEPT	SALARY
10	arun	sales	10000
20	babu	production	12000
30	chitra	accounts	14000

Attribute

Attributes are column names in the table. An attribute gives the characteristics of the entity.

Example
Attributes in table "employee" are empno, name, dept, salary.

Domain

Domain is defined as the set of all unique values permitted for an attribute.

Example
The domain for "department" attribute in an organization may be production, sales, marketing, accounts, if only these four departments are available in the organization.

Degree

The number of attributes in a relation (table) is called degree of the relation.

Example
The degree of the table “employee” with four columns is four.

Cardinality

The number of tuples (rows) in a relation is called cardinality of the relation. The cardinality changes when rows are inserted or deleted.

Example
The cardinality of “employee” table containing 3 rows is three.

Meta data

A metadata is data about the data. It is also called as the System Catalog. It holds information about each data element in the database, such as name, type, range of values, source and access authorization.

Data dictionary

A data dictionary is a file or a set of files that contains a database's metadata. It is used to control database operations, integrity and accuracy.

Data Integrity

Data integrity is the maintenance of accuracy and consistency of data in the database over its entire life cycle. Data integrity is imposed through the use of standard rules and procedures.

1.3.4.. Keys

Key is defined as an attribute or group of attributes that is used to uniquely identify a row in a relation. The different types of keys are

(1) Primary key
(2) Composite key
(3) Unique key
(4) Foreign key
(5) Super key
(6) Candidate key

Primary key

Primary key is a column or a set of columns that is used to uniquely identify a row in a relation. The primary key column values should be unique and not null. If more than one column is used in primary key, it is called “**Composite key**”.

Syntax - Creating primary key with “create table” command

```
create table table_name ( column_name1 type, column_name2 type, … ,
                                        primary keykey_name (key_column) ) ;
```

where

table_name	-	name of the table
column_name	-	name of the columns
key_name	-	name of the primary key (optional)
key_column	-	column name(s) on which primary key is created

Example

mysql > create table employee (empno int , name varchar(40) , deptno int , salary int ,
primary key emp_pk (empno)) ;

Syntax - Creating primary key with "alter table" command

```
alter tabletable_nameadd primary keykey_name ( key_column
) ;
```

where

table_name	-	name of the table
key_name	-	name of the primary key (optional)
key_column	-	column name(s) on which primary key is created

Example

mysql > alter table employee add primary key emp_pk (empno) ;

Composite key

If more than one column is used in primary key, it is called "**Composite key**".

Syntax - Creating candidate key with "create table" command

```
create table table_name ( column_name1 type, column_name2 type, … ,
                    primary keykey_name (key_column1, key_column2) )
;
```

where

table_name	-	name of the table
column_name	-	name of the columns
key_name	-	name of the candidate key (optional)
key_column1 and 2	-	column name(s) on which composite key is created

Example

mysql > create table employee (empno int , name varchar(40) , deptno int , salary int ,
primary key emp_ck (empno,name)) ;

1.3.4.3 Unique key

Unique key is a column or a set of columns that is used to uniquely identify a row in a relation. It is similar to primary key in enforcing values to be unique, but allows null values.

Syntax - Creating Unique key with "create table" command

create table *table_name* (*column_name1 type, column_name2 type*, **… ,**
unique key*key_name* (*key_column*)) **;**

where

table_name	-	name of the table
column_name	-	name of the columns
key_name	-	name of the unique key (optional)
key_column	-	column name(s) on which unique key is created

Example
mysql > create table employee (empno int , name varchar(40) , deptno int , mobile int ,
unique key emp_uni (mobile)) ;

Syntax - Creating Unique key with "alter table" command

alter table*table_name***add unique key***key_name* (*key_column*) **;**

where

table_name	-	name of the table
key_name	-	name of the unique key (optional)
key_column	-	column name(s) on which unique key is created

Example
mysql > alter table employee add unique key emp_uni (mobile) ;

Foreign Key

A foreign key is a column or set of columns in one table whose values must have matching values in the primary key column of another table. Consider a table "t1" has a column "c1" which is the primary key in another table "t2". Then the column “c1” in table “t1” which refers to the primary key of table “t2” is called “foreign key”. Here, the table “t1” is called “child table” and the table “t2” is called “parent table”. The data type and size of the foreign key should be same as that of the referring primary key.

Syntax - Creating foreign key with “create table” command

create table *child_table* (*column_name1 type, column_name2 type*, **… ,**
foreign key*key_name* (*fkey_column*) **references** *parent_table* (*pkey_column*) **;**

where

child_table	-	name of the child table
parent_table	-	name of the parent table
key_name	-	name of the foreign key (optional)
fkey_column	-	foreign key column name in child table
pkey_column	-	primary key column name in parent table

Example

Creating parent table “department” with primary key

```
mysql > create table department (deptno int, dname varchar(40),
                primary key dept_pk (deptno) );
```

Creating child table “employee” with foreign key

```
mysql > create table employee (empno int , name varchar(40) , deptno int , salary int ,
                foreign key emp_fk (deptno) references department (deptno) ) ;
```

Syntax - Creating foreignkey with “alter table” command

alter table*child_table***add foreign key***key_name* (*fkey_column*)
references *parent_table (pkey_column)***;**

where

child_table	-	name of the child table
parent_table	-	name of the parent table
key_name	-	name of the foreign key (optional)
fkey_column	-	foreign key column name in child table
pkey_column	-	primary key column name in parent table

Example

```
mysql > alter table employee add foreign key emp_fk(deptno)
                references department (deptno) ;
```

Super key

Super key is a column or a set of columns that is used to uniquely identify a row in a relation. A relation may have number of super keys. Super keys are formed by adding other fields to primary key.

Candidate key

Candidate key is a column or a set of columns that is used to uniquely identify a row in a relation. Candidate keys are super keys which are not having any redundant attributes. Candidate keys are minimal super keys.

Data Constraints

Data constraints are policies to maintain accuracy and integrity of data in the database during database operations. Constraints are enforced to guarantee that the values of the table are always valid and they obey rules specified in the database scheme.

Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

Types of Constraints

Relational model has the following types of constraints.

(1) Entity integrity constraint
- (i) Unique constraint
- (ii) Primary key constraint

(2) Domain integrity constraint
- (i) Not null constraint
- (ii) Default constraint

(3) Referential integrity constraint (or) Foreign key constraint

Entity Integrity constraint

Entity Integrity constraint is used to enforce unique key and primary key in the database. The following are the two types of Entity integrity constraints.

(i) Unique constraint
(ii) Primary key constraint

(i) Unique constraint

Unique constraint ensures that each row for a column or a set of columns must have a unique value. But null values are allowed for that column. If this constraint is assigned to more than one column, it is called “composite unique key”.

<u>Creating unique constraint</u>

Unique constraint can be created as column constraint or table constraint in "create table" command.

Unique constraint as column constraint

Syntax

create table *table_name* (*column_name1 type* **unique**, *column_name2 type*, **…**) **;**

Example

```
mysql >create table student1(regno int unique, name varchar(40), dob date);
```

Unique constraint as table constraint

Syntax

create table *table_name* (*column_name1 type, column_name2 type*, **… ,** **constraint** *constraint_name* **unique** (*column_name*)) **;**

Example

```
mysql >create table student2 ( regno int, name varchar(40), dob date,
           constraint c1 unique(regno) );
```

(ii) Primary key constraint

Primary key constraint ensures that each row for a column or a set of columns must have a unique value. Here, null values are not allowed. If this constraint is assigned to more than one column, it is called "composite primary key".

Creating primary key constraint

Primary key constraint can be created as column constraint or table constraint in "create table" command.

Primary key constraint as column constraint

Syntax

create table *table_name* (*column_name1 type* **primary key**, *column_name2 type*, **…**) **;**

Example

```
mysql >create table student1 (regno int primary key, name varchar(40), dob date);
```

Primary key constraint as table constraint

Syntax

create table *table_name* (*column_name1 type, column_name2 type*, **… ,** **constraint** *constraint_name* **primary key**(*column_name*)) **;**

Example

```
mysql >create table student2 ( regno int, name varchar(40), dob date,
```

constraint c1 primary key (regno));

Domain Integrity constraint

Domain Integrity constraint is used to verify whether the data entered is in proper form and range. The following are the two types of Domain integrity constraints.

(i) Not null constraint
(ii) Default constraint

(i) Not null constraint

Not null constraint ensures that a column cannot have "NULL" value. This is used to make the user to enter values for the column compulsory.

<u>Creating not null constraint</u>

<u>Syntax</u>

create table *table_name* (*column_name1 type* **not null**, *column_name2 type*, **…**) **;**

<u>Example</u>

mysql >create table student1 (regno int primary key, name varchar(40) not null, dob date);

(ii) Default constraint

Default constraint provides a default value for a column. The default value is used if no value is specified for that field when inserting a record.

<u>Creating default constraint</u>

<u>Syntax</u>

create table *table_name* (*column_name1 type* **default** *value, column_name2 type*, **…**) **;**

<u>Example</u>

mysql >create table student1 (regno int, name varchar(40), address varchar(40) default 'chennai');

(3) Referential Integrity constraint (or) Foreign key constraint

Referential integrity constraint or foreign key constraint ensures that a column or set of columns in one table must have matching values in the primary key column of another table. The referring table is called child table and the referred table is called child table.

<u>Syntax - Creating foreign key with "create table" command</u>

create table *child_table* (*column_name1 type, column_name2 type*, **… ,**
foreign key*key_name* (*fkey_column*) **references** *parent_table* (*pkey_column*) **;**

Example

Creating parent table “department” with primary key

```
mysql > create table department (deptno int, dname varchar(40),
                primary key dept_pk (deptno) );
```

Creating child table “employee” with foreign key

```
mysql > create table employee (empno int , name varchar(40) , deptno int , salary int ,
                foreign key emp_fk (deptno) references department (deptno) ) ;
```

Difference between SQL and MYSQL

	SQL	MySQL
1	SQL stands for Structured Query Language.	Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and SQL.
2	SQL is language specification for accessing and manipulating databases standardized by ANSI / ISO.	MySQL is a database management system and it is one implementation of SQL specification.
3	It is fixed in nature and does not undergo any changes.	MySQL undergoes regular updates.
4	SQL is a standard made up of multiple components including SQL Framework, SQL/CLI and SQL/XML.	MySQL is used in popular web applications as a means of storage and logging user data.

– ER DIAGRAM & NORMALIZATION

Methodologies of Designing Database

Database design methodology is an approach taken in designing and building database. It is step by step procedure used to support and facilitate designers in planning, modeling and managing a database development. The methodology consists of three main phases namely conceptual, logical and physical database design phase.

Conceptual database design phase

The conceptual database design is aimed to produce a conceptual representation of the required database. The core activity in this phase involves the use of ER modeling in which the entities, relationship and attributes are defined.

Logical database design phase

This phase is aimed to map and validate the conceptual model that has been created in the conceptual phase on to the logical structure of the database. The main activity in this phase is the use of normalization process to derive and validate relations.

Physical database design phase

This phase is aimed to translate the logical structure to the physical implementation of the database using the defined database management system. This phase works on base relations, storage structures, access methods and security mechanism.

E-R Model (Entity – Relationship model)

Entity Relationship (E-R) model is a high level conceptual model developed by Chan in 1976. This model describes the structure of the database and the operations on the database in a pictorial representation. The components of E-R model are

(1) Entity
(2) Relationship
(3) Attribute

1.4.3.E- R Diagram

The overall logical structure of the database can be expressed graphically by E-R diagram. The following symbols are used to draw E-R diagram

(1) Rectangle	-	to represent entity set
(2) Diamond	-	to represent relationship set
(3) Ellipse	-	to represent attribute
(4) Underline	-	to represent primary key

Example

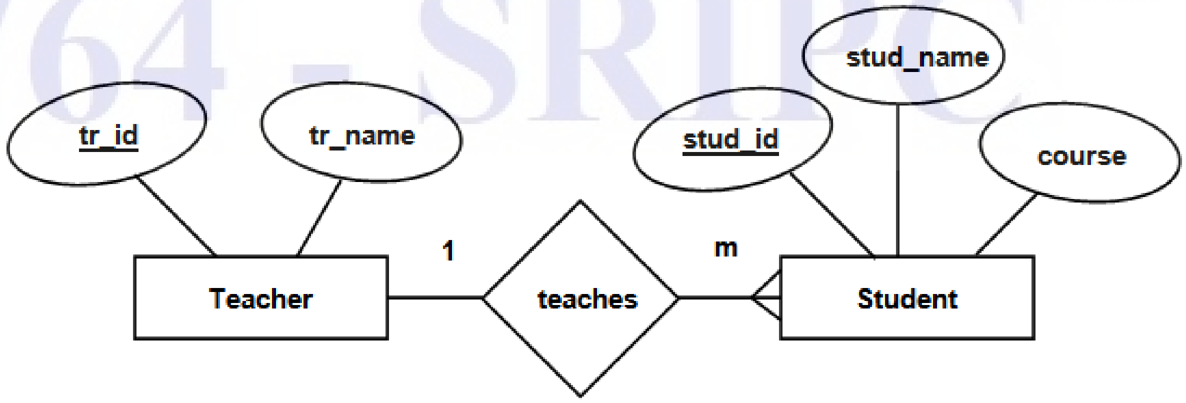


Fig. 1.9 - E- R Diagram

In this example,

Entity	-	Teacher, Student
Relationship	-	teaches (one to many relationship)
Attribute	-	tr_id, tr_name, stud_id, stud_name, course (tr_id and stud_id are key attributes)

<u>Advantages of E – R Diagram</u>

(1) E-R diagrams can be directly translated into database tables.
(2) E-R diagrams are used to represent different relationships between tables.
(3) E-R diagrams are easy to understand.

<u>Disadvantages of E – R Diagram</u>

(1) It is not possible to represent all the relationships.
(2) It is not possible to represent all the constraints.

Entity

An entity is an object or a thing such as person, place, concept, etc. Entity has a set of properties or attributes. Entities are represented by a “rectangle” symbol with the name of the entity.

<u>Example</u>

Relationship

Relationship is defined as an association between the entities. Relationships are represented by “diamond” symbol with the name of the relationship. The following are the different types of relationship among the entities

(1) one to one
(2) one to many
(3) many to one
(4) many to many

<u>(1) one to one relationship</u>

A relationship between one entity with only one entity is called “one to one” relationship. This is represented symbolically as 1 : 1.

<u>Example</u>

Relationship between student and register number.

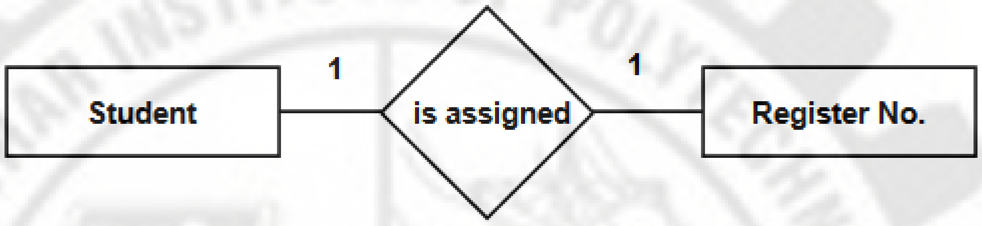


Fig. 1.10 - one to one relationship

(2) one to many relationship

A relationship between one entity with many entities is called "one to many" relationship. This is represented symbolically as 1 : m.

Example

Relationship between teacher and students.

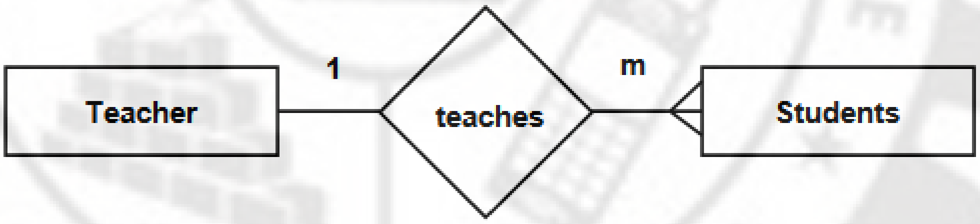


Fig. 1.11 - one to many relationship

(3) many to one relationship

A relationship between many entities with one entity is called "many to one" relationship. This is represented symbolically as m : 1.

Example

Relationship between students and course.

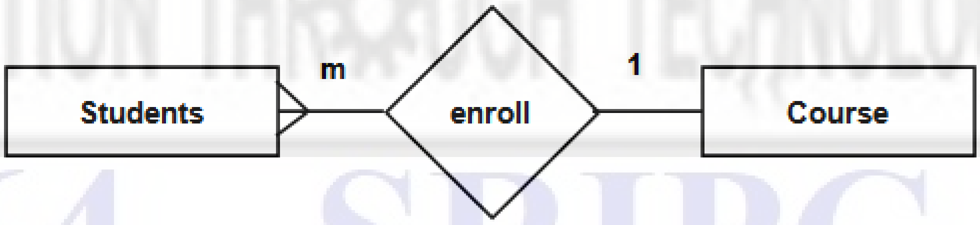


Fig. 1.12 - many to one relationship

(4) many to many relationship

A relationship between many entities with many entities is called "many to many" relationship. This is represented symbolically as m : n.

Example

Relationship between students and games.

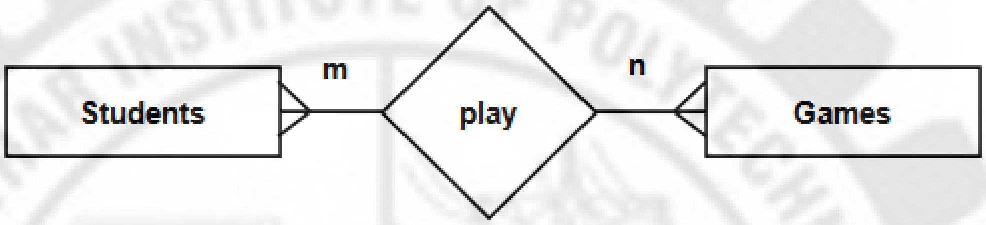


Fig. 1.13 - many to many relationship

Attribute

Attributes are the characteristic properties of an entity. It is the name of the column in the relation. Attributes are represented by "ellipse" symbol.

<u>Types of Attributes</u>

<u>(1) Single valued Attribute</u>

An attribute that has a single value for a particular entity is known as single valued attribute. For example, "age" attribute of employee entity.

<u>(2) Multi valued Attribute</u>

An attribute that can have multiple values for the same entity is known as multi valued attributes. For example, "sports" attribute of student entity.

<u>(3) Compound Attribute / Composite Attribute</u>

Compound / Composite attribute is the one that can be subdivided into two or more other attributes. For Example, "name" attribute can be divided into First name, Middle name and Last name.

<u>(4) Simple Attribute / Atomic Attribute</u>

The attribute which cannot be divided into smaller subparts are called simple or atomic attribute. For example, "designation" attribute of employee entity.

<u>(5) Stored Attribute</u>

An attribute which cannot be derived from other attributes is known as stored attribute. For example, "BirthDate" of employee.

<u>(6) Derived Attribute</u>

Derived attribute is the one that can be derived from other stored attributes. For example, "age" attribute can be derived from "BirthDate" attribute.

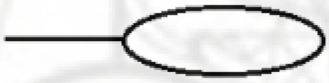
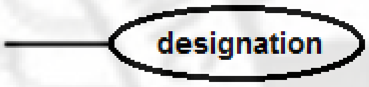
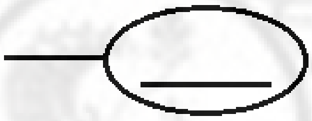
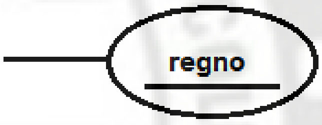
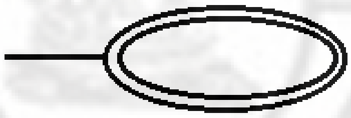
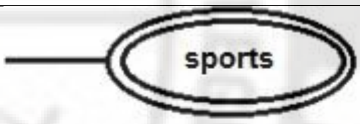
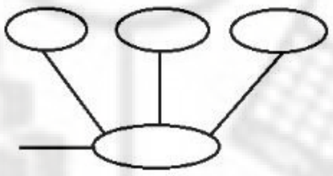
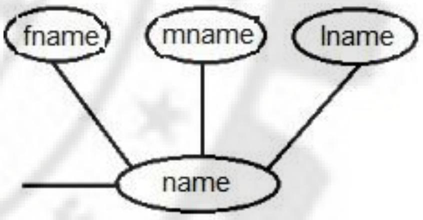
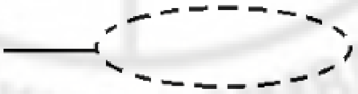
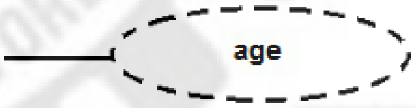
<u>(7) Key Attribute</u>

Key attribute represents primary key or candidate key of the entity. It has distinct value for each entity in an entity set. For example, "regno" attribute in student entity.

<u>(8)Non Key Attribute</u>

These are attributes other than candidate key attributes in a table. For example, “age” is a non key attribute in employee table.

Example – Attribute representation

S.N	Attribute type	Symbol	Example
1	Simple attribute		
2	Key attribute		
3	Multi valued attribute		
4	Compound / Composite attribute		
5	Derived attribute		

Sample E-R Diagrams

E-R diagram for Customer &Loan

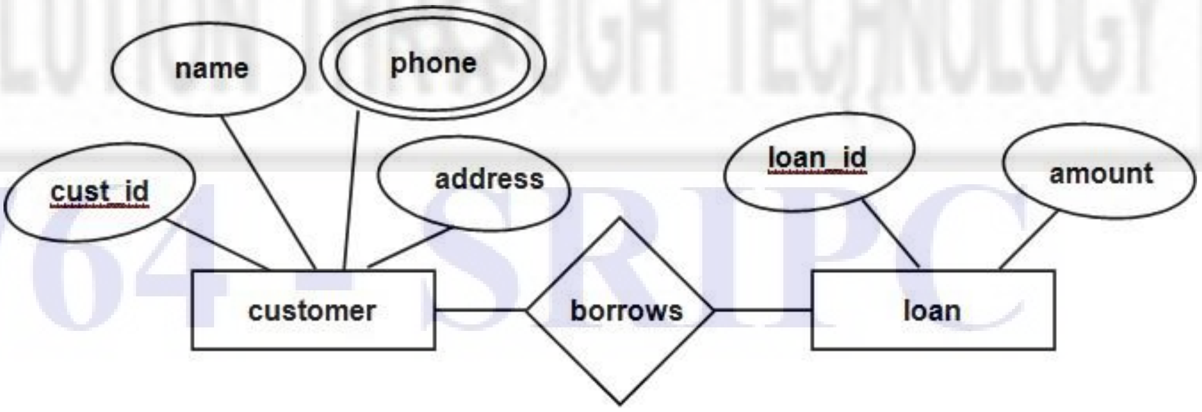


Fig. 1.14–Customer & Loan

E-R diagram for Library Management System

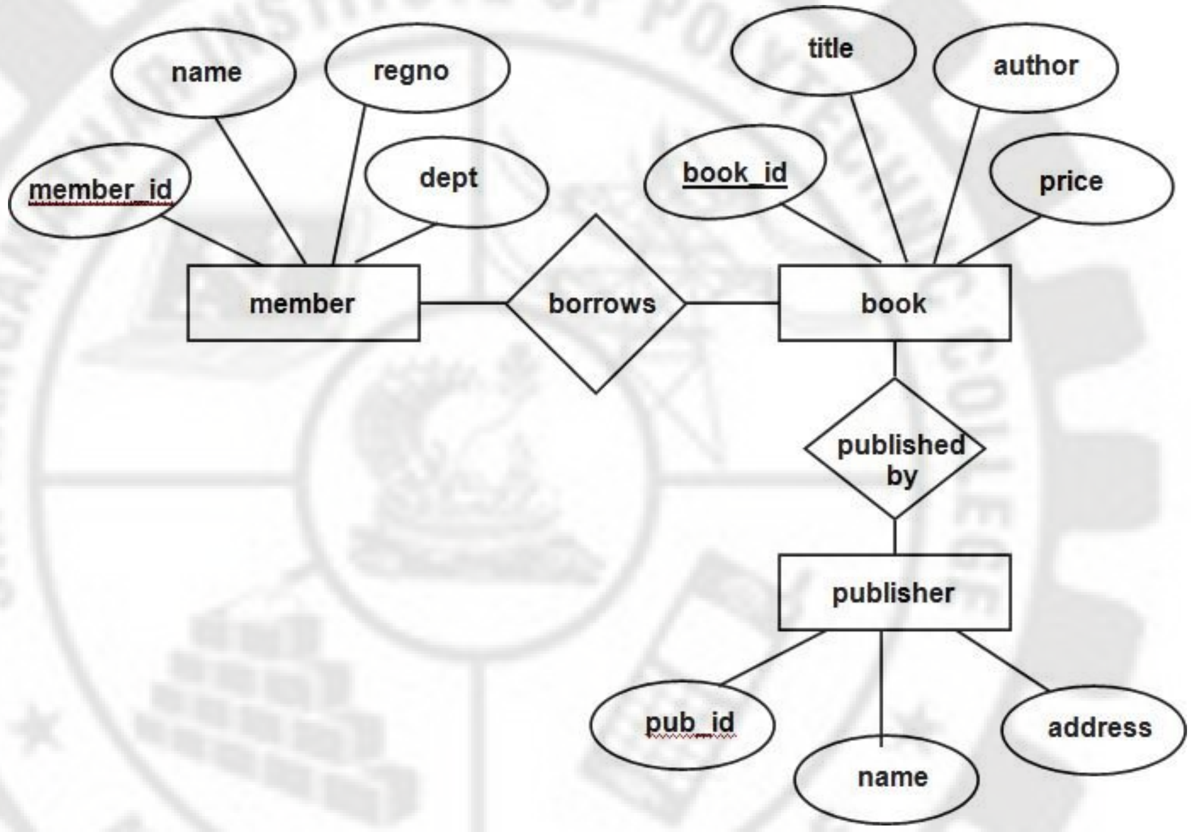


Fig. 1.15– Library Management System

Normalization

Normalization is a database design technique to reduce redundancy and inconsistency of data by splitting larger tables into smaller tables and defining relationship between them. The process of normalization was proposed by E.F.Codd. Later he joined with Raymond Boyce to develop Boyce-Codd Normal Form.

Benefits of Normalization

- Normalization is used to remove inconsistency in the database.
- It is used to reduce redundancy.
- It will remove anomalies with respect to insert, delete and update of data.
- It is used to design the database close to real world entities and relationships.

Types of Normalization

The following are the different types of Normalization.

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF)
4. Boyce Codd Normal Form (BCNF)
5. Fourth Normal Form (4NF)

6. Fifth Normal Form (5NF)

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – insertion, updation and deletion anomaly. This can be explained by taking an example.

Consider a table “company”

emp_id	emp_name	emp_address	proj_id	proj_name	proj_months
E1	arun	puducherry	P1	bank	24
E1	arun	puducherry	P2	school	12
E2	babu	cuddalore	P3	web	6

Here, (emp_id, proj_id) is the primary key.

(i) Insertion anomaly

Suppose a new project “railways” is started with project id “P4” and currently no employee is assigned in the project, that record cannot be inserted.

(ii) Updation anomaly

Suppose, the address of employee “arun” is changed, we need to change in all the records of the employee; otherwise data will be inconsistent.

(iii) Deletion anomaly

Suppose the employee “babu” leaves the company. We cannot delete his record, since the information related to the project “web” will also be deleted.

First Normal Form (1NF)

A relation is said to be in “first normal form”,

- if every record is unique
- if every attribute of the table is atomic, that is, the values should not be multi-valued or composite.

Multi-valued attribute

Multi-valued attribute is the attribute that can have more than one value in a single record. Consider the table “employee” where an employee can work in more than one project as given below

emp_id	emp_name	proj_name
1	arun	bank, school
2	babu	web

Here, “proj_name” attribute is multi-valued and the table is not in 1NF. To conform to 1NF, the table is split into two as given below

“employee1”

emp_id	emp_name
1	arun

2	babu

"employee2"

emp_id	proj_name
1	bank
1	school
2	web

Composite attribute

Composite attribute is an attribute that follows some structure and can be split further into meaningful data.
Consider the following table "employee"

emp_id	emp_name	address
1	arun	10, gandhi street, puducherry
2	babu	20, nehru street, cuddalore

Here, "address" attribute is composite attribute and hence the table is not in 1NF.
To conform to 1NF, the structure of the table is changed as given below

emp_id	emp_name	door_no	street	city
1	arun	10	gandhi street	puducherry
2	babu	20	nehru street	cuddalore

Second Normal Form (2NF)

A relation is said to be in "second normal form" if both the following conditions hold

- table is in 1NF
- if every non-key attribute is dependent on the entire primary key.

Consider a table "company"

emp_id	emp_name	emp_address	proj_id	proj_name	proj_months
E1	arun	puducherry	P1	bank	24
E1	arun	puducherry	P2	school	12
E2	babu	cuddalore	P3	web	6

The table is in 1NF because each attribute has atomic values. Here, (emp_id, proj_id) is the primary key. The non-key attributes emp_name and emp_address are dependent only on emp_id and not on proj_id. The non-key attributes proj_name and proj_months are dependent only on proj_id and not on emp_id. So the table is not in 2NF. To conform to 2NF, the table is split into 3 tables as follows

Table "employee"

emp_id	emp_name	emp_address
E1	arun	puducherry
E2	babu	cuddalore

Table “project”

proj_id	proj_name	proj_months
P1	bank	24
P2	school	12
P3	web	6

Table emp_project

emp_id	proj_id
E1	P1
E1	P2
E2	P3

Now, the tables are in 2NF.

Third Normal Form (3NF)

A relation is said to be in “third normal form” if both the following conditions hold

- table is in 2NF
- it has no transitive functional dependency of non-key attribute on super key, that is, every non-key attribute is directly dependent on the entire primary key.

Transitive dependency is dependency as follows

If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Consider table “employee”

emp_id	name	city	state
E1	arun	puducherry	puducherry
E2	babu	cuddalore	tamilnadu
E3	chitra	chennai	tamilnadu
E4	daniel	chennai	tamilnadu

The primary key is “emp_id”. Here, the non-key attributes “name” and “city” are directly dependent on the primary key “emp_id”; the non-key attribute “state” is dependent on “city” and transitively dependent on “emp_id”. This violates the rule of 3NF. To conform to 3NF, the table is split into two table as follows.

Table “employee”

emp_id	name	city
E1	arun	puducherry
E2	babu	cuddalore
E3	chitra	chennai
E4	daniel	bangalore

Table “city_state”

city	state
puducherry	puducherry
cuddalore	tamilnadu
chennai	tamilnadu

DATABASE ADMINISTRATION

Client Server technology

The Client Server technology is popular technology used for distributing data on the networks. The important components of this technology are

(i) Server
(ii) Client

(1) Server

Server is a powerful machine with multi-user operating system. Many users can access the Server at the same time. Software and data that are to be shared among different users are stored in this machine. This machine will be controlled by System Administrator or MIS (Management Information System) staff.

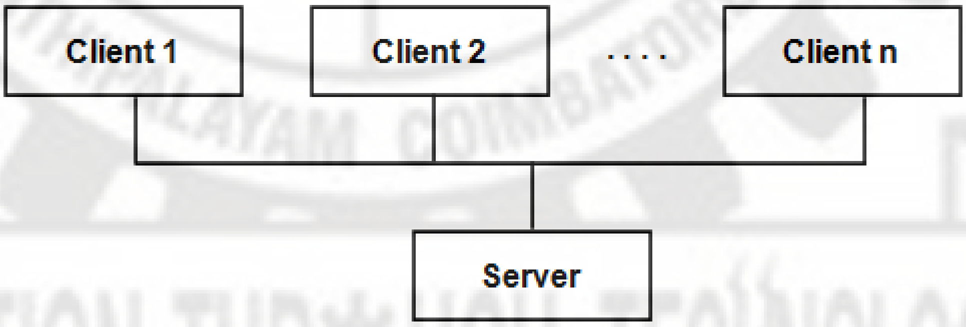


Fig. 1.16– Client Server Technology

(2) Client

Client is a low-end computer used by individual user. The client can access data and software stored in the server. The client requests the server and the server responds to the client.

<u>Advantages</u>

1. Centralization
   There is centralized control of resources and data.
2. Scalability
   Any component can be upgraded when needed.
3. Back-up and Recovery
   As all the data is stored on server, it is easy to back-up and re-cover at need.
4. Security

Servers have better security mechanisms to ensure that only authorized clients can access and manipulate data.

Disadvantages

1. Overloaded Server
   When there are frequent and simultaneous client requests, servers get overloaded resulting in performance degradation.
2. Impact of centralized architecture
   If the server fails, the whole network goes down and the client requests cannot be accomplished.

Distributed System

In distributed system, the data and software are stored on several computers. These computers vary in size and configuration, ranging from workstations to mainframes. The computers communicate with one another through high speed networks or through telephone line. The computers are referred by different names such as “sites” or “nodes”.

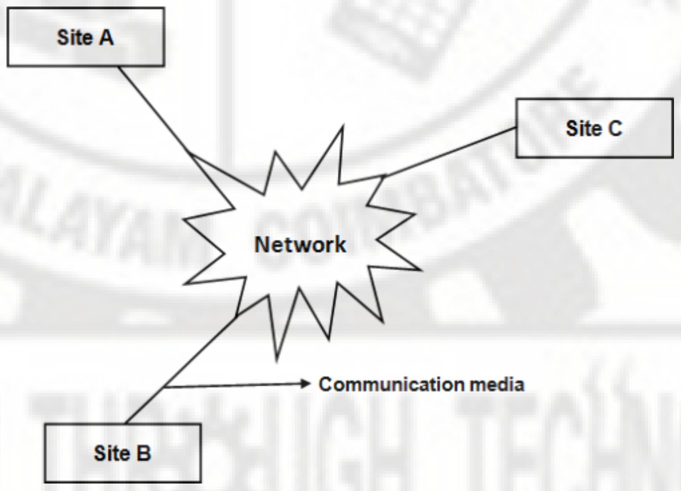


Fig. 1.17– Distributed System

There are two types of transactions in distributed system

(1) Local transaction
    If the user access data in a single site, it is called local transaction.
(2) Global transaction
    If the user access data in multiple sites, it is called global transaction.

Advantages

1. Sharing data
   User at one site shall be able to access data residing at other sites.
2. Availability of data
   If one site fails, the other sites shall continue operating.

Disadvantages

1. Increased cost
   Procurement and maintenance cost is higher than centralized system.
2. Security overhead
   Security in distributed system cannot be enforced strictly.

DBA Tasks

DBA stands for Database Administrator. DBA is a role in IT department who is in charge of the DBMS. The following are the important tasks of DBA

(1) Install, configure and upgrade database software and related products.
(2) Create and maintain the database.
(3) Starting and shutting down the database instance
(4) Manage schema objects, such as tables, indexes, and views.
(5) Enroll users and granting access rights and privileges.
(6) Database tuning and performance monitoring.
(7) Establish and maintain sound backup and recovery policies and procedures.
(8) Implement and maintain database security.
(9) Coordinate with DBMS vendor and plan for changes.
(10) Generate various reports by querying from database as per need

DBA Tools and Utilities

DBA tools and utilities are programs and software that are used to perform database administration tasks. Various tools are used for each type of DBMS. Some popular tools and utilities for different DBMS are discussed below

SQL Server based tools

(1) Microsoft SQL Server Management Studio (SSMS)
    SQL Server Management Studio is a software application used for configuring, managing, and administering all components within Microsoft SQL Server.
(2) Microsoft SQL Server Data Tools (SSDT)
    It is a modern development tool to build SQL Server databases.

Oracle based tools

(1) Oracle Enterprise Manager Database Control
    It is a web based tool to manage and administer oracle database.
(2) Oracle SQL Developer
    It is an IDE for working with SQL in Oracle databases. It is provided free from Oracle Corporation.

MySQL based tools

(1) MySQL workbench

It is a visual database design tool that integrates development, administration, database design, creation and maintenance for MySQL database.

(2) Php MyAdmin

It is a web based tool for MySQL DBMS. This can be used to create database, tables, users, run queries, export / import data and much more.

<u>Common tools</u>

(1) DB Visualizer Free

It is the universal database tool for developers, DBAs and analysts. It can be used on all major operating systems accessing a wide range of databases.

(2) DBeaver

It is free SQL database tool for developers and database administrators. It can work with any database server which has JDBC driver.

Database Maintenance

Database Maintenance is an activity performed to keep a database running smoothly and efficiently. The maintenance of databases is generally performed by people who are familiar with the database system like DBA using the given procedures and tools. The activities in database maintenance include

- Backup of the database at regular intervals
- Restore the database at need
- Freeing the disk space and other resources
- Rebuilding indexes, removing duplicate records, etc.

Backup and Recovery

Backup

Backup is a process of copying the database onto another storage medium. The backup is normally performed at regular intervals. The backup is used to reconstruct the database when the original database becomes corrupt in the following conditions

- System crash
- Network failure
- Disk failure
- Natural and physical disaster
- Exception condition

Types of backup

There are two types of backup. They are

(1) Physical backup

In this type of backup, the actual physical database is copied onto another storage device like CD-ROM, magnetic tape, zip drive, etc.

(2) Logical backup
In this type of backup, the data are extracted from the database using SQL statements and stored as binary file.

Recovery

Database recovery is the process of restoring the database to its most recent consistent state in case of any failure. The recovery procedure is given below

- If there is a damage to the database due to hardware problem like disk crash, the database is recovered by copying the files from the backup.
- If the database fails due to transaction error, the database is recovered by rollbacking the transactions that caused inconsistency.

Summary

- ➢ Database Management System is defined as a collection of interrelated data and a set of programs to create, access and maintain those data in an efficient manner.
- ➢ The logical design of the database is called Database Model.
- ➢ The three types of database models are Hierarchical database model, Network database model and Relational database model.
- ➢ The four types of database are Transactional database, Decision support database, Hybrid database and Open source database.
- ➢ Codd's rules are the rules that every DBMS must obey to be Relational DBMS.
- ➢ Key is defined as an attribute or group of attributes used to uniquely identify a row.
- ➢ Data constraints are policies to maintain accuracy and integrity of data in the database during database operations.
- ➢ E-R model describes the structure of the database and the operations on the database in a pictorial representation.
- ➢ Normalization is a database design technique to reduce redundancy and inconsistency of data.
- ➢ Backup is a process of copying the database onto another storage medium.
- ➢ Database recovery is the process of restoring the database to its most recent consistent state in case of any failure.

Review Questions

Unit 1.1

PART A

1. 1.Define DBMS.
2. 2.What is open source database?
3. 3.What is Codd’s rule?
4. 4.Define record. (or) Define row. (or) Define tuple.
5. 5.Define attribute.
6. 6.What is key?
7. What is primary key?
8. What is composite key?
9. Define meta data.
10. What is data dictionary?
11. Define data integrity.
12. Define entity.
13. What is Normalization?
14. .Define DBA.
15. What is backup?
16. What is recovery?

PART B

1. List the components of database.
2. Explain the functions of database.
3. What are the different types of database model?
4. Mention the advantages and disadvantages of relational data model
5. Discuss transactional database.
6. Discuss decision support database.
7. Discuss hybrid database
8. What are the different types of constraints?
9. List the different types of Normalization
10. List the difference between SQL and MySQL.
11. Explain first normalization
12. Explain open source database

PART C

1. Discuss characteristics of database.
2. Discuss components of database.
3. Explain briefly the functions of database.
4. Discuss evolution of database model.

5. Explain various types of database models.
6. Explain (a) Hierarchical database model (b) Network database model
7. (c) Relational database model
8. What is primary key? How do you create primary key?
9. What is foreign key? How do you create foreign key?
10. List the differences between SQL and MySQL.
11. Explain Codd's rules.
12. Explain various types of data constraints.
13. Briefly discuss E-R diagram.
14. Discuss various types of relationships.
15. Draw some sample E-R diagrams.
16. Explain 1NF, 2NF and 3NF Normalization.
17. Discuss DBA tasks.
18. Discuss DBA tools and utilities.
19. Discuss backup and recovery.
20. Explain Client Server system and Distributed system.

****

UNIT II – MySQL ADMINISTRATION & DATABASE DESIGN

Learning Objectives

At the end of the unit, the students will be able to

- Understand the features of MySQL.
- Install, connect and access MySQL
- Create and view databases.
- Manipulate databases with MySQL.
- Understand different types of operators and expressions in MySQL.
- Import and export data.
- Use various built-in functions.
- Make different types of queries and sub-queries to retrieve data
- Understand and use flow control commands.

Installation of MYSQL

Introduction to

MySQL

MySQL is the most popular open source relational database management system (RDBMS) used for developing web-based software applications based on Structured Query Language (SQL).

- MySQL is originally developed, distributed, and supported by the Swedish company (MySQL AB), and acquired by Oracle in 2008.
- Applications which use MySQL databases include: TYPO3, MODx, Joomla, WordPress, phpBB, MyBB, Drupal.
- MySQL is also used in many high-profile, large-scale websites, including Google, Facebook, Twitter, Flickr, and YouTube.

Features of MySQL

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is becoming so popular because of many good reasons:

- MySQL is released under an open-source license. So it is free.
- MySQL is a very powerful. It handles a large subset of the functionality of the most expensive and powerful database packages.

- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

Downloading and Installing MySQL

- Go to the download page for MySQL Installer for Windows. This page is available at the following URL: https://dev.mysql.com/downloads/installer/
- Download the required version of the Installer, either 32 bit or 64 bit for your system.
- Follow the instructions provided on that web page to download the installer file to your hard disk.
- Find/Open the installer file on your disk and run it.
- Respond to the resulting dialog boxes. You can accept most of the default options. Specify the password for the root. Remember the password that you enter.
- To make sure that the database has been installed correctly, start MySQL Server from your system.

Starting and Stopping Connections to MySQL Server

- MySQL provides an interactive shell for creating tables, inserting data, etc.
- On Windows, just go to "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysql.exe"
- Or, go to start menu→ all Programs → MySQL →MySQL Server 5.5 →MySQL 5.5 Command Line Client. (as shown in Fig. 2.1)
- MySQL will start and prompt for password. (Fig. 2.2)

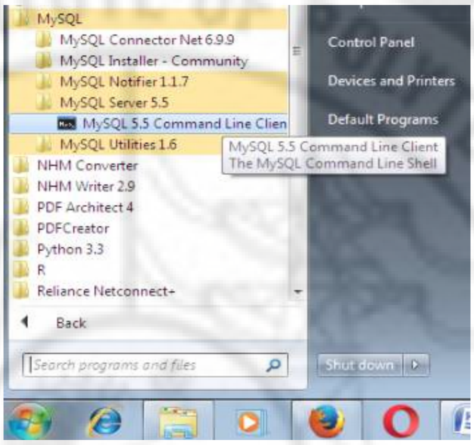
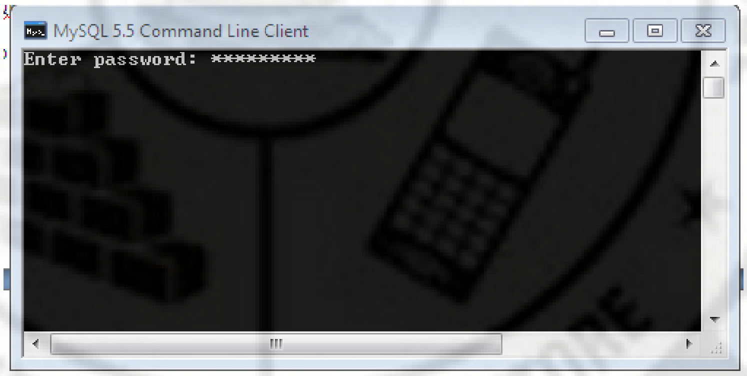


Fig. 2.1 Starting MySQL

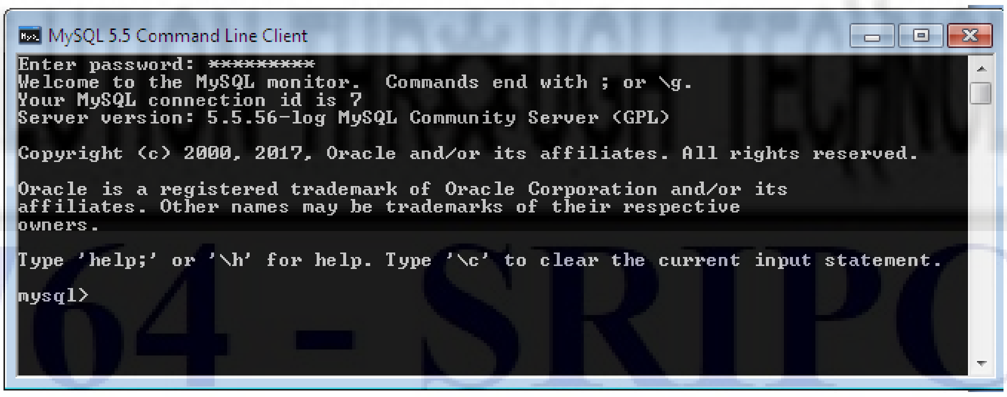


```
Enter password: **********
```

Fig. 2.2 Connecting to MySQL

Enter password: *****

The welcome screen appears as shown in Fig.2.3



```
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.5.56-log MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Fig. 2.3 Command line interface – Welcome screen

After connecting to the MySQL server, the mysql> prompt appears. Now user can enter the sql commands. To see the command line help type '\h' on the mysql prompt.

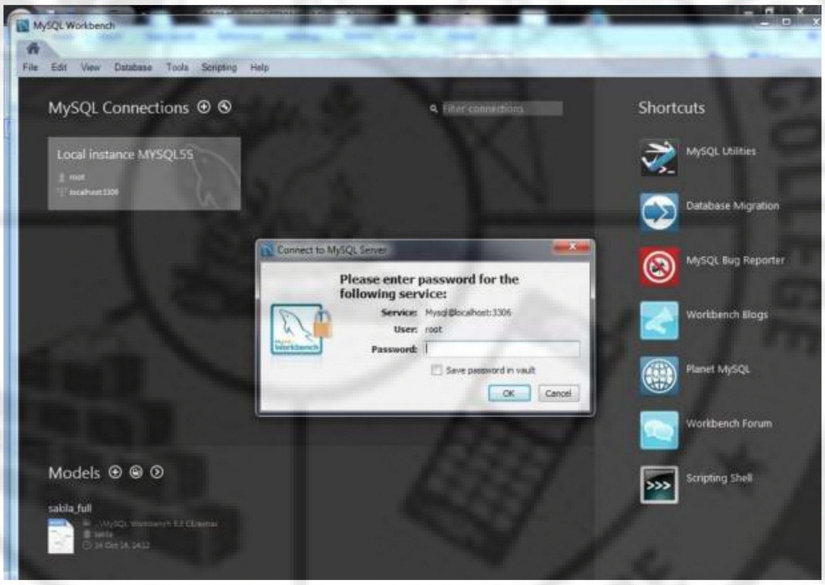
mysql>

To exit MySQL just type QUIT or EXIT:

```
mysql> QUIT

mysql> exit
```

Starting and Stopping connections to the MySQL server using Workbench



Please enter password for the following service:

Service: Mysql@localhost:3306

User: root

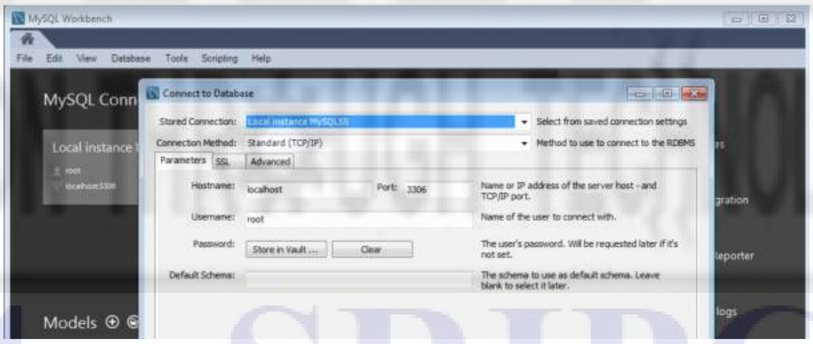
Password:

Save password in vault

Fig: 2.4 Connecting to MySql Server

By specifying the root password, MySQL server can be connected.

From the Database menu select Connect to Database



Stored Connection: Local instance MySQL55 — Select from saved connection settings

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Hostname: localhost Port: 3306 — Name or IP address of the server host - and TCP/IP port.

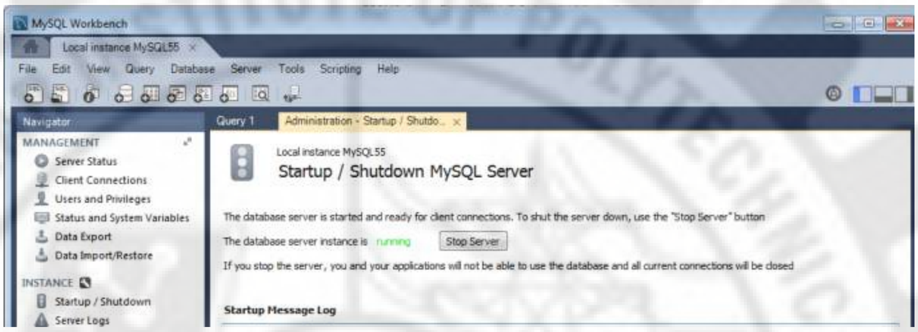
Username: root — Name of the user to connect with.

Password: — The user's password. Will be requested later if it's not set.

Default Schema: — The schema to use as default schema. Leave blank to select it later.

Fig. 2.5 Connect to Database

MySQL Workbench can Startup/Shutdown the server from the Workbench navigator.



Local instance MySQL55

Startup / Shutdown MySQL Server

The database server is started and ready for client connections. To shut the server down, use the "Stop Server" button

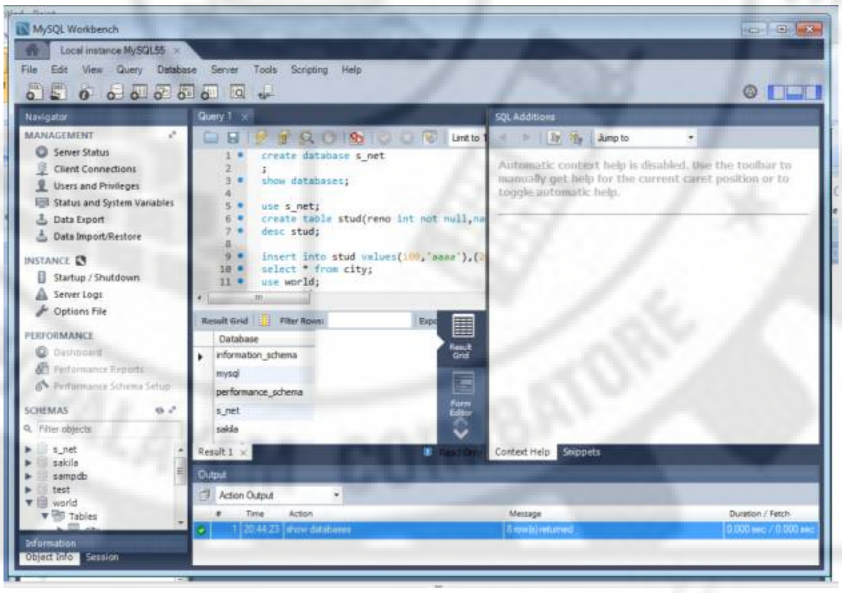
The database server instance is running

If you stop the server, you and your applications will not be able to use the database and all current connections will be closed

Startup Message Log

Fig. 2.6 Startup / Shutdown the server

After connecting to a database the user can use the menus and navigator to manage the databases and tables.



```
create database s_net
;
show databases;

use s_net;
create table stud(reno int not null,na
desc stud;

insert into stud values(100,'aaaa'),(2
select * from city;
use world;
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Database
information_schema
mysql
performance_schema
s_net
sakila

#	Time	Action	Message	Duration / Fetch
1	20:44:23	show databases	8 row(s) returned	0.000 sec / 0.000 sec

Fig. 2.7 Navigator window

Accessing MySQL

MySQL can be accessed using different interfaces.

MySQL Command Line Client

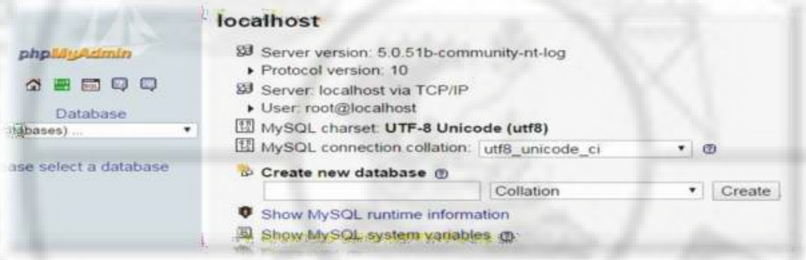
MySQL Command Line Client is a single exe that allows connecting and running a sample query. It is a simple SQL shell for creating tables, inserting data, etc

Free graphical administration applications (or "front ends") are available that integrate with MySQL and enable users to work with database structure and data visually.

Web interface (PHP MyAdmin)

PHP MyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL over the Web. PHP MyAdmin supports a wide range of operations on MySQL and MariaDB. Frequently used operations (managing databases, tables, columns, relations, indexes, users, permissions, etc) can be performed via the user interface, while you still have the ability to directly execute any SQL statement. PHP MyAdmin comes with a wide range of documentation.

It's a PHP driven tool and handles every aspect of creating and managing a **MySQL database application**. PHP MyAdmin also allows for the execution of MySQL commands from within the interface.



localhost

Server version: 5.0.51b-community-nt-log
Protocol version: 10
Server: localhost via TCP/IP
User: root@localhost
MySQL charset: **UTF-8 Unicode (utf8)**
MySQL connection collation: utf8_unicode_ci

2.1.5.Desktop tools (MySQL Workbench)

It is a unified visual tool for database architects and developers. MySQL Workbench provides data modeling, SQL development, and administration tools for server configuration, user administration, backup etc. MySQL Workbench is available on Windows, Linux and Mac OS X.

MySQL Workbench is a free graphical tool that makes it easier to work with MySQL databases. MySQL Workbench is an ideal tool for learning how to work with MySQL. It is bundled with the Server in the installation package. If you use the MySQL Installer to install the Community Server, you also get MySQL Workbench. However MySQL Workbench can be installed separately. MySQL Workbench provides five main areas of functionality:

• SQL Development:

Enables you to create and manage connections to database servers. As well as enabling you to configure connection parameters, MySQL Workbench provides the capability to execute SQL queries on the database connections using the built-in SQL Editor.

• Data Modeling:

Enables you to create models of your database schema graphically, and edit all aspects of your database using the comprehensive Table Editor. The Table Editor provides easy-to-use facilities for editing Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts and Privileges, Routines and Views.

• Server Administration:

Enables you to create and administer server instances.

• Data Migration:

Allows you to migrate from Microsoft SQL Server, Sybase ASE, SQLite, and other RDBMS tables, objects and data to MySQL. Migration also supports migrating from earlier versions of MySQL to the latest releases.

• MySQL Enterprise Support:

Support for Enterprise products such as MySQL Enterprise Backup and MySQL Audit.

MySQL Workbench for Windows Prerequisites:
To be able to install and run MySQL Workbench on Windows your system needs to have libraries listed below installed. The listed items are can be downloaded from Microsoft download page.

Microsoft .NET Framework 4.5
Visual C++ Redistributable for Visual Studio 2015

Working with MySQL Databases

Introduction

At this point, it is more important to know about how to issue or type commands in the mysql prompt before creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering queries.

- A query normally consists of an SQL statement followed by a semicolon.

MySql displays query output in tabular form (rows and columns). The first row contains labels for the columns. Other rows are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column, MySql labels the column using the expression itself.

- Keywords may be entered in any letter case. A query need not be given all on a single line, so lengthy queries that require several lines are not a problem. MySql determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line.

Basic Queries
Once logged in, you can try some simple queries.

For example:

```
mysql> SELECT VERSION( ), CURRENT_DATE;
-------------------------------------------------------------
| VERSION()        |        CURRENT_DATE |
-------------------------------------------------------------
| 3.23.49          |        2002-05-26          |
-------------------------------------------------------------
1 row in set (0.00 sec)
```

Note that most MySQL commands end with a semicolon (;). MySQL returns the total number of rows found, and the total time to execute the query.

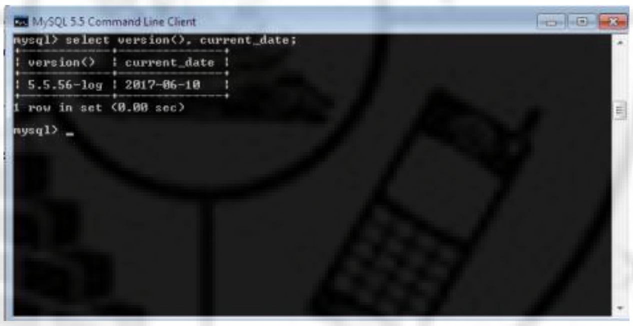
Note: MySQL Command Line Client is a single executable that allows connecting and running a sample query. It is a simple SQL shell (with GNU read line capabilities).

Keywords may be entered in any letter case. The following queries are equivalent:

mysql> SELECT VERSION(), CURRENT_DATE;

mysql> select version(), current_date;

mysql> SeLeCt vErSiOn(), current_DATE;



```
mysql> select version(), current_date;
+------------+--------------+
| version()  | current_date |
+------------+--------------+
| 5.5.56-log | 2017-06-10   |
+------------+--------------+
1 row in set (0.00 sec)

mysql> _
```

Multi-Line Commands

MysSQL determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. Here's a simple multiple-line statement:

```
mysql> SELECT  USER( ) CURRENT_DATE;
-------------------------------------------------------------
| USER( )                              | CURRENT_DATE |
-------------------------------------------------------------
| Karwan@localhost   | 1999-03-18              |
-------------------------------------------------------------
1 row in set (0.00 sec)
```

Canceling a Command

If you decide you don't want to execute a command that you are in the process of entering, cancel it by typing \c

```
mysql> SELECT
-> USER()
-> \c
mysql>
```

Using and selecting Database

To the select a database, issue the “use” command:

mysql> use test;

Creating a Database

We will begin by creating sample database and the tables within it, populating its tables, and performing some simple queries on the data contained in those tables. Using a database involves several steps:

1. Creating (initializing) the database
2. Creating the tables within the database
3. Manipulating the tables by inserting, retrieving, modifying, or deleting data.

CREATE command is used to create objects like database, table, view, index, function etc.

Syntax:

CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name [create_specification] ...

create specification:

```
  [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

CREATE DATABASE creates a database with the given name. To use this statement, you need the CREATE privilege for the database. CREATE SCHEMA is a synonym for CREATE DATABASE.

mysql> **CREATE DATABASE sampledb;**

You will need to create the sampledb database before you can create any of the tables that will go in it or do anything with the contents of those tables.

CREATE TABLE statement is used to specify the layout of the table. This statement has the following general form:

CREATE TABLE tbl_name (column_specs);

tbl_name indicates the name of the table.
column_specs provides the specifications for the columns in the table.
A CREATE TABLE statement specifies, at a minimum, the table name and a list of columns in it. For example:

```
CREATE TABLE mytbl
(
name CHAR(20),
birth DATE NOT NULL,
weight INT,
sex ENUM('F','M')
);
```

Selecting database

Creating a database does not select it for use; you must do that explicitly. To make database the current database, invoke USE command with database name. Your database needs to be created only once, but it must be selected for use each time a mysql session begins. You can do this by issuing a USE statement.

```
mysql> USE sampledb;
mysql> SELECT DATABASE();
- - - - - - - - - - - - -
| DATABASE( )       |
- - - - - - - - - - - - -
| sampledb          |
- - - - - - - - - - - - -
```

The USE db_name statement tells MySQL to use the db_name database as the default (current) database for subsequent statements. The database remains the default until the end of the session or another USE statement is issued.

```
USE db1

SELECT COUNT (*) FROM mytable # selects from db1.mytable

USE db2

SELECT COUNT (*) FROM mytable  # selects from db2.mytable
```

USE like QUIT does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The USE statement is special in another way, too: it must be given on a single line.

Describing database

Getting Information about Databases and Tables

It is necessary to get the information about the databases and tables. Sometimes the structure (column details) of a given table is required. We can get this information in MySQL. The statements SHOW and DESCRIBE provide information about the databases and tables it supports.

```
mysql> describe city;

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| Field        | Type        | Null | Key   | Default |      Extra          |
-------------------------------------------
| ID           | int(11)     | NO   | PRI   | NULL| auto_increment |
| Name         | char(35)    | NO   |       |       |              |
| CountryCode | char(3)     | NO   | MUL |       |              |
| District     | char(20)    | NO   |       |       |              |
| Population  | int(11)     | NO   |       | 0            |              |
-----------------------------------------------------------------
5 rows in set (0.02 sec)
```

DESC is a short form of DESCRIBE.

The DESCRIBE command is a shortcut for the SHOW COLUMNS command; DESCRIBE table is identical to SHOW COLUMNS FROM table.

SHOW command

The SHOW DATABASES command lists the databases managed by the server.

```
mysql> show databases;
- - - - - - - - - - - - - - - - - - -
| Database            |
- - - - - - - - - - - - - - - - - - -
| information_schema        |
| mysql               |
| performance_schema|
| s_net               |
| sakila              |
| sampdb              |
| test                |
| world               |
- - - - - - - - - - - - - - - - - - -
8 rows in set (0.13 sec)
```

To find out which database is currently selected, use the DATABASE() function

```
mysql> select database();
- - - - - - - - - - -
| database( ) |
- - - - - - - - - - -
| world        |
- - - - - - - - - - -
1 row in set (0.00 sec)
```

SHOW has many forms that provide information about databases, tables, columns, or status information about the server. SHOW is helpful for keeping track of the contents of your databases and to know about the structure of your tables.

To display the CREATE DATABASE statement for a database:

```
SHOW CREATE DATABASE db_name;
```

To list the tables in the default database or in a given database:

```
SHOW TABLES;
SHOW TABLES FROM db_name;
```

SHOW TABLES doesn't show TEMPORARY tables.

Backing up Databases

One important task of database administrator is to backup the database regularly. Then, if the database is corrupted, the administrator can use the backup files to restore the database.

- It’s important for the database administrator to regularly back up the database. Then, if the database becomes corrupted, the database administrator can use the backup to restore the database.
- A *full backup* includes the structure and content of a database.
- An *incremental backup* contains only changes that have been made to the structure and content of a database since the last full backup.
- You often want to include the database named Mysql in your backups, since this database stores information about the users and privileges for all database on the server.
- You shouldn’t store your backup files (SQL Scripts or log files) on the same hard drive where the MYSQL server is running. If you do, those backup files will be lost along with the database if that hard drive fails.
- A *point-in-time recovery* (PITR) allows you to restore the data up to any specified point in time.

The **mysqldump** program generates backup files that are more useful for database restoration, and it enables you to create backups without taking down the server. You might also need the backup files for moving databases to a different location.

MySQLdump program can be executed from the command prompt to perform a backup. To backup a single database, specify the name of the database. To backup multiple databases, use the –databases or -all-databases option. After specifying the database name, enter the > character followed by the path to the SQL script file where you want to store the backup.

The –u option is coded followed by the name of the user with backup privileges. The final code –p prompts for a password.

Making Backups with mysqldump

The **mysqldump** program can make backups. It can backup all kinds of tables.

Dump each database using mysqldump

```
>mysqldump --databases db_name > db_name.sql
```

For a single database

```
>mysqldump ap > /murach/mysql/ap-2012-02-23.sql –u root -p
```

For specified databases

```
>mysqldump – databases ap ex om mysql > /murach/mysql/backup-2012-02-
23.sql -u root –p
```

For all databases

```
>mysqldump – all –databases >  /murach /mysql/all-ab-2012-02-23.sql –u  root  -p
```

Introduction to MySQL

MySQL data types

MySQL supports a number of SQL data types in several categories: **character types, numeric types, date and time types, large objects and spatial types.** Each table in a database contains one or more columns. When you create a table using a CREATE TABLE statement, you specify a data type for each column. A data type is more specific than a general category such as “number” or “string.”

MySQL uses different data types which are classified into five major categories.

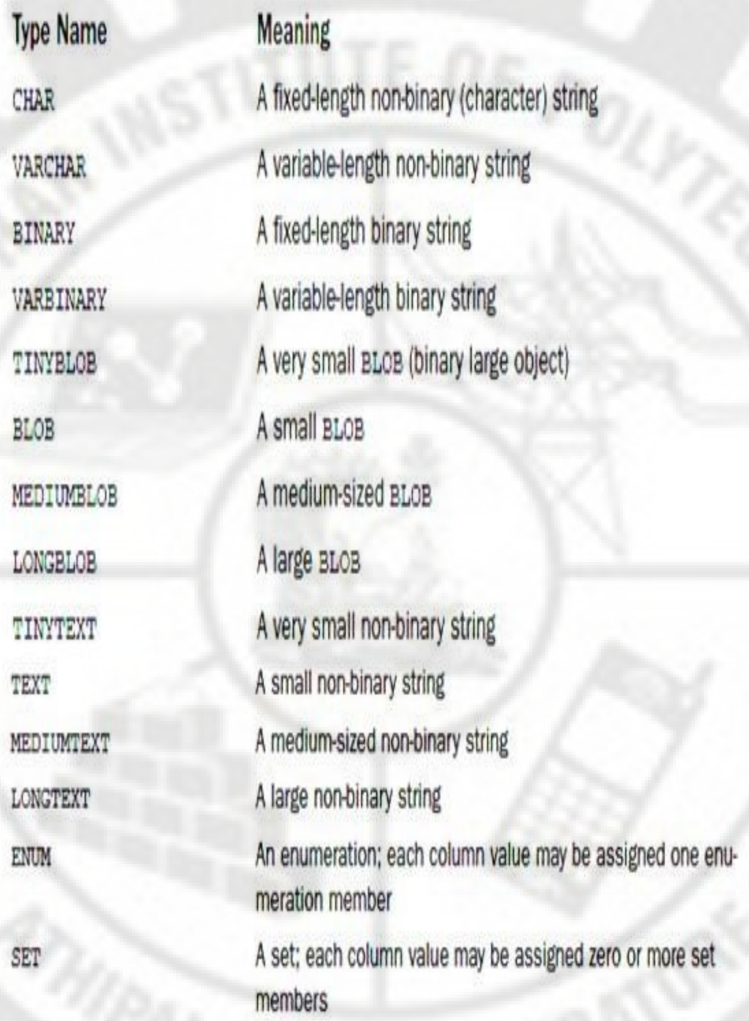
Category	Description
Character	Strings of character data
Numeric	Numbers that don’t include a decimal point (integers) and numbers that include a decimal point (real numbers)
Date and time	Dates, times, or both
Large Object (LOB)	Large strings of character or binary data
Spatial	Geometric or geographical values

Numeric data types

Type Specification	Storage Required	
TINYINT[(*M*)]	1 byte	**A very small integer**
SMALLINT[(*M*)]	2 bytes	**A small integer**
MEDIUMINT[(*M*)]	3 bytes	**A medium-sized integer**
INT[(*M*)]	4 bytes	**A standard integer**
BIGINT[(*M*)]	8 bytes	**A large integer**
DECIMAL([*M*[,*D*]])	Varies depending on *M*, *D*	**A fixed point number**
FLOAT[(*M*,*D*)]	4 bytes	**A single-precision floating point number**
DOUBLE[(*M*,*D*)]	8 bytes	**A double-precision floating point number**
BIT[(*M*)]	Varies depending on *M*	**A bit field**

MySQL uses all the standard ANSI

String data types



Type Name	Meaning
CHAR	A fixed-length non-binary (character) string
VARCHAR	A variable-length non-binary string
BINARY	A fixed-length binary string
VARBINARY	A variable-length binary string
TINYBLOB	A very small BLOB (binary large object)
BLOB	A small BLOB
MEDIUMBLOB	A medium-sized BLOB
LONGBLOB	A large BLOB
TINYTEXT	A very small non-binary string
TEXT	A small non-binary string
MEDIUMTEXT	A medium-sized non-binary string
LONGTEXT	A large non-binary string
ENUM	An enumeration; each column value may be assigned one enumeration member
SET	A set; each column value may be assigned zero or more set members

String Types

Although numeric and date types are fun, most data you'll store will be in string format. This list describes the common string data types in MySQL.

- **CHAR(M)** - A fixed-length string between 1 and 255 characters in length, right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.

- **VARCHAR(M)** - A variable-length string between 1 and 255 characters in length; for example VARCHAR(25). You must define a length when creating a VARCHAR field.

- **BLOB or TEXT** - A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data; the difference between the two is that sorts and comparisons on stored data are case sensitive on BLOBs and are not case sensitive in TEXT fields. You do not specify a length with BLOB or TEXT.

- **TINYBLOB or TINYTEXT** - A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.

- **MEDIUMBLOB or MEDIUMTEXT** - A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.

- **LONGBLOB or LONGTEXT** - A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.

- **ENUM** - When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) can be used for that field.

Date and Time data types

Type Name	Meaning
DATE	A date value, in *'CCYY-MM-DD'* format
TIME	A time value, in *'hh:mm:ss'* format
DATETIME	A date and time value, in *'CCYY-MM-DD hh:mm:ss'* format
TIMESTAMP	A timestamp value, in *'CCYY-MM-DD hh:mm:ss'* format
YEAR	A year value, in *CCYY* or *YY* format

Spatial Data Types
The MySQL spatial data types represent various kinds of geometrical or geographical values.

Type Name	Meaning
GEOMETRY	A spatial value of any type
POINT	A point (a pair of X,Y coordinates)
LINESTRING	A curve (one or more POINT values)
POLYGON	A polygon
GEOMETRYCOLLECTION	A collection of GEOMETRY values
MULTILINESTRING	A collection of LINESTRING values
MULTIPOINT	A collection of POINT values
MULTIPOLYGON	A collection of POLYGON values

Data Definition Commands

The statements that create and work with the objects within a database are called the data definition language (DDL). These statements are used exclusively by database administrators (DBAs). It is the DBA's job to maintain existing databases, tune them for faster performance and create new databases. The common tasks in data definition language are **create**, **alter** and **delete** or **drop** databases, tables and indexes.

<u>**CREATE TABLE**</u>

Create table statement creates a table with the given name. You must have the CREATE privilege for the table. By default, tables are created in the default database, using the InnoDB storage engine. An error occurs if the table exists, if there is no default database, or if the database does not exist. The original CREATE TABLE statement, including all specifications and table options are stored by MySQL when the table is created

A CREATE TABLE statement specifies, at a minimum, the table name and a list of the columns in it. To create a table only if it doesn't already exist, use CREATE TABLE IF NOT EXISTS. You can use this statement for an application that makes no assumptions about whether a table that has been set up in advance

Complete syntax of CREATE command

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
( col_name data_type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'string']
[COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
[STORAGE {DISK|MEMORY|DEFAULT}]

| [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)

| [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY] (index_col_name,...)
| [CONSTRAINT [symbol]] FOREIGN KEY (index_col_name,...) reference_definition
| CHECK (expr)
);
```

The original CREATE TABLE statement, including all specifications and table options are stored by MySQL when the table is created.

CREATE TABLE ... LIKE Syntax - Cloning or copying tables

Use CREATE TABLE ... LIKE to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

CREATE TABLE *new_tbl* LIKE *orig_tbl*;

The copy is created using the same version of the table storage format as the original table. The SELECT privilege is required on the original table. LIKE works only for base tables, not for views.

The table name can be specified as *db_name.tbl_name* to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists.

Some of the optional parameters

IF NOT EXISTS

Prevents an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the CREATE TABLE statement.

Temporary Tables

You can use the TEMPORARY keyword when creating a table. A TEMPORARY table is visible only to the current session, and is dropped automatically when the session is closed.

[AS] query_expression

To create one table from another, add a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl AS SELECT * FROM orig_tbl;
```

IGNORE|REPLACE

The IGNORE and REPLACE options indicate how to handle rows that duplicate unique key values when copying a table using a SELECT statement.

Column Data Types and Attributes

There is a hard limit of 4096 columns per table.

data_type represents the data type in a column definition. Some attributes do not apply to all data types. AUTO_INCREMENT applies only to integer and floating-point types. DEFAULT does not apply to the BLOB, TEXT, GEOMETRY, and JSON types.

NOT NULL | NULL

If neither NULL nor NOT NULL is specified, the column is treated as though NULL had been specified.

DEFAULT

Specifies a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression

AUTO_INCREMENT

An integer or floating-point column can have the attribute AUTO_INCREMENT. When you insert a value of NULL (recommended) or 0 into an indexed AUTO_INCREMENT column, the column is set to the next sequence value. Typically this is value+1, where value is the largest value for the column currently in the table. AUTO_INCREMENT sequences begin with 1.

COMMENT

A comment for a column can be specified with the COMMENT option, up to 1024 characters long. The comment is displayed by the SHOW CREATE TABLE and SHOW FULL COLUMNS statements.

Indexes and Foreign Keys

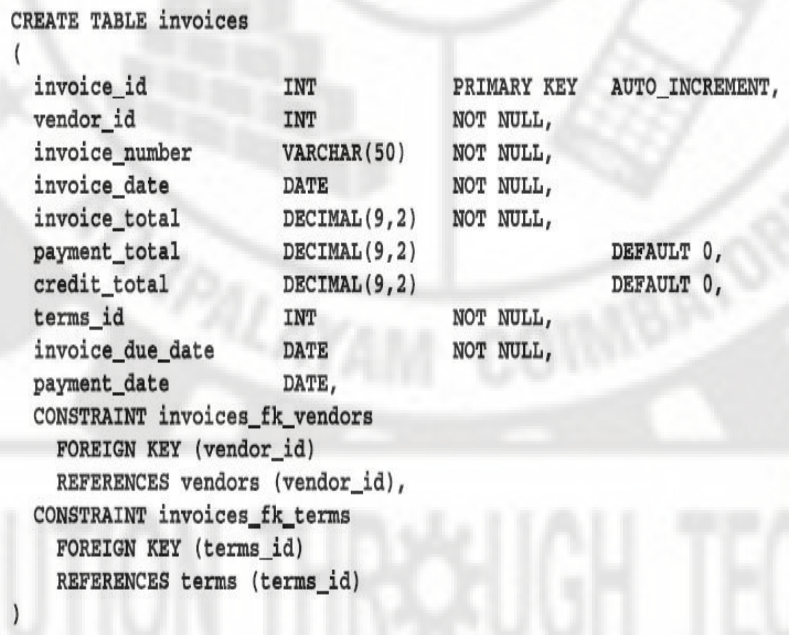
CONSTRAINT symbol

If the CONSTRAINT symbol clause is given, the symbol value, if used, must be unique in the database. A duplicate symbol results in an error. If the clause is not given, or a symbol is not included following the CONSTRAINT keyword, a name for the constraint is created automatically.

PRIMARY KEY

A unique index where all key columns must be defined as NOT NULL. If they are not explicitly declared as NOT NULL, MySQL declares them so implicitly (and silently). A table can have only one PRIMARY KEY. The name of a PRIMARY KEY is always PRIMARY, which thus cannot be used as the name for any other kind of index.

Examples:
A statement that creates a new table invoices



```
CREATE TABLE invoices
(
  invoice_id          INT            PRIMARY KEY   AUTO_INCREMENT,
  vendor_id           INT            NOT NULL,
  invoice_number      VARCHAR(50)    NOT NULL,
  invoice_date        DATE           NOT NULL,
  invoice_total       DECIMAL(9,2)   NOT NULL,
  payment_total       DECIMAL(9,2)                 DEFAULT 0,
  credit_total        DECIMAL(9,2)                 DEFAULT 0,
  terms_id            INT            NOT NULL,
  invoice_due_date    DATE           NOT NULL,
  payment_date        DATE,
  CONSTRAINT invoices_fk_vendors
    FOREIGN KEY (vendor_id)
    REFERENCES vendors (vendor_id),
  CONSTRAINT invoices_fk_terms
    FOREIGN KEY (terms_id)
    REFERENCES terms (terms_id)
)
```

<u>**ALTER TABLE**</u>

ALTER TABLE changes the structure of a table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change characteristics such as the storage engine used for the table or the table comment.

• To use ALTER TABLE, you need ALTER, CREATE, and INSERT privileges for the table. Renaming a table requires ALTER and DROP on the old table, ALTER, CREATE, and INSERT on the new table.

• Following the table name, specify the alterations to be made. If none are given, ALTER TABLE does nothing.

The syntax for ALTER TABLE looks like this:

```
ALTER TABLE tbl_name
[alter_specification [, alter_specification] ...]
[partition_options]
| ADD [COLUMN] col_name column_definition
[FIRST | AFTER col_name ]
| MODIFY [COLUMN] col_name column_definition
[FIRST | AFTER col_name]
| DROP [COLUMN] col_name
| DROP PRIMARY KEY
| DROP {INDEX|KEY} index_name
| DROP FOREIGN KEY fk_symbol
```

You can issue multiple ADD, ALTER, DROP, and CHANGE clauses in a single ALTER TABLE statement, separated by commas. This is a MySQL extension to standard SQL, which permits only one of each clause per ALTER TABLE statement. For example, to drop multiple columns in a single statement do this:

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

To add a new AUTO_INCREMENT integer column named c:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
ADD PRIMARY KEY (c);
```

A statement that adds a new column to a table

```
  ALTER TABLE invoices ADD balance_due DECIMAL (9, 2);
```

<u>RENAME TABLE</u>

This statement renames one or more tables. The rename operation is done atomically, which means that no other session can access any of the tables while the rename is running.

Syntax

```
RENAME TABLE tbl_name TO new_tbl_name [, tbl_name2 TO new_tbl_name2] ...
```

You can also use a RENAME clause that specifies the new table name in the ALTER TABLE command.

```
ALTER TABLE tbl_name  RENAME TO new_tbl_name;
```

A table named old_table can be renamed to new_table as shown here:

```
RENAME TABLE old_table TO new_table;
```

This statement is equivalent to the following ALTER TABLE statement:

ALTER TABLE old_table RENAME new_table;

If the statement renames more than one table, renaming operations are done from left to right. If you want to swap two table names, you can do so like this (assuming that tmp_table does not already exist):

RENAME TABLE old_table TO tmp_table,
new_table TO old_table,
tmp_table TO new_table;

As long as two databases are on the same file system, you can use RENAME TABLE to move a table from one database to another.

One thing that RENAME TABLE can do that ALTER TABLE cannot is rename multiple tables in the same statement. For example, you can swap the names of two tables like this:

RENAME TABLE t1 TO tmp, t2 TO t1, tmp TO t2;

If you qualify a table name with a database name, you can move a table from one database to another by renaming it. Either of the following statements moves the table t from the **sampledb** database to the **test** database:

ALTER TABLE sampledb.t RENAME TO test.t;

RENAME TABLE sampledb.t TO test.t;

<u>COPYING table</u>

Copying a table is a facility to create a new table from existing table.

You can create one table from another by adding a SELECT statement at the end of the CREATE TABLE statement:

CREATE TABLE *new_tbl* [AS] SELECT * FROM *orig_tbl*;

MySQL creates new table with all columns (or specified columns) in the SELECT clause.

<u>DELETING tables</u>

MySQL tables can be deleted using DROP, DELETE and TRUNCATE commands.

DROP command

DROP TABLE removes one or more tables. You must have the DROP privilege for each table. All table data and the table definitions are *removed*, so *be careful* with this statement! If any of the tables named in the argument list do not exist, MySQL returns an error indicating by name of the non-existing tables, but it drops all of the tables in the list that exist.

DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]
```

DROP TABLE automatically commits the current active transaction, unless you use the TEMPORARY keyword.

The TEMPORARY keyword has the following effects:
• The statement drops only TEMPORARY tables.
• The statement does not end an ongoing transaction.
• No access rights are checked.

```
DROP TABLE tbl_name1, tbl_name2, ;
```

DELETE command

Deleting Existing Rows

Sometimes you want to get rid of rows or change their contents. The DELETE and UPDATE statements let you do this. This section discusses how to use them. The DELETE statement has this form:

```
DELETE FROM tbl_name WHERE which rows to delete;
```

The WHERE clause that specifies which rows should be deleted is optional, but if you leave it out; all rows in the table are deleted. In other words, the simplest DELETE statement is also the most dangerous:

```
DELETE FROM tbl_name;
```

That statement wipes out the table's contents entirely. To delete specific rows, use the WHERE clause to identify the rows you want to delete. This is similar to using a WHERE clause in a SELECT statement to avoid selecting the entire table.

```
mysql> DELETE FROM president WHERE state='Ohio';
Query OK, 7 rows affected (0.00 sec)
```

TRUNCATE TABLE command

TRUNCATE TABLE empties a table completely. TRUNCATE TABLE deletes all the data in the table without deleting column definitions. It requires the DROP privilege. Logically, TRUNCATE TABLE is similar to a DELETE statement that deletes all rows, or a sequence of DROP TABLE and CREATE TABLE statements.

To achieve high performance, it bypasses the DML method of deleting data. Thus, it cannot be rolled back; it does not cause ON DELETE triggers to fire. Although TRUNCATE TABLE is similar to DELETE, it is classified as a DDL statement rather than a DML statement.

Syntax

```
TRUNCATE TABLE table_name;
```

Temporary tables

We can use the **TEMPORARY** keyword when creating a table. A **TEMPORARY** table is visible only to the current session, and is dropped automatically when the session is closed.
To create a table as a temporary copy of itself, include the TEMPORARY keyword:

```
CREATE TEMPORARY TABLE new_tbl_name LIKE tbl_name;
```

Using a TEMPORARY table with the same name as the original can be useful when you want to try some statements that modify the contents of the table, but you don't want to change the original table.

CREATE TABLE does not automatically commit the current active transaction if you use the **TEMPORARY** keyword.

TEMPORARY tables have a very loose relationship with databases (schemas). Dropping a database does not automatically drop any **TEMPORARY** tables created within that database.

To create a temporary table, you must have the **CREATE TEMPORARY TABLES** privilege. After a session has created a temporary table, the server performs no further privilege checks on the table. The creating session can perform any operation on the table, such as **DROP TABLE, INSERT, UPDATE, or SELECT.**

Data Manipulation Commands

The statements that change or modify the data in a database are called the data manipulation language (DML) commands. These are the most frequently used statements by the application programmers. The INSERT, UPDATE and DELETE are data manipulation commands.

INSERT command

Insert command is used to add one or more rows of values in to the table. The different forms of INSERT are given below.

```
INSERT INTO tbl_name VALUES(value1,value2,...);

INSERT INTO tbl_name (col_name1,col_name2,...) VALUES(value1,value2,...);

INSERT INTO tbl_name VALUES(...),(...),... ;
```

INSERT inserts new rows into an existing table. The INSERT ... VALUES and INSERT ... SET forms of the statement insert rows based on explicitly specified values. The INSERT ... SELECT
form inserts rows selected from another table or tables.

```
INSERT INTO tbl_name ( ) VALUES ( ) ;
```

An expression *expr* can refer to any column that was set earlier in a value list.

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

But the following is not valid.

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

Here col2 is unknown when inserting col1. So, it is invalid.

Inserting multiple rows

INSERT statements that use VALUES syntax can insert multiple rows. To do this, include multiple lists of column values, each enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

The values list for each row must be enclosed within parentheses.

With INSERT ... SELECT, you can quickly insert many rows into a table from one or many tables.

For example:

```
INSERT INTO tbl_temp2 (fld_id) SELECT tbl_temp1.fld_order_id
FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

When selecting from and inserting into a table at the same time, MySQL creates a temporary table to hold the rows from the SELECT and then inserts those rows into the target table.

This following syntax uses a SET clause containing col_name=value assignments rather than a VALUES( ) list:

```
INSERT INTO tbl_name SET col_name1=value1, col_name2=value2, ... ;
```

Example:

```
mysql> INSERT INTO member SET last_name='Stein',first_name='Waldo';
```

UPDATE command

UPDATE is a DML statement that modifies rows in a table. One or more tables can be updated at a time.

Single-table syntax

UPDATE [LOW_PRIORITY] [IGNORE] *table_reference*
SET *col_name1*={*expr1*|DEFAULT} [, *col_name2*={*expr2*|DEFAULT}] ...
[WHERE *where condition*]

```
[ORDER BY ...]
[LIMIT row_count]
```

Multiple-table syntax

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
[WHERE where_condition]
```

For the single-table syntax, the UPDATE statement updates columns of existing rows in the named table with new values. The SET clause indicates which columns to modify and the values they should be given. Each value can be given as an expression.

The keyword DEFAULT is used to set a column to its default value.

The WHERE clause specifies the conditions that identify the rows to update. With no WHERE clause, all rows are updated.

If the ORDER BY clause is specified, the rows are updated in the order that is specified. The LIMIT clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, UPDATE updates rows in each table named in table_references that satisfy the conditions. Each matching row is updated once, even if it matches the conditions multiple times. For multiple-table syntax, ORDER BY and LIMIT cannot be used.

Example

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

You can use LIMIT *row_count* to restrict the scope of the UPDATE. A LIMIT clause is a rowsmatched restriction. The statement stops as soon as it has found *row_count* rows that satisfy the WHERE clause, whether or not they actually were changed.

<u>DELETE command</u>

DELETE is a DML statement that removes rows from a table.

Single-Table Syntax

```
DELETE FROM tbl_name
[WHERE where_condition]
[ORDER BY ...]
[LIMIT row_count]
```

The DELETE statement deletes rows from *tbl_name* and returns the number of deleted rows. To check the number of deleted rows, the ROW_COUNT() function can be used.

The conditions in the optional WHERE clause identify which rows to delete. With no WHERE clause, all rows are deleted.

Where condition is an expression that evaluates to true for each row to be deleted.

If the ORDER BY clause is specified, the rows are deleted in the order that is specified. The LIMIT clause places a limit on the number of rows that can be deleted. These clauses apply to single-table deletes, but not multi-table deletes.

Multiple-Table Syntax

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
tbl_name[.*] [, tbl_name[.*]] ...
FROM table_references
[WHERE where_condition]
```

When you do not need to know the number of deleted rows, the TRUNCATE TABLE statement is a faster way to empty a table than a DELETE statement with no WHERE clause. Unlike DELETE, TRUNCATE TABLE cannot be used within a transaction or if you have a lock on the table.

You cannot delete from a table and select from the same table in a sub-query.

```
DELETE FROM orders WHERE user = 'john'
ORDER BY timestamp_column LIMIT 1
```

Data retrieval commands

The SQL SELECT statement lets you retrieve one or more records from a table — or even multiple tables at once — based on criteria that you supply. The basic syntax is:

```
SELECT fieldNames FROM tableName [WHERE criteria];
```

The SELECT statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select   FROM  which_table  WHERE conditions_to_satisfy;
```

What_to_select indicates what you want to see. This can be a list of columns, or * to indicate "all columns."

which_table indicates the table from which you want to retrieve data.

The WHERE clause is optional. If it is present, conditions_to_satisfy specifies one or more conditions that rows must satisfy to qualify for retrieval.

```
mysql> SELECT * FROM books;
-----------------------------------------------------
| id | title                        | author              | price |
-----------------------------------------------------
|  1 | The Grapes of Wrath          | John Steinbeck              | 12.99|
|  2 | Nineteen Eighty-Four         | George Orwell               |  8.99 |
|  3 | The Wind-Up Bird Chronicle   | Haruki Murakami       |  7.99 |
-----------------------------------------------------
3 rows in set (0.00 sec)
```

This SELECT query retrieves all fields (*) from the books table. Since we haven't supplied any additional criteria, the query retrieves all the records in the table, and displays the field values in the MySQL monitor

What if we want to retrieve just one record from the table, such as the book "Nineteen Eighty-Four"? To narrow down the selection, we can add a WHERE clause, like this:

```
mysql> SELECT * FROM books WHERE id = 2;
-------------------------------------------
| id | title                        | author         | price |
-------------------------------------------
|  2 | Nineteen Eighty-Four         | George Orwell  |  8.99 |
-------------------------------------------
1 row in set (0.00 sec)
```

We can also use other operators, such as < (less than), > (greater than), and the boolean AND operator, to retrieve a range of records:

```
mysql> SELECT * FROM books WHERE price < 10 AND price > 5;
-----------------------------------------------
| id | title                        | author         | price |
-----------------------------------------------
|  2 | Nineteen Eighty-Four         | George Orwell        |  8.99 |
|  3 | The Wind-Up Bird Chronicle | Haruki Murakami|  7.99 |
-----------------------------------------------
2 rows in set (0.00 sec)
```

Finally, instead of retrieving all fields using *, we can specify just the field or fields we want to retrieve. Here's an example:

```
mysql> SELECT title, author FROM books;
-----------------------------------------------
| title                              | author          |
-----------------------------------------------
| The Grapes of Wrath                | John Steinbeck          |
| Nineteen Eighty-Four               | George Orwell           |
| The Wind-Up Bird Chronicle     | Haruki Murakami|
-----------------------------------------------
3 rows in set (0.00 sec
```

You can select only particular rows from your table using WHERE clause.

mysql> SELECT * FROM student WHERE name = 'Babu';

Specifying Retrieval Criteria

To restrict the set of rows retrieved by the SELECT statement, use a WHERE clause that specifies criteria for selecting rows. You can select rows by looking for column values that satisfy various criteria, and you can look for different types of values.

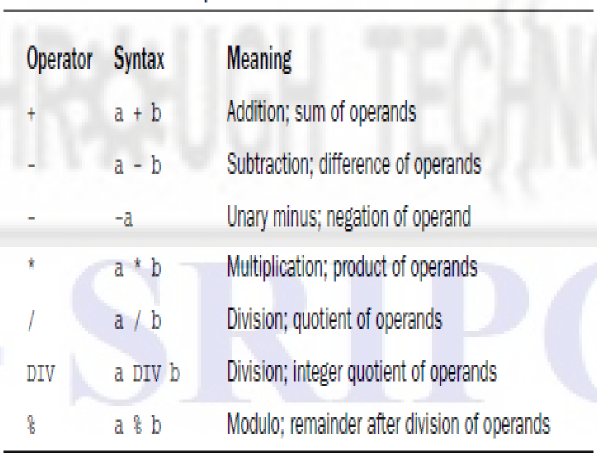
mysql> **SELECT * FROM score WHERE score > 95;**

```
- - - - - - - - - - - - - - - - - - - - - - - - - - -
| student_id   | event_id    | score |
--------------------------------------------
| 5            | 3            | 97    |
| 18           | 3                    | 96    |
| 1            | 6                    | 100   |
| 5            | 6                    | 97    |
| 11           | 6                    | 98    |
| 16           | 6                    | 98    |
--------------------------------------------
```

MySQL Operators and Expressions

Operators are symbols representing operations. They are used to combine terms in expressions to perform arithmetic, compare values, perform bitwise or logical operations, and match patterns. They are classified into different types based on their functions.

Arithmetic operators



Operator	Syntax	Meaning
+	a + b	Addition; sum of operands
-	a - b	Subtraction; difference of operands
-	-a	Unary minus; negation of operand
*	a * b	Multiplication; product of operands
/	a / b	Division; quotient of operands
DIV	a DIV b	Division; integer quotient of operands
%	a % b	Modulo; remainder after division of operands

Arithmetic operators include the usual addition, subtraction, multiplication, and division operators, as well as the modulo operator. Arithmetic is performed using BIGINT (64-bit) integer values for +, -, and * when both operands are integers. If both operands are integers, the result is unsigned if either operand is unsigned. For each operator other than DIV, if any operand is an approximate value, double precision floating-point arithmetic is used. This is also true for strings converted to numbers, because strings are converted to double-precision numbers. Integer operation which involves large values such that the result exceeds 64-bit range will result in unpredictable values.

Comparison operators

Comparison operators, shown below, include operators for testing relative magnitude or lexical ordering of numbers and strings, as well as operators for performing pattern matching and for testing NULL values. The <=> operator is MySQL-specific.

Operator	Syntax	Meaning
`=`	`a = b`	True if operands are equal
`<=>`	`a <=> b`	True if operands are equal (even if `NULL`)
`<>, !=`	`a <> b, a != b`	True if operands are not equal
`<`	`a < b`	True if `a` is less than `b`
`<=`	`a <= b`	True if `a` is less than or equal to `b`
`>=`	`a >= b`	True if `a` is greater than or equal to `b`
`>`	`a > b`	True if `a` is greater than `b`
`IN`	`a IN (b1, b2, ...)`	True if `a` is equal to any of `b1`, `b2`, ...
`BETWEEN`	`a BETWEEN b AND c`	True if `a` is between the values of `b` and `c`, inclusive
`NOT BETWEEN`	`a NOT BETWEEN b AND c`	True if `a` is not between the values of `b` and `c`, inclusive
`LIKE`	`a LIKE b`	SQL pattern match; true if `a` matches `b`
`NOT LIKE`	`a NOT LIKE b`	SQL pattern match; true if `a` does not match `b`
`REGEXP`	`a REGEXP b`	Regular expression match; true if `a` matches `b`
`NOT REGEXP`	`a NOT REGEXP b`	Regular expression match; true if `a` does not match `b`
`IS NULL`	`a IS NULL`	True if operand is `NULL`
`IS NOT NULL`	`a IS NOT NULL`	True if operand is not `NULL`

Logical Operators

Logical operators evaluate expressions to determine whether they are true (non-zero) or false (zero). It is also possible for a logical expression to evaluate to NULL if its value cannot be ascertained. For example, 1 AND NULL is of indeterminate value.

Operator	Syntax	Meaning
AND, &&	a AND b, a && b	Logical intersection; true if both operands are true
OR, ||	a OR b, a || b	Logical union; true if either operand is true
XOR	a XOR b	Logical exclusive-OR; true if exactly one operand is true
NOT, !	NOT a, !a	Logical negation; true if operand is false

In SQL, all logical operators evaluate to TRUE, FALSE, or NULL (UNKNOWN). In MySQL, these are implemented as 1 (TRUE), 0 (FALSE), and NULL. Most of this is common to different SQL database servers, although some servers may return any nonzero value for TRUE. MySQL evaluates any nonzero, non-NULL value to TRUE.

Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions. SQL pattern matching enables you to use _ (underscore) to match any single character and % (percent) to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. You do not use = or <> when you use SQL patterns; use the LIKE or NOT LIKE comparison operators instead.

To find names beginning with ***'b'*** using wildcard character.

mysql> **SELECT * FROM pet WHERE name LIKE 'b%';**

Examples for WHERE clauses that use LIKE and REGEXP operators

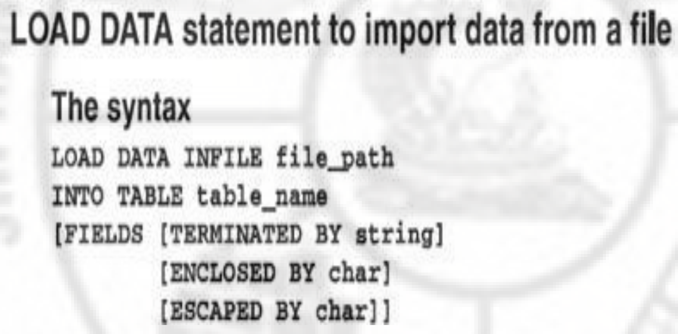
WHERE vendor_city LIKE 'SAN%'	"San Diego", "Santa Ana"
WHERE vendor_name LIKE 'COMPU_ER%'	"Compuserve", "Computerworld"
WHERE vendor_city REGEXP 'SA'	"Pasadena", "Santa Ana"
WHERE vendor_city REGEXP '^SA'	"Santa Ana", "Sacramento"
WHERE vendor_city REGEXP 'NA$'	"Gardena", "Pasadena", "Santa Ana"
WHERE vendor_city REGEXP 'RS|SN'	"Traverse City", "Fresno"
WHERE vendor_state REGEXP 'N[CV]'	"NC" and "NV" but not "NJ" or "NY"
WHERE vendor_state REGEXP 'N[A-J]'	"NC" and "NJ" but not "NV" or "NY"
WHERE vendor_contact_last_name REGEXP 'DAMI[EO]N'	"Damien" and "Damion"
WHERE vendor_city REGEXP '[A-Z][AEIOU]N$'	"Boston", "Mclean", "Oberlin"

Import and Export of data

Import data

Data can be imported using LOAD DATA statement. It is used to load data from an input file into a table. To import data the columns in the input file must match with the columns in the table. The INTO TABLE clause identifies the table name.

In the following example, the Vendor_Contacts table has three required columns: an INT column followed by two VARCHAR(50) columns. As a result, MySQL must be able to convert the data stored in the vendor_contacts.txt file to the data types specified by the Vendor_Contacts table.



LOAD DATA statement to import data from a file

The syntax

```
LOAD DATA INFILE file_path
INTO TABLE table_name
[FIELDS [TERMINATED BY string]
        [ENCLOSED BY char]
        [ESCAPED BY char]]
```

The data in the input file must not conflict with the values of any unique keys that are already stored in the rows of the table. The default delimiter for the input file is tab. If a comma delimited file is used for import, the FIELDS clause that identifies the delimiters and the escape character need to be included.

mysqlimport — A Data Import Program

The **mysqlimport** client provides a command-line interface to the LOAD DATA INFILE SQL statement. Most options to **mysqlimport** correspond directly to clauses of LOAD DATA INFILE syntax. Invoke **mysqlimport** like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, **mysqlimport** strips any extension from the file name and uses the result to determine the name of the table into which to import the file's contents.

Export data

The data in the table can be exported to an output file using INTO OUTFILE clause to a SELECT statement. By default, this clause uses a tab character (\t) to separate or delimit columns. And it uses a new line character (\n) to separate or delimit rows. This type of data file is known as *tab-delimited file.* This type of file is commonly used to store and transfer data.

The example given below, exports data from the Vendor_Contacts table and stores it in a tab-delimited file named vendor_contacts.txt. If necessary, the number of records exported can be limited by including a column list and a WHERE clause in the SELECT statement.

To export data to a comma-delimited file, you can include a FIELDS clause with TERMINATED BY clause that indicates that every column should be terminated by acomma (,) and an ENCLOSED BY clause that indiacates that each column should be enclosed by double quotes (“). The exported data will be in comma-delimited format.

If the data need to contain a double quote character, you need to include an ESCAPED BY clause to specify an escape character. In the second example, the backslash character is used as the escape character

The syntax of the SELECT for exporting data to a file

```
SELECT column_list
INTO OUTFILE file_path
[FIELDS [TERMINATED BY string]
        [ENCLOSED BY char]
        [ESCAPED BY char]
FROM table name
[WHERE search_condition]
[ORDER BY order_by_list]
```

A tab-delimited file

The statement

```
SELECT *
INTO OUTFILE '/mysql/vendor_contacts.txt'
FROM vendor_contacts
```

The file contents

```
5     Davison      Michelle
12    Mayteh       Kendall
17    Onandonga    Bruce
44    Antavius     Anthony
76    Bradlee      Danny
94    Suscipe      Reynaldo
101   O'Sullivan   Geraldine
123   Bucket       Charles
```

.

Built-in Functions

MySQL has number of built-in functions to manipulate various data items. These functions accept zero or more arguments. Functions are called to perform a calculation and return a value. By default, functions must be invoked with no space between the function name and the parenthesis following it. Multiple arguments to a function are separated by commas. Spaces are allowed around function arguments.

These functions are mainly classified into

- Single Row functions or scalar functions.
- Group or aggregate functions.

Single row functions

Single row functions return only one value for every row queried in the table. It can appear in a select statement and can be included in the WHERE clause. The single row functions are divided into

(i) String functions
(ii) Arithmetic functions
(iii) Date and Time functions

(i) String functions

Name	Description
ASCII()	Return numeric value of left-most character
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument equivalent to length()
CONCAT()	Return concatenated string
HEX()	Return a hexadecimal representation of a decimal or string value
INSERT()	Insert a substring at the specified position up to the specified number of characters
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
MID()	Return a substring starting from the specified position
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings

ASCII(*str*)

Returns the numeric value of the leftmost character of the string *str*. Returns 0 if *str* is the empty string. Returns NULL if *str* is NULL. ASCII() works for 8-bit characters.

```
mysql> SELECT ASCII('2');
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

CHAR(N,...)

CHAR() interprets each argument N as an integer and returns a string consisting of the characters given by the code values of those integers. NULL values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
        -> 'MMM'
```

CHAR_LENGTH(str) or CHARACTER_LENGTH(str)

These functions are similar to LENGTH(), except that the argument length is counted in characters, not bytes.

CONCAT(str1,str2,...)

Returns the string that results from concatenating the arguments. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent nonbinary string form. CONCAT() returns NULL if any argument is NULL.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
        -> NULL
mysql> SELECT CONCAT(14.3);
        -> '14.3'
```

INSERT(str, pos, len, newstr)

Returns the string *str*, with the substring beginning at position *pos* and *len* characters long replaced by the string *newstr*. Returns the original string if *pos* is not within the length of the string. Replaces the rest of the string from position *pos* if *len* is not within the length of the rest of the string. Returns NULL if any argument is NULL.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
```

```
        -> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
        -> 'QuWhat'
```

INSTR(str,substr)

Returns the position of the first occurrence of substring substr in string str. This is the same as the two-argument form of LOCATE(), except that the order of the arguments is reversed.

```
mysql> SELECT INSTR('foobarbar', 'bar');
        -> 4
mysql> SELECT INSTR('xbar', 'foobar');
        -> 0
```

LCASE(*str*)

LCASE() is a synonym for LOWER().

LEFT(*str*, *len*)

Returns the leftmost *len* characters from the string *str*, or NULL if any argument is NULL.

```
mysql> SELECT LEFT('foobarbar', 5);
        -> 'fooba'
```

LENGTH (*str*)

Returns the length of the string *str*, measured in bytes. A multibyte character counts as multiple bytes. This means that for a string containing five 2-byte characters, LENGTH() returns 10, whereas CHAR_LENGTH() returns 5.

```
mysql> SELECT LENGTH('text');
        -> 4
```

MID(*str*,*pos*,*len*)

MID(str, pos, len) is a synonym for SUBSTRING(str, pos, len).

REPEAT(*str*,*count*)

Returns a string consisting of the string *str* repeated *count* times. If *count* is less than 1, returns an empty string. Returns NULL if *str* or *count* are NULL.

```
mysql> SELECT REPEAT('MySQL', 3);
        -> 'MySQLMySQLMySQL'
```

REPLACE(*str*,*from_str*,*to_str*)

Returns the string *str* with all occurrences of the string *from_str* replaced by the string *to_str*. REPLACE() performs a case-sensitive match when searching for *from_str*.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

REVERSE(*str*)

Returns the string *str* with the order of the characters reversed.

```
mysql> SELECT REVERSE('abc');
        -> 'cba'
```

RIGHT(str,len)

Returns the rightmost *len* characters from the string *str*, or NULL if any argument is NULL.

```
mysql> SELECT RIGHT('foobarbar', 4);
        -> 'rbar'
```

UPPER(str)

Returns the string str with all characters changed to uppercase.

```
mysql> SELECT UPPER('Hej');

        -> 'HEJ'
```

(ii) Arithmetic Functions

Name	**Description**
ABS()	Return the absolute value
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
COS()	Return the cosine
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
MOD()	Return the remainder
PI()	Return the value of pi
POW()	Return the argument raised to the specified power
ROUND()	Round the argument

SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
TRUNCATE()	Truncate to specified number of decimal places

All mathematical functions return NULL in the event of an error.

ABS(*X*)

Returns the absolute value of *X*.

```
mysql> SELECT ABS(2);
                -> 2
mysql> SELECT ABS(-32);
                -> 32
```

This function is safe to use with BIGINT values.

CEIL(X)

CEIL() is a synonym for CEILING().

CEILING(*X*)

Returns the smallest integer value not less than *X*.

```
mysql> SELECT CEILING(1.23);
       -> 2
mysql> SELECT CEILING(-1.23);
       -> -1
```

For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

COS(*X*)

Returns the cosine of *X*, where *X* is given in radians.

```
mysql> SELECT COS(PI());
       -> -1
```

EXP(*X*)
Returns the value of *e* (the base of natural logarithms) raised to the power of *X*. The inverse of this function is LOG() (using a single argument only) or LN().

```
mysql> SELECT EXP(2);
       -> 7.3890560989307


mysql> SELECT EXP(-2);
```

```
        -> 0.13533528323661
mysql> SELECT EXP(0);
        -> 1
```

FLOOR(*X*)

Returns the largest integer value not greater than *X*.

```
mysql> SELECT FLOOR(1.23), FLOOR(-1.23);
        -> 1, -2
```

For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

LN(*X*)

Returns the natural logarithm of *X*; that is, the base-*e* logarithm of *X*. If *X* is less than or equal to 0, then NULL is returned.

```
mysql> SELECT LN(2);
        -> 0.69314718055995
mysql> SELECT LN(-2);
        -> NULL
```

This function is synonymous with LOG(*X*). The inverse of this function is the EXP() function.

LOG(*X*), LOG(*B*,*X*)

If called with one parameter, this function returns the natural logarithm of *X*. If *X* is less than or equal to 0, then NULL is returned. The inverse of this function (when called with a single argument) is the EXP() function.

```
mysql> SELECT LOG(2);
        -> 0.69314718055995
mysql> SELECT LOG(-2);
        -> NULL
```

If called with two parameters, this function returns the logarithm of *X* to the base *B*. If *X* is less than or equal to 0, or if *B* is less than or equal to 1, then NULL is returned.

```
mysql> SELECT LOG(2,65536);
        -> 16
mysql> SELECT LOG(10,100);
        -> 2

mysql> SELECT LOG(1,100);
        -> NULL
```

MOD(N,M), N % M, N MOD M

Modulo operation. Returns the remainder of *N* divided by *M*.

```
mysql> SELECT MOD(234, 10);
        -> 4
mysql> SELECT 253 % 7;
        -> 1
mysql> SELECT MOD(29,9);
        -> 2
mysql> SELECT 29 MOD 9;
        -> 2
```

This function is safe to use with BIGINT values.

MOD() also works on values that have a fractional part and returns the exact remainder after division

```
mysql> SELECT MOD(34.5,3);
        -> 1.5
```

MOD(*N*,0) returns NULL.

PI( )

Returns the value of π (pi). The default number of decimal places displayed is seven, but MySQL uses the full double-precision value internally.

```
mysql> SELECT PI();
        -> 3.141593
mysql> SELECT PI()+0.000000000000000000;
        -> 3.141592653589793116
```

POW(*X*,*Y*)

Returns the value of *X* raised to the power of *Y*.

```
mysql> SELECT POW(2,2);
        -> 4
mysql> SELECT POW(2,-2);
        -> 0.25
```

POWER(*X*,*Y*)

This is a synonym for POW().

ROUND(X), ROUND(X,D)

Rounds the argument X to D decimal places. The rounding algorithm depends on the data type of X. D defaults to 0 if not specified. D can be negative to cause D digits left of the decimal point of the value X to become zero.

```
mysql> SELECT ROUND(-1.23);
        -> -1
mysql> SELECT ROUND(-1.58);
        -> -2
mysql> SELECT ROUND(1.58);
        -> 2
mysql> SELECT ROUND(1.298, 1);
        -> 1.3
mysql> SELECT ROUND(1.298, 0);
        -> 1
mysql> SELECT ROUND(23.298, -1);
        -> 20
```

The return value has the same type as the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places):

```
mysql> SELECT ROUND(150.000,2), ROUND(150,2);
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| ROUND(150.000,2)       | ROUND(150,2)           |
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
|          150.00              |          150          |
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

<u>SIGN(*X*)</u>
Returns the sign of the argument as -1, 0, or 1, depending on whether *X* is negative, zero, or positive.

```
mysql> SELECT SIGN(-32);
        -> -1
mysql> SELECT SIGN(0);
        -> 0
mysql> SELECT SIGN(234);
        -> 1
```

SQRT(X)

Returns the square root of a nonnegative number X.

```
mysql> SELECT SQRT(4);
        -> 2
mysql> SELECT SQRT(20);
        -> 4.4721359549996
mysql> SELECT SQRT(-16);
        -> NULL
```

TAN(X)

Returns the tangent of X, where X is given in radians.

```
mysql> SELECT TAN(PI());
        -> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
        -> 1.5574077246549
```

TRUNCATE(X,D)

Returns the number X, truncated to D decimal places. If D is 0, the result has no decimal point or fractional part. D can be negative to cause D digits left of the decimal point of the value X to become zero.

```
mysql> SELECT TRUNCATE(1.223,1);
        -> 1.2
mysql> SELECT TRUNCATE(1.999,1);
        -> 1.9
mysql> SELECT TRUNCATE(1.999,0);
        -> 1
mysql> SELECT TRUNCATE(-1.999,1);
        -> -1.9
mysql> SELECT TRUNCATE(122,-2);
       -> 100
mysql> SELECT TRUNCATE(10.28*100,0);
       -> 1028
```

All numbers are rounded toward zero.

(iii) Date and Time Functions

Name	Description
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
CURDATE()	Return the current date
CURTIME()	Return the current time
DATE()	Extract the date part of a date or datetime expression
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time

SECOND()	Return the second (0-59)
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or date time argument converted to seconds since Year 0
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

This section describes the functions that can be used to manipulate temporal (date and time) values.

DAY(*date*)

DAY() is a synonym for DAYOFMONTH().

DAYNAME(*date*)

Returns the name of the weekday for *date*.

```
mysql> SELECT DAYNAME('2007-02-03');
        -> 'Saturday'
```

DAYOFMONTH(date)

Returns the day of the month for date, in the range 1 to 31, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero day part.

```
mysql> SELECT DAYOFMONTH('2007-02-03');
        -> 3
```

DAYOFWEEK(date)

Returns the weekday index for date (1 = Sunday, 2 = Monday, …, 7 = Saturday)

```
mysql> SELECT DAYOFWEEK('2007-02-03');
        -> 7
```

DAYOFYEAR(date)

Returns the day of the year for date, in the range 1 to 366.

```
mysql> SELECT DAYOFYEAR('2007-02-03');
        -> 34
```

HOUR(time)

Returns the hour for time. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.

```
mysql> SELECT HOUR('10:05:03');
        -> 10
mysql> SELECT HOUR('272:59:59');
        -> 272
```

LAST_DAY(date)

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

```
 mysql> SELECT LAST_DAY('2003-02-05');
        -> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
        -> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
        -> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
        -> NULL
```

MINUTE(time)

Returns the minute for time, in the range 0 to 59.

```
mysql> SELECT MINUTE('2008-02-03 10:05:03');
        -> 5
```

MONTH(date)

Returns the month for date, in the range 1 to 12 for January to December, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero month part.

```
mysql> SELECT MONTH('2008-02-03');
        -> 2
```

MONTHNAME(date)

Returns the full name of the month for date.

```
mysql> SELECT MONTHNAME('2008-02-03');
        -> 'February'
```

NOW()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT NOW();
        -> '2007-12-15 23:50:26'
mysql> SELECT NOW() + 0;
        -> 20071215235026.000000
```

WEEKDAY(date)

Returns the weekday index for date (0 = Monday, 1 = Tuesday, … 6 = Sunday).

```
mysql> SELECT WEEKDAY('2008-02-03 22:23:00');
        -> 6
mysql> SELECT WEEKDAY('2007-11-06');
        -> 1
```

SYSDATE()

```
mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
-------------------------------------------------------------------------------
| SYSDATE()                | SLEEP(2)       | SYSDATE()                |
-------------------------------------------------------------------------------
| 2006-04-12 13:47:44      |       0        | 2006-04-12 13:47:46      |
-------------------------------------------------------------------------------
```

ADDDATE

ADDDATE(date,INTERVAL expr unit), ADDDATE(expr,days)

When invoked with the INTERVAL form of the second argument, ADDDATE() is a synonym for
DATE_ADD(). The related function SUBDATE() is a synonym for DATE_SUB().

```
mysql> SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
-> '2008-02-02'
mysql> SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
-> '2008-02-02'
```

When invoked with the *days* form of the second argument, MySQL treats it as an integer number of days to be added to *expr*.

```
mysql> SELECT ADDDATE('2008-01-02', 31);
-> '2008-02-02'
```

2.4.2 Aggregate Functions

Name	**Description**
AVG()	Return the average value of the argument
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
MAX()	Return the maximum value
MIN()	Return the minimum value

STD(expr)
STDDEV(expr)
	Return the population standard deviation

STDDEV_POP(expr)	
SUM()	Return the sum

VARIANCE()
VAR_POP(expr)
	Return the population standard variance

Assume the existence of a table mytbl with an integer column mycol that contains eight rows with the values 1, 3, 5, 5, 7, 9, 9, and NULL.

mysql> **SELECT mycol FROM mytbl;**

```
-----------
| mycol |
| 1 |
-----------------------------------------------------------------
| 3 |
| 5 |
| 5 |
| 7 |
| 9 |
| 9 |
| NULL |
- - - - - -
```

The following function calls return the values given as below.

SELECT AVG(mycol) FROM mytbl;   □□

5.5714

SELECT COUNT(mycol) FROM mytbl;

7 (Without NULL)

SELECT COUNT(*) FROM mytbl;

8 (including NULL value)

SELECT COUNT(DISTINCT mycol) FROM mytbl;

5

SELECT MAX(mycol) FROM mytbl;

9

SELECT MIN(mycol) FROM mytbl;

1

SELECT STDDEV_POP(mycol) FROM mytbl;

2.7701

SELECT SUM(mycol) FROM mytbl;

39

```
SELECT VAR_POP(mycol) FROM mytbl;
7.6735
```

Conversion functions

By default MySQL attempts to convert values to the type required by an expression rather than generating an error. Depending on the context, it converts values of each of the three general categories (numbers, strings, or dates and times) to values in any of the other categories.

Implicit type conversion

Automatically converting one data type to another by MySQL is known as implicit conversion. When an expression involves different types of values, MySQL implicitly converts them when it evaluates the expression.

Explicit type conversion

A data of one type can be explicitly converted to the desired type by using CAST and CONVERT functions.

CAST(expr AS type)

The CAST() function takes an expression of any type and produces a result value of the specified type, similar to CONVERT().

Examples: CAST(304 AS BINARY) → '304'
CAST(13 AS DECIMAL(5,2)) → 13.00

CAST() can be useful for forcing columns to have a particular type when creating a new table with CREATE TABLE ... SELECT.

```
mysql> CREATE TABLE t SELECT CAST(20080101 AS DATE) AS date_val;
```

Querying the table

The tables have been created and loaded with data. To retrieve and display information from these tables the SELECT statement is used. It enables you to retrieve the desired information.

The entire contents of a table can be displayed using

```
SELECT * FROM table_name;
```

Or particular column of a single row can be selected using column name

```
SELECT birth FROM table_name WHERE column=value;
```

The SELECT statement has several clauses that can be combined as necessary to retrieve the required information.

The syntax of SELECT statement is:

```
SELECT [ALL | DISTINCT | DISTINCTROW] select_expr [, select_expr ...]
[FROM table_references
[WHERE where_condition]
[GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
[HAVING where_condition]
[ORDER BY {col_name | expr | position} [ASC | DESC], ...]
```

In SELECT statement, different clauses can be used.

WHERE clause

To restrict the set of rows retrieved by the SELECT statement use a WHERE clause that specifies criteria for selecting rows. You can select rows by looking for column values that satisfy various criteria, and you can look for different types of values. For example, you can search for certain numeric values:

```
mysql> SELECT * FROM score WHERE score > 95;
------------------------------------
| student_id  | event_id   | score |
------------------------------------
| 5           | 3          | 97    |
| 18          | 3          | 96    |
| 1           | 6          | 100   |
| 5           | 6          | 97    |
| 11          | 6          | 98    |
| 16          | 6          | 98    |
------------------------------------
```

Expressions in WHERE clause can use arithmetic operators, comparison operators, and logical operators. The parentheses can be used to group parts of an expression. Operations can be performed using constants, table columns, and function calls.

The WHERE clause, indicates the condition or conditions that rows must satisfy to be selected. ***where_condition*** is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no WHERE clause. In the WHERE expression any of the functions and operators that MySQL supports can be used except for aggregate (summary) functions.

The syntax of the WHERE clause with an IN phrase

```
        WHERE test_expression [NOT] IN ({subquery | expression_1 [, expression2]
… })
```

Examples of the IN phrase

An IN phrase with a list of numeric literals

```
WHERE terms_id IN (1, 3, 4)
```

An IN phrase preceded by NOT

```
WHERE vendor_state NOT IN ('CA', 'NV', 'OR')
```

An IN phrase with a subquery

```
WHERE vendor_id IN
    (SELECT vendor_id
     FROM invoices
     WHERE invoice_date = '2011-07-18')
```

The syntax of the WHERE clause with a BETWEEN phrase

```
WHERE test_expression [NOT] BETWEEN begin_expression AND end_expression
```

Examples of the BETWEEN
phrase

A BETWEEN phrase with literal values

```
WHERE invoice_date BETWEEN '2011-06-01' AND '2011-06-30'
```

A BETWEEN phrase preceded by NOT

```
WHERE vendor_zip_code NOT BETWEEN 93600 AND 93799
```

A BETWEEN phrase with a test expression coded as a calculated value

```
WHERE invoice_total - payment_total - credit_total BETWEEN 200 AND 500
```

A BETWEEN phrase with the upper and lower limits coded as calculated values

```
WHERE payment_total BETWEEN credit_total AND credit_total + 500
```

The syntax of the WHERE clause with a LIKE phrase

WHERE match_expression {NOT} LIKE pattern

ORDER BY clause

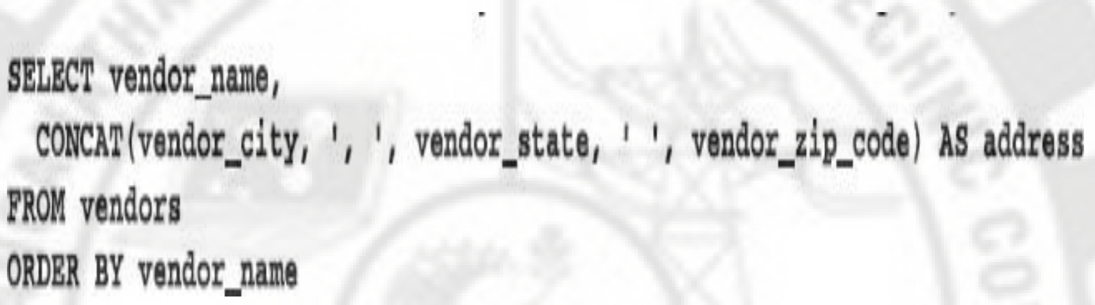
The ORDER BY clause specifies the sort order for the rows in a result set retrieved by the SELECT statement. The selected rows can be ordered either in ascending or descending order. Ascending is the default sequence, the ASC keyword can be omitted. The sort order can be explicitly specified by using ASC or DESC keywords after column names in the ORDER BY clause.

The records (rows) can be sorted using more than one column, and each column can be sorted independently in ascending or descending order. To sort by more than one column, the column names must be separated by comma. This can be referred to as nested sort because one sort is nested within another.

```
ORDER BY expression [ASC|DESC][, expression [ASC|DESC]] ...
```

The expanded syntax of the ORDER BY clause

The ORDER BY clause that sorts by one column in ascending sequence



```
SELECT vendor_name,
  CONCAT(vendor_city, ', ', vendor_state, ' ', vendor_zip_code) AS address
FROM vendors
ORDER BY vendor_name
```

Columns selected for output can be referred to in ORDER BY and GROUP BY clauses using column names, column aliases, or column positions

SELECT college, region, state FROM tournament ORDER BY region, state;

GROUP BY clause

The GROUP BY clause determines how the selected rows are grouped and the HAVING clause determines which groups are included in the final results. These clauses are used in the SELECT statement after the WHERE clause, but before the ORDER BY clause. This makes sense because the WHERE clause is applied before the rows are grouped, and the ORDER BY clause is applied after the rows are grouped.

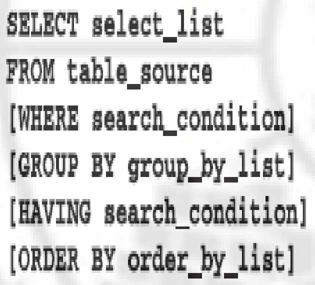
One or more columns or expressions are separated by commas in the GROUP BY clause. The rows in the result set are grouped by those columns or expressions in ascending sequence. That means a single row is returned for each unique set of values in the GROUP BY columns. The following query determines how many presidents were born in each month of the year:

```
mysql> SELECT MONTH(birth) AS Month, MONTHNAME(birth) AS Name,
->     COUNT (*) AS count FROM president GROUP BY Name ORDER BY
Month;
- - - - - - - - - - - - - - - - - - - - - - - - - - -
| Month        | Name          | count |
- - - - - - - - - - - - - - - - - - - - - - - - - - -
| 1            | January       | 4 |
| 2            | February      | 4 |
| 3            | March         | 4 |
| 4            | April         | 4 |
| 5            | May           | 2 |
| 6            | June          | 1 |
| 7            | July          | 4 |
| 8            | August        | 4 |
| 9            | September     | 1 |
| 10           | October       | 6 |
| 11           | November      | 5 |
| 12          | December   | 3 |
--------------------------
```

If aggregated functions are included in the SELECT clause, the aggregate is calculated for each group. If two or more expressions are included in the GROUP BY clause, they form a hierarchy where each column or expression is subordinate to the previous one.

HAVING clause

The syntax of SELECT with different clauses (as we have seen before)



```
SELECT select_list
FROM table_source
[WHERE search_condition]
[GROUP BY group_by_list]
[HAVING search_condition]
[ORDER BY order_by_list]
```

The HAVING clause specifies a search condition for a group or an aggregate. MySQL applies this condition after it groups the rows that satisfy the search condition in the WHERE clause. The SQL standard requires that HAVING must refer only columns in the GROUP BY clause or columns used in aggregate functions.

However, MySQL supports an extension to this behavior, and permits HAVING to refer to columns in the SELECT list and columns in outer sub-queries as well.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

The HAVING clause must follow the GROUP BY clause in a query and must also precedes the ORDER BY clause if used.

The HAVING clause can refer to aggregate functions, which the WHERE clause cannot:

```
SELECT user, MAX(salary) FROM users
  GROUP BY user HAVING MAX(salary) > 10;
```

HAVINNG clause can include aggregate functions, but the WHERE clause cannot. Because the search condition in a WHERE clause is applied before the rows are grouped.

The WHERE clause can refer to any column in the base tables. The HAVING clause can only refer to columns included in the SELECT clause.

The HAVING clause is applied nearly last, just before items are sent to the client. (LIMIT is applied after HAVING.)

Sub-queries

A sub-query is a SELECT statement that is coded within another SQL statement. Sub-queries can be introduced or coded in WHERE, HAVING, FROM or SELECT clause of a SELECT statement. Sub-queries must be enclosed within parentheses.

Four ways to introduce a sub-query in a SELECT statement

- In a WHERE clause as a search condition
- In a HAVING clause as a search condition
- In the FROM clause as a table specification
- In the SELECT clause as a column specification.

Sub-query support is a capability that allows one SELECT statement to be written within parentheses and nested inside another. Here's an example that looks up the IDs for grade event rows that correspond to tests ('T') and uses them to select scores for those tests:

```
SELECT * FROM score
WHERE event_id IN (SELECT event_id FROM grade_event WHERE category =
'T');
```

Sub-queries can return different types of information:

- A scalar sub-query returns a single value.
- A column sub-query returns a single column of one or more values.
- A row sub-query returns a single row of one or more values.
- A table sub-query returns a table of one or more rows of one or more columns.

Sub-query results can be tested in different ways (Operators used in subqueries)

- Scalar sub-query results can be evaluated using relative comparison operators such as = or <.
- IN and NOT IN test whether a value is present in a set of values returned by a sub-query
- ALL, ANY, and SOME compare a value to the set of values returned by a sub-query.
- EXISTS and NOT EXISTS test whether a sub-query result is empty.
- Can be used in the FROM clause in place of a table.

For example, to identify the scores for the quiz that took place on '2008-09-23', use a scalar sub-query to determine the quiz event ID and then match score rows against that ID in the outer SELECT:

SELECT * FROM score WHERE event_id = (SELECT event_id FROM grade_event
WHERE date = '2008-09-23' AND category = 'Q');

With this form of statement, where the sub-query is preceded by a value and a relative comparison operator, it is necessary that the sub-query produce a single value.

Operators used in sub-queries

The =, <>, >, >=, <, and <= operators perform relative-value comparisons. When used with a scalar sub-query, they find all rows in the outer query that stand in particular relationship to the value returned by the sub-query.

```
SELECT * FROM score
WHERE event_id =
(SELECT event_id FROM grade_event
WHERE date = '2008-09-23' AND category = 'Q');
```

The IN and NOT IN operators can be used when a sub-query returns multiple rows to be evaluated in comparison to the outer query. They test whether a comparison value is present in a set of values.

IN is true for rows in the outer query that match any row returned by the sub-query.

NOT IN is true for rows in the outer query that match no rows returned by the sub-query.

The following statements use IN and NOT IN to find those students who have absences listed in the absence table, and those who have perfect attendance

```
mysql> SELECT * FROM student
-> WHERE student_id IN (SELECT student_id FROM absence);
- - - - - - - - - - - - - - - - - - - - - - - - - - -
| name          | sex  | student_id |
---------------------------------------
| Kyle     | M    | 3     |
| Abby          | F     | 5      |
| Peter    | M    | 10    |
| Will     | M    | 17    |
| Avery         | F     | 20     |
---------------------------------------
```

```
mysql> SELECT * FROM student
-> WHERE student_id NOT IN (SELECT student_id FROM absence);
```

IN and NOT IN also work for sub-queries that return multiple columns. In other words, you can use them with table sub-queries. In this case, use a row constructor to specify the comparison values to test against each column:

```
mysql> SELECT last_name, first_name, city, state FROM president
-> WHERE (city, state) IN
-> (SELECT city, state FROM president
-> WHERE last_name = 'Roosevelt');
```

The ALL and ANY operators are used in conjunction with a relative comparison operator to test the result of a column sub-query. They test whether the comparison value stands in particular relationship to all or some of the values returned by the sub-query

The EXISTS and NOT EXISTS operators merely test whether a sub query returns any rows. If it does, EXISTS is true and NOT EXISTS is false. The following statements show some trivial examples of these sub queries. The first returns 0 if the absence table is empty, the second returns 1:

```
SELECT EXISTS (SELECT * FROM absence);
SELECT NOT EXISTS (SELECT * FROM absence);
```

EXISTS and NOT EXISTS are commonly used in correlated sub-queries.

Correlated Sub-queries

Sub queries can be uncorrelated or correlated:

An uncorrelated sub query contains no references to values from the outer query. An uncorrelated sub query can be executed by itself as a separate statement. For example, the sub query in the following statement is uncorrelated because it refers only to the table t1 and not to t2:

```
SELECT   j   FROM   t2   WHERE   j  IN (SELECT i FROM t1);
```

A correlated sub query contains references to values from the outer query, and thus is dependent on it. Due to this linkage, a correlated sub query cannot be executed by itself as a separate statement. For example, the sub query in the following statement is true for each value of column j in t2 that matches a column i value in t1:

```
SELECT j FROM t2 WHERE (SELECT i FROM t1 WHERE i = j);
```

Correlated sub-queries are commonly used for EXISTS and NOT EXISTS sub-queries, which are useful for finding rows in one table that match or don’t match rows in another. Correlated sub-queries work by passing values from the outer query to the sub-query to see whether they match the conditions specified in the sub-query. For this reason, it is necessary to qualify column names with table names if they are ambiguous (appear in more than one table).

The following EXISTS sub-query identifies matches between the tables—that is, values that are present in both. The statement selects students who have at least one absence listed in the absence table:

```
        SELECT student_id, name FROM student WHERE EXISTS
        (SELECT * FROM absence WHERE absence.student_id =
student.student_id);
```

NOT EXISTS identifies non-matches—values in one table that are not present in the other. This statement selects students who have no absences:

```
        SELECT student_id, name FROM student WHERE NOT EXISTS
        (SELECT * FROM absence WHERE absence.student_id =
student.student_id);
```

Flow control<u>IF ()</u>

The IF() function tests an expression and return one value if the expression is true and another value if the expression is false.

Syntax:
IF(expr1,expr2,expr3)

If expr1 is TRUE (expr1 <> 0 and expr1 <> NULL) then IF() returns expr2 otherwise it returns expr3. IF() returns a numeric or string value, depending on the context in which it is used.

Example

```
mysql> select if(5%2=1,'ODD','EVEN');
+________________+
| if(5%2=1,'ODD','EVEN') |
+________________+
| ODD               |
+________________+
1 row in set (0.07 sec)
```

<u>IFNULL()</u>

Syntax
IFNULL (expr1, expr2)

Returns expr2 if the value of the expression expr1 is NULL; otherwise, it returns expr1.
IFNULL () returns a number or string according to the context in which it is used.
IFNULL (NULL, 'null') → 'null'
IFNULL ('not null', 'null') → 'not null'
<u>CASE</u>
Syntax

```
        CASE [expr]
                WHEN expr1 THEN stmt_list1
                [WHEN expr2 THEN stmt_list2] ...
                [ELSE stmt_list]
```

END IF

The CASE statement provides a branching flow-control construct. When the initial expression, expr, is present, CASE compares it to the expression following each WHEN. For the first one that is equal, the statement list for the corresponding THEN value is executed. This is useful for comparing a given value to a set of values.

CASE variable

```
  WHEN 1 THEN
    SELECT 'Net due 10 days' AS Terms;
  WHEN 2 THEN
    SELECT 'Net due 20 days' AS Terms;
  WHEN 3 THEN
    SELECT 'Net due 30 days' AS Terms;
  ELSE
    SELECT 'Net due more than 30 days' AS Terms;
 END CASE;
```

When the initial expression, expr, is not present, CASE evaluates WHEN expressions. For the first one that is true, the statement list for the corresponding THEN value is executed. This is useful for performing non-equality tests or testing arbitrary conditions. If no WHEN expression matches, the statement list for the ELSE clause is executed, if there is one.

LOOP

Syntax

LOOP statement_list END LOOP

label: LOOP statement_list END LOOP [label]

This statement sets up an execution loop. The statements within the loop execute repeatedly until control is transferred out of the loop.

Example for simple loop:

```
 testLoop : LOOP
   SET s = CONCAT(s, 'i=', i, ' | ');
   SET i = i + 1;

   IF i = 4 THEN
      LEAVE testLoop;
   END IF;
 END LOOP testLoop;
```

LEAVE

Syntax

```
LEAVE label;
```

The LEAVE statement is used to exit a labeled flow-control construct. The statement must appear within the construct that has the given label.

Example

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
        label1: LOOP
        SET p1 = p1 + 1;
        IF p1 < 10 THEN
        ITERATE label1;
        END IF;
        LEAVE label1;
END LOOP label1;
SET @x = p1;
END;
```

This statement sets up an execution loop. The statements within the loop execute repeatedly until control is transferred out of the loop.

ITERATE

Syntax

```
ITERATE label
```

The ITERATE statement is used within looping constructs to begin the next iteration of the loop. It can appear within LOOP, REPEAT, and WHILE.

WHILE

Syntax

```
        WHILE expr DO
                statement_list
        END WHILE;
```

Labeled WHILE

```
        label: WHILE expr DO
                        statement_list
                END WHILE [label];
```

This statement sets up an execution loop. The statements within the loop execute repeatedly
as long as the expression **expr** is true.

Example:

```
CREATE PROCEDURE dowhile()
BEGIN
        DECLARE v1 INT DEFAULT 5;
        WHILE v1 > 0 DO
                ...
                SET v1 = v1 - 1;
        END WHILE;
END;
```

REPEAT

Syntax

```
        REPEAT statement_list UNTIL expr END REPEAT

label: REPEAT statement_list UNTIL expr END REPEAT [label]
```

This statement sets up an execution loop. The statements within the loop execute repeatedly until the expression **expr** is true
This works similar to the WHILE loop except that the condition is specified at the end of the loop. As a result, a REPEAT loop always executes at least once.

Example:

```
Mysql> CREATE PROCEDURE dorepeat(p1 INT)
-> BEGIN
-> SET @x = 0;
-> REPEAT
-> SET @x = @x + 1;
-> UNTIL @x > p1 END REPEAT;
-> END
```

Review Questions

PART A

1. What is MySQL
2. Give the syntax of creating database
3. Give the syntax of describing database
4. What are the typesof MySQL datatypes
5. Give the syntax of renaming a table
6. What is the use of truncate table command
7. What are the data manipulation commands
8. What is an expression
9. What is the use of CURDATE()
10. What is the use of aggregate functions
11. What is the use of IFNULL()

PART B

1. List some of the MySQL features
2. How will you create a database? Explain
3. Name some numeric datatypes
4. Name some string data types
5. Explain the difference between drop table command and truncate table command
6. Name some aggregate functions
7. Explain HAVING clause
8. Explain LEAVE command and ITERATE command

PART C

1. Explain how will you install MySQL on windows
2. Explain about MySQL data types

3. Explain data definition commands in MySQL
4. Explain in detail about data manipulation commands
5. Explain data retrieval commands
6. Explain in detail about MySQL operators
7. Explain single row functions
8. Explain flow control statements used in MySQL
9. Explain GROUP BY,ORDER BY and HAVING clauses used in SELECT statement.

UNIT III - MYSQL PERFORMANCE TUNING

Learning Objectives

At the end of the unit, the students will be able to

- Understand index and its usage
- Create, alter and delete sequence
- Perform various types of Join operation
- Describe the usage of Union
- Explain the concept of View, create view, update view and delete view
- Create and delete user
- Grant and revoke privileges
- Handle transactions in database using commit and rollback commands

Introduction

Database in any Enterprise consists of thousands and lakhs of records in a table. Queries are often raised to retrieve records from the table. Index is data structure technique used to speed up the performance of queries.Sequence is a feature supported by some database systems to produce unique values on demand.

In databases, the required data may be available in two or more tables. Join is used for combining column from two or more tables by using values common to both tables.Union operator is used to combine the result sets of two or more “select” statements.

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. A view can be queried similar to table.

“grant” and “revoke” are Data Control Language (DCL) commands used to enforce database security in a multiple user database environment. Grant command is used to grant privileges to other uses and revoke command is used to cancel the given privileges.

“commit” and “rollback” are Transaction Control Language (TCL) commands used to manage transactions in database. Commit command is used to save the changes to the database permanently and ends the transaction. Rollback command is used to undo the changes to the database and ends the transaction.

- INDEXES & SEQUENCES

INDEX

A database index is a data structure that improves the speed of operations in a table. Index is used to find data more quickly and efficiently, without reading the whole table.

Updating a table with indexes takes more time than updating a table without indexes because the indexes also need to be updated. So indexesshould be created on columns and tables that will be frequently searched.

Advantages of Index

<u>(1)</u> <u>Fast query execution</u>
Query can find the data faster when index fields are used for searching.

<u>(2)</u> <u>Reduced disk I/O</u>
Since the index directly points to the location of the row containing the value, there is less disk I/O operations.

<u>(3)</u> <u>Data independence</u>
We can create and drop index at any time without affecting the data of the table.

Creating Index

“create index” command is used to create indexes in tables. Indexes can be created while creating tables using “create table” and also using “alter table” command.

<u>Creating Index with “create index” command</u>

create [unique | fulltext | spatial] index *index_name* **on** *table_name* (*column_list*) **;**

where

create, index, on	-	keywords
unique | fulltext | spatial	-	type of index
index_name	-	name of the index
table_name	-	name of the table
column_list	-	column name(s) on which index is created

<u>Example</u>
mysql >create index dept_index on employee (deptno) ;

<u>Creating Index with “create table” command</u>

create table *table_name* (*column_name1 type, column_name2 type*, **… ,**

index*index_name* (*column list*)) **;**

Example

mysql >create table employee (empno int , name varchar(40) , deptno int , salary int ,

index dept_index (deptno));

Creating Index with "alter table" command

alter table*table_name***add index***index_name* (*column list*) ;

Example

mysql >alter table employee add index dept_index (deptno) ;

Displaying Index Information

The command to view the index information created on a table along with index names is

>**show index from***table_name***;**

Example

mysql >show index from employee ;

Types of Index

Indexes can be classified as follows

(1) Simple index
(2) Composite index
(3) Non-unique index
(4) Unique index
(5) Full text index
(6) Spatial Index

(1) Simple Index

When the index is created on single field of the table, it is called "simple index".

Syntax

create index *index_name* **on** *table_name* (*column_name*) **;**

where

create, index, on	-	keywords
index_name	-	name of the index
table_name	-	name of the table
column_name	-	column name on which index is created

Example
mysql >create index dept_index on employee (deptno) ;

(2) Composite Index

When the index is created on multiple fields of the table, it is called "composite index".

Syntax

create index *index_name* **on** *table_name* (*column_names*) **;**

where

create, index, on	-	keywords
index_name	-	name of the index
table_name	-	name of the table
column_names	-	column names on which index is created

Example
mysql >create index emp_index on employee (empno, name) ;

(3) Non-uniqueIndex

Non-unique indexes are created when "unique" keyword is not used in the "create index" command. Duplicate values are allowed on the index column.

Syntax

create index*index_name***on***table_name* (*column_list*) **;**

where

create, index, on	- keywords
index_name	-name of the index
table_name	-name of the table
column_list	-column name(s) on which index is created

Example
mysql >create index dept_index on employee (deptno) ;

When non-unique indexes are created, the "key" value for that column in the "describe" command will display "MUL", meaning "multiple" values are allowed.

mysql> desc employee;

Field	Type	Null	Key	Default	Extra
empno	int(11)	No	Pri	Null	
name	varchar(40)	Yes		Null	
deptno	int(11)	Yes	Mul	Null	

salary	int(11)	Yes		Null	

(4) Unique Index

Unique indexes are created when “unique” keyword is used in the “create index” command. Duplicate values are not allowed on the index column.

<u>Syntax</u>

create unique index*index_name***on***table_name* (*column_list*) **;**

where

create, unique, index, on	- keywords
index_name	-name of the index
table_name	-name of the table
column_list	-column name(s) on which index is created

<u>Example</u>

```
mysql >create unique index name_index on employee (name) ;
```

When unique indexes are created, the “key” value for that column in the “describe” command will display “UNI”, meaning “unique” index.

```
mysql> desc employee;
```

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
empno	int(11)	No	Pri	Null	
name	varchar(40)	Yes	Uni	Null	
deptno	int(11)	Yes	Mul	Null	
salary	int(11)	Yes		Null	

(5) Full text Index

“Full text” indexes are created when “fulltext” keyword is used in the “create index” command. Full text indexes are useful for full text searches. Full text indexes can be created on CHAR, VARCHAR and TEXT column types only.

<u>Syntax</u>

create fulltext index*index_name***on***table_name* (*column_list*) **;**

where

create, fulltext, index, on	- keywords
index_name	-name of the index
table_name	-name of the table
column_list	-column name(s) on which index is created

Example
mysql >create fulltext indexname_index on employee (name) ;

(6) Spatial Index

“Spatial” indexes are created when “spatial” keyword is used in the “create index” command. The use of spatial index is to search for spatial objects, that is, objects of higher order dimension like 2D, 3D, etc.Columns in spatial indexes must be declared NOT NULL.

Syntax

create spatial index*index_name***on***table_name* (*column_list*) **;**

where

create, spatial, index, on	- keywords
index_name	-name of the index
table_name	-name of the table
column_list	-column name(s) on which index is created

Example
mysql >create table geo_table (geo_columngeometrynotnull);
mysql >create spatial index geo_index on geo_table (geo_column) ;

Leftmost Indexing

MySQL can create composite indexes, that is, indexes on multiple columns. An index may consist of up to 16 columns.MySQL can use multiple-column indexes for queries that test all the columns in the index, or just the first column, or the first two columns, or the first three columns, and so on. That is, the indexing will be tested from the leftmost column and this said to be “leftmost indexing”.

Illustration
Consider the queries

(1) select * fromtable_namewhere column1=value1;
(2) select * fromtable_namewhere column1=value1 and column2=value2;
(3) select * fromtable_namewhere column2=value2;
(4) select * fromtable_namewhere column2=value2 and column3=value3;

If an index exists on (col1, col2, col3), only the first two queries use the index. The third and fourth queries do not use the index, because (col2) and (col2, col3) are not leftmost prefixes of (col1, col2, col3).

Example
Suppose that a table has the following specification:

```
create table test (
        id int,
        last_name  varchar(30),
        first_name varchar(30),
        primary key (id),
        index name_index (first_name, last_name)
    );
```

Here, the index “name_index” is an index over the first_name and last_name columns.

Consider the queries

(1) select * fromtest wherefirst_name='Widenius';
(2) select * fromtestwherefirst_name='Widenius' andlast_name='Michael';
(3) select * fromtest wherelast_name='Michael';

Here, the first two queries will use the index. The third query will not use the index because “last_name” is not the leftmost prefix in the index.

Dropping Index

The index can be dropped using “drop index” command and “alter table” command.

Dropping Index with “drop index” command

drop index*index_name***on***table_name***;**

where

drop, index, on -	keywords
index_name -	name of the index
table_name -	name of the table

Example

```
mysql >drop index dept_index on employee ;
```

Dropping Index with“alter table” command

alter table*table_name***drop index***index_name***;**

where

alter, table, drop, index	- keywords
index_name	-name of the index
table_name	-name of the table

Example

```
mysql >altertable employee dropindex dept_index ;
```

- SEQUENCES

Sequence - Definition

Sequence is a list of integers generated in ascending order. Sequences are generally used in generating unique values for primary key column.

Creating Sequence

Sequence is created in MySQL by setting "auto_increment" attribute to a column, which is typically primary key column. Each table can have only one "auto_increment" column and it should be indexed, that is, it can be either primary key or "unique" index.

Creating Sequence with "create table" command

Sequence is created generally while creating table using "create table" command.

Syntax

```
create table table_name (
        seq_column_name   data_type   auto_incrementprimary
key,
        other_column_list ) ;
```

where

create, table, auto_increment, primary key - keywords

table_name	-	name of the table
seq_column_name	-	column name on which sequence is created
other_column_list	-	other column name(s)

Example

```
mysql >create table employee(
        empnointauto_increment primary key,
        name varchar(50)) ;
```

Creating Sequence with "alter table" command

Sequence can be created using "alter table" command. The column that is set as "auto_increment" should have "integer" data type and existing values should be unique. The sequence will continue from the number next to the largest existing value.

Syntax

```
alter tabletable_namemodifyseq_column_name
data_typeauto_increment
        unique | primary key;
```

where

alter, table, modify, auto_increment, unique, primary key - keywords

table_name	-	name of the table
seq_column_name	-	column name on which sequence is created

Example

```
mysql >alter table employee modify id intauto_increment unique;
```

Identifying Sequence in Describe command

The “describe” command will display the sequence column as “auto_increment” against the “Extra” field.

```
mysql> desc employee;
```

Field	Type	Null	Key	Default	Extra
empno	int(11)	No	Pri	Null	Auto_increment
name	varchar(40)	Yes		Null	

(1) Update operation

Now, the value is updated using “update” statement.

```
mysql >update employeesetempno = 50 whereempno = 13;
mysql >select * from employee;
```

empno	name
1	arun
2	babu
10	chitra
11	daniel
50	francis

Now, a row is inserted.

```
mysql >insertinto employee (name) values (“gopal”) ;
mysql >select * from employee;
```

empno	name
1	arun
2	babu
10	chitra
11	daniel
50	francis
14	gopal

Here, the sequence does not continue from updated number, instead, continue from the previous sequence after 13.

Altering Sequence

Setting Sequence starting value

MySQL will start sequence from 1 by default, but it can be set to any other number at the time of table creation.

Syntax

```
create table table_name (
        seq_column_name data_type auto_increment primary key,
        other_column_list ) auto_increment = value ;
```

where

create, table, auto_increment, primary key	-	keywords
seq_column_name	-	column name on which sequence is created
other_column_list	-	other column name(s)
value	-	starting value

Example

```
mysql >createtable employee(
        empnoint auto_increment primary key,
        name  varchar(50))auto_increment = 100 ;
```

In the above example, MySQL will start the sequence from 100.

Resetting Sequence

Sequence can be reset using "alter table" command using the following syntax.

```
alter tabletable_nameauto_increment = value
;
```

where

alter, table, auto_increment	- keywords
table_name	-name of the table

Example

```
mysql >altertable employeeauto_increment = 200 ;
```

In the above example, MySQL will resetthe sequence from 200.

Obtain Sequence value

The last generated sequence number can be obtained using the function "last_insert_id()".

Example

```
mysql>selectlast_insert_id();
```

Last_insert_id( )

5

Deleting Sequence

The sequence is deleted using the “alter table” command by re-defining the sequence column without “auto_increment” attribute.

Syntax

altertable *table_name***modify***seq_column_name data_type***;**

where

alter, table, modify	-	keywords
table_name	-	name of the table
seq_column_name	-	column name on which sequence is deleted

Example

```
mysql >altertable employee modifyempno int ;
```

– PERFORMING MULTIPLE TABLE RETREVAL USING JOINS & UNIONS

JOINS

Definition

In database, Join is used to retrieve records from two or more logically related tables based on common set of values.These values are usually the same column name and datatype that appear in both the participating tablesin join. This common column is called “join key”.

Syntax

select*column1, column2, . . . , columnn***from***table1* **join_type** *table2*[**on***condition*] ;

where

select, from, join_type, on	-	keywords
column1, column2,...	-	column names retrieved
table1, table2	-	name of the tables
condition	-	join condition

Example

```
select empno, empname,deptname from employeeinner join department on
employee.deptno = department.deptno;
```

Aliasing

In SQL, aliases are used to temporarily rename a table or column heading in a query. In Join, alias names are used for tables to shorten the lengthier names and for readability purpose. Alias names are used in column name for better display of headings. Aliases are written just after the column name or table name with or without double quotes. If the alias name comprise of more than one word, double quotes is compulsory.

Syntax

select*column_name alias_name, column_name alias_name***from***table1 alias_name***join_type***table2 alias_name*[**on***condition*] ;

Example

mysql >select empno "Employee Number", empname "Employee Name",deptname from employeeemp inner join departmentdept on emp.deptno = dept.deptno;

Employee Number	Employee Name	deptname
11	arun	civil
22	babu	mechanical

Types of Join

There are different types of Join available in MySql. They are

1. Inner join
2. Natural join
3. Left join
4. Right join
5. Self Join

(1) Inner Join

Inner Join selects rows from both participating tables to appear in the result, if and only if, both tables meet the conditions specified in the "on" clause. Cross join and Inner join are syntactic equivalents. Inner join is normally used with "on" condition.

Syntax

select*column1, column2*, **… from***table1***inner join***table2* [**on***condition*];

where

select, from, inner join, on	-	keywords
column1, column2,…	-	column names retrieved
table1, table2	-	name of the tables
condition	-	join condition

Illustration

Consider the following two tables

Table "emp"

empno	empname	deptno
11	arun	10
22	babu	20
33	chitra	NULL

Table “dept”

deptno	deptname
10	civil
20	mechanical
30	electrical

Inner Join without condition

mysql >select * from emp inner join dept;

empno	empname	deptno	deptno	deptname
11	arun	10	10	civil
22	babu	20	10	civil
33	chitra	NULL	10	civil
11	arun	10	20	mechanical
22	babu	20	20	mechanical
33	chitra	NULL	20	mechanical
11	arun	10	30	electrical
22	babu	20	30	electrical
33	chitra	NULL	30	electrical

Inner Join with condition

mysql >select * from emp inner join depton emp.deptno = dept.deptno;

empno	empname	deptno	deptno	deptname
11	arun	10	10	civil
22	babu	20	20	mechanical

(2) Natural Join

Natural Join is a type of inner join that assumes join criteria to be columns of same name in both the tables. The “on” clause should not be used in Natural join.

Syntax

select*column1, column2*, **… from***table1***natural join***table2* ;

where

select, from, natural join	-	keywords
column1, column2,…	-	column names retrieved
table1, table2	-	name of the tables

Illustration

Consider the following two tables

Table “emp”

empno	empname	deptno
11	arun	10
22	babu	20
33	chitra	NULL

Table “dept”

deptno	deptname
10	civil
20	mechanical
30	electrical

Natural Join

mysql >select * from emp natural join dept;

deptno	empno	empname	deptname
10	11	arun	civil
20	22	babu	mechanical

(3) Left Join

Left Join joins two tables and fetches rows from both the tables based on a condition matching in both the tables and the unmatched rows will also be fetched from the left table with “NULL” values for the right table.

Syntax

select*column1, column2*, **… from***table1***left join***table2* [**on***condition*];

where

select, from, left join	-	keywords
column1, column2,…	-	column names retrieved
table1, table2	-	name of the tables
condition	-	join condition

Illustration

Consider the following two tables

Table “emp”

empno	empname	deptno
11	arun	10
22	babu	20
33	chitra	NULL

Table “dept”

deptno	deptname
10	civil
20	mechanical
30	electrical

Left Join

mysql >select * from emp left join dept on emp.deptno = dept.deptno;

empno	empname	deptno	deptno	deptname
11	arun	10	10	civil

22	babu	20	20	mechanical
33	chitra	NULL	NULL	NULL

(4) Right Join

Right Join joins two tables and fetches rows from both the tables based on a condition matching in both the tables and the unmatched rows will also be fetched from the right table with "NULL" values for the left table.

Syntax
select*column1, column2*, **… from***table1***right join***table2* [**on***condition*];
where

select, from, right join	-	keywords
column1, column2,…	-	column names retrieved
table1, table2	-	name of the tables
condition	-	join condition

Illustration
Consider the following two tables
Table "emp"

empno	**empname**	**deptno**
11	arun	10
22	babu	20
33	chitra	NULL

Table "dept"

deptno	**deptname**
10	civil
20	mechanical
30	electrical

Right Join
mysql >select * from emp right join dept on emp.deptno = dept.deptno;

empno	**empname**	**deptno**	**deptno**	**deptname**
11	arun	10	10	civil
22	babu	20	20	mechanical
NULL	NULL	NULL	30	electrical

(5) Self Join

SelfJoin is a special type of join where a table is joined to itself. To perform the self join, we perform "inner join"of the same table and alias name is used to distinguish the left table from the right table.
Syntax
select*column1, column2*, **… from***table1* alias1 **innerjoin***table1*alias2 [**on***condition*];

where

select, from, inner join	-	keywords
column1, column2,…	-	column names retrieved
table1	-	name of the table
alias1, alias2	-	alias names of the table
condition	-	join condition

Illustration
Consider the following table
Table “emp”

empno	empname	deptno	hodno
11	arun	10	NULL
22	babu	20	NULL
33	chitra	10	11
44	daniel	20	22

To know, hod name, we can perform self join.

Self Join

```
mysql >select e1.empno,e1.empname,e1.hodno,e2.empname hodname from emp e1
        inner join emp e2 where e1.hodno=e2.empno;
```

empno	empname	hodno	hodname
33	chitra	11	arun
44	daniel	22	babu

UNION

Definition

Union operator is used to combine the result sets of two or more “select” statements into a single result set. The number of columns and data type of the columns must be the same in all the “select” statements. The column names from the first select statement is used as the column names for the result.

Syntax

```
select column1 , column2 , ... from table1
union
select column1 , column2 , … from table2 ;
```

Types of Union

Union operator can be used in two forms as

(1) Union (or) Union Distinct

(2) Union All

(1) Union (or) Union Distinct
"Union Distinct" will display only distinct records in the result set removing the duplicate rows. This is the default behavior of "Union".

Example
Consider two tables
Table "faculty"

fname	**faddress**
arun	gandhi street
babu	nehru street
chitra	ambedkar street

Table "student"

sname	**saddress**
daniel	voc street
elango	kannagi street
chitra	ambedkar street

```
mysql >select fname, faddress from faculty union distinct
                    select sname, saddress from student ;
```

fname	**faddress**
arun	gandhi street
babu	nehru street
chitra	ambedkar street
daniel	voc street
elango	kannagi street

(2) Union All
"Union All " will display all the records in the result set without removing the duplicate rows.

Example
```
mysql >select fname, faddress from facultyunion all
                    select sname, saddress from student ;
```

fname	**faddress**
arun	gandhi street
babu	nehru street
chitra	ambedkar street

daniel	voc street
elango	kannagi street
chitra	ambedkar street

Order by

While using union, “order by” clause can be used to order the result set by referencing the column name by its position in the query.

```
select column1 , column2 , ... from table1 union
          select column1 , column2 , … from table2 order by <position>;
```

Example

```
mysql >select fname, faddress from facultyunion
                    select sname, saddress from student order by 2;
```

fname	faddress
chitra	ambedkar street
arun	gandhi street
elango	kannagi street
babu	nehru street
daniel	voc street

Here, the result set is ordered by column number 2, that is, by “faddress”.

Limit Handling

Limit handling is used to limit the number of records displayed in the select query. It can be used in normal select query and also in union.

Example

Consider two tables

Table “faculty”

fname	faddress
arun	gandhi street
babu	nehru street
chitra	ambedkar street

Table “student”

sname	saddress
daniel	voc street
elango	kannagi street
chitra	ambedkar street

```
mysql >select fname, faddress from facultyunion
                    select sname, saddress from student limit 4 ;
```

fname	faddress
arun	gandhi street
babu	nehru street
chitra	ambedkar street
daniel	voc street

```
mysql > (select fname, faddress from faculty limit 2)union
                    (select sname, saddress from student limit 2) ;
```

fname	faddress
arun	gandhi street
babu	nehru street
daniel	voc street
elango	kannagi street

– VIEWS View –

Definition

View is a database object created from tables by using query to view selected portion of the table. Views are called virtual tables. Views do not contain any data of its own but derive the data from the underlying tables called base tables. If data is changed in the base table, the same is reflected in the view. A view can be built from a single or multiple tables. Views can be queried similar to tables.

Advantages of View

(1) View allows us to simplify complex queries.
(2) View can havecalculated columns.
(3) View occupies less storage compared to other database objects because view holds no data until it is called.
(4) View provides extra security layer in DBMS by restricting the view of the table.

Dis-advantages of View

(1) Querying data from view can be slow, because, the data has to be retrieved after identifying the base tables.
(2) When the structure of the table is changed, the view also need be changed.
(3) When the table is dropped, view becomes irrelevant.

Creating View

View can be created using “create view” command.

Syntax

```
create [ or replace ] viewview_nameas
            select columnsfromtable_name(s) [ wherecondition
] ;
```

where

create, view, as, select, from, where	-	keywords
or replace	-	optional. If specified, it will replace the view, if it exists already.
view_name	-	name of the View
columns	-	columns selected from the table, expression
table_name(s)	-	name of the base table(s)
where condition	-	optional.

Example

Consider a table "employee" with columns (empno, name, dept, salary, branch)

empno	**name**	**dept**	**salary**	**branch**
11	arun	hr	10000	chennai
22	babu	sales	12000	madurai
33	chitra	hr	14000	trichy
44	daniel	accounts	16000	chennai

```
mysql >create view hr_view as
        selectempno, name, dept, salary, branchfrom employee where dept =
"hr" ;
```

This will create a view called "hr_view" which will select employees in "hr" department.

Querying View

A View is queried similar to a table using "select" statement.

Syntax

```
select column(s)from view_name ;
```

where

select, from	-	keywords
view_name	-	name of the view
column(s)	-	list of column names

Example

```
mysql >select * from hr_view ;
```

empno	**name**	**dept**	**salary**	**branch**

11	arun	hr	10000	chennai
33	chitra	hr	14000	trichy

View with calculated column

The view can be created with calculated column having expression. The column can be renamed in the view by using “as” keyword in the “select” statement.

Example

```
mysql >createview annual_salary_view as
        selectempno, name, salary, salary * 12 as annual from employee ;

select * from annual_salary_view ;
```

empno	**name**	**salary**	**annual**
11	arun	10000	120000
22	babu	12000	144000
33	chitra	14000	168000
44	daniel	16000	192000

Displaying View name

The views created in the database can be viewed using “show tables” or “show full tables” command. The “show tables” command will display the tables and views in the database. The “show full tables” command will display the “table_type” too.

```
mysql > show tables ;
```

Tables_in_schema
employee
hr_view

```
mysql > show full tables ;
```

Tables_in_schema	**Table_type**
employee	Base table
hr_view	View

Alter View

“Alter view” command is used to alter the definition of existing view.

Syntax

alter view*view_name***as select***column(s)***from***table_name(s)* [**where** *condition*] **;**

where

alter, view, as, select, from, where - keywords

view_name	-name of the view
column(s)	-list of column names
table_name(s)	-name of the base tables

Example

```
mysql >alter view hr_view as
        selectempno, name from employeewhere dept = "hr" ;
```

The view "hr_view" is altered by selecting only two columns.

```
mysql >select * from hr_view ;
```

empno	name
11	arun
33	chitra

Updating View

Updatable views are views on which data modifications can be done. Since views are imaginary tables, all the modifications performed on the views will be effected on the base table. All views are not updatable.

Rules for updating view

- Views created on a single table are updatable if the view contains primary key fieldas well as all the "not null" fields of the base table.
- Views defined on multiple tables using joins or sub-query are generally not updatable.
- Views defined using grouping and aggregate functions are not updatable.

The updatable views can be inserted, deleted and updated.

The syntax for update statement in View is similar to updating table as given below

update *view_name***set***column* = *value* [**where***condition*] ;

Example

Consider a table "employee" with columns (empno, name, dept);

empno	name	dept
11	arun	hr
22	babu	sales
33	chitra	hr
44	daniel	accounts

Creating View

```
mysql >create view hr_view as
        select empno, name  from employee where dept = "hr" ;
```

This will create a view called “hr_view” which will select employees in “hr” department.
mysql >select * from hr_view ;

empno	name
11	arun
22	chitra

Updating View
mysql >update hr_view set name = “arunkumar” where empno = 11 ;
mysql > select * from hr_view ;

empno	name
11	arunkumar
22	chitra

The data gets updated in the base table “employee” as given below.
mysql >select * from employee ;

empno	name	dept
11	arunkumar	hr
22	babu	sales
33	chitra	hr
44	daniel	accounts

Dropping View

The view can be dropped using “drop view” command. Dropping views will not affect the base tables in any way.

Syntax

drop view*view_name***;**

where

drop, view	-	keywords
view_name	-	name of the view

Example
mysql >drop view hr_view ;

– USER & TRANSACTION MANAGEMENT

3.4.1.USER MANAGEMENT

Creating User

The user account created after the installation of MySQL is “root” only. It is the administrator user with all privileges. For security reasons, it is not recommended to use the “root” user account by everyone. Hence, various user accounts are created for different users with relevant privileges depending upon their project.

Users are created in MySQL using “create user” command. An account in MySQL consists of username and hostname, separated by “@” symbol. For example, if the root user connects from the mysql.org host to the database server, the account name would be root@mysql.org.This allows us to setup multiple accounts with the same name but connects from different hosts with different privileges.The username and hostname are stored in the table “mysql.user”.

Syntax

create user “*username*”@”*hostname*” **identified by** “*password*” ;

where

create, user, identified, by	-	keywords
username	-	name of the user
hostname	-	name of the host
password	-	password

Example

```
mysql >create user “user1”@”localhost” identified by “password1” ;
```

This will create user called “user1” with hostname “localhost” and password “password1”;

If “hostname” is omitted in the “create user” command, the hostname is taken as “%”, and the user can be connected from any host.

Connecting User

The command to connect to user from dos prompt is

```
C:\>mysql –u username -p
```

Example

```
C:\>mysql –u user1 -p
Enter password : *********
mysql >
```

Display current user

The command to display the current logged in user is

select user ( ) ;

Example

mysql > select user( ) ;

user( )
user1@localhost

Exit User

The command to exit from current user is “exit” or “quit”.

mysql.user table

All the users created will have an entry in the “user” table at “mysql” database. The following query can be used to display the username and hostname from the table.

mysql > select user, host from mysql.user ;

user	**host**
root	localhost
user1	localhost

Rename User

A user in the MySQL database can be renamed using “rename user” command and by updating the “mysql.user” table.

The syntax for “rename user” command is

rename user “*old_user_name*” **to** “*new_user_name*” **;**

Example

mysql >rename user “user1” to “user2” ;

The query to rename user by updating the “mysql.user” table is

update*mysql.user***setuser** = “*new_user_name*” **where user=** “*old_user_name*” ;

Example

mysql >update mysql.user set user = “user2” whereuser = “user1” ;

Drop User

“drop user” command is used to drop user.

Syntax

```
drop user "username"@"hostname" ;
```

Example

```
mysql >dropuser "user1"@"localhost" ;
```

Grant Privileges

Privileges are the permissions given to the user to access the database and perform particular action on database objects."grant" command is used to provide privileges to the users.

Syntax

```
grant privilegeson database_name . object_type
tousername@hostname
              [ with grant option ] ;
```

where

privileges	Type of privilege, example, create, drop, select, insert, etc. To grant all privileges, use "all"
database_name	Database on which privileges are granted. Use "*" symbol to denote all databases.
object_type	Name of the database objects like table, view, etc. Use "*" symbol to denote all database objects.
with grant option	This is optional. If this option is used, the user can further grant privileges to other user.

Example

```
mysql >grant select on db_emp.employeeto user1@localhost ;
```

This will grant "select" privilege on the table "employee" on the database "db_emp" to the user "user1" from the host "localhost". Now, "user1" can select the table "employee".

View granted privileges

The privileges granted for a user can be viewed using the following command

```
>show grants forusername ;
```

Example

```
mysql >show grants for user1 ;
```

Grants for user1@localhost
Grant usage on *.* to user1@localhost Grant select on db_emp.employee to user1@localhost

```
Grant all privileges on mysql . * to user1@localhost
```

Working of "grant" command

Login to "root"

```
        c:\>    mysql –u root –p
                Enter password : *********

        mysql > grant select on db_emp.employee to user1@localhost ;
```

Login to "user1"

```
        c:\>    mysql –u user1 –p
                Enter password : *********
```

Select the table "employee" from "user1"

```
        mysql > select * from db_emp.employee ;
```

empno	**name**	**salary**
11	arun	10000
22	babu	12000

This will succeed.

Now, insert one row into "employee" table

```
        mysql > insert into db_emp.employee values (33, "chitra", 14000) ;
                Error : INSERT command denied to user "user1"@"localhost" for
                table "employee"
```

This will issue error, since "insert" privilege is not granted.

Login to "root" user and grant "insert" privilege

```
        mysql > exit
        c:\>    mysql –u root –p
                Enter password : *********

        mysql > grant insert on db_emp.employee to user1@localhost ;
```

Login to "user1" user and try "insert" command

```
        mysql > exit
        c:\>    mysql –u user1 –p
                Enter password : *********
        mysql > insert into db_emp.employee values (33, "chitra", 14000) ;
        mysql > select * from db_emp.employee ;
```

empno	**name**	**salary**
11	arun	10000
22	babu	12000

33	chitra	14000

Now, the “insert” statement will succeed, since the privilege is granted.

Sample “grant” statements

Command to grant all privileges on “db_emp” database on “employee” table to “user1”
mysql >grant all on db_emp.employee to user1@localhost ;

Command to grant all privileges on “db_emp” database on all tables to “user2”
mysql >grant all on db_emp**.*** to user2@localhost ;

Command to grant all privileges on all databases on all tables to “user3”
mysql >grant all on ***.*** to user3@localhost ;

Revoke Privileges

“revoke” command is used to withdraw the granted permissions on database objects from the users when needed.

Syntax

revoke*privileges***on***database_name***.**
*object_type***from***username@hostname***;**

where

privileges	Type of privilege, example, create, drop, select, insert, etc. To grant all privileges, use “all”
database_name	Database on which privileges are granted. Use “*” symbol to denote all databases.
object_type	Name of the database objects like table, view, etc. Use “*” symbol to denote all database objects.

Example
mysql >revoke select on db_emp.employee from user1@localhost ;

Working of “Revoke” command

Login to “root” user and revoke “insert” privilege

```
mysql > exit
c:\>    mysql –u root –p
        Enter password : *********
        mysql > revoke insert on db_emp.employee from user1@localhost ;
```

<u>Login to “user1” user and try “insert” command</u>

```
        mysql > exit
        c:\>    mysql –u user1 –p
                Enter password : *********

        mysql > insert into db_emp.employee values (44, “daniel”, 16000) ;
                  Error : INSERT command denied to user “user1”@”localhost” for
                  table “employee”
```

“insert” command will not work, since it is revoked. But “select” will work.

```
        mysql > select * from db_emp.employee ;
```

empno	**name**	**salary**
11	arun	10000
22	babu	12000
33	chitra	14000

<u>Login to “root” user and revoke “select” privilege</u>

```
        mysql > exit
        c:\>    mysql –u root –p
                Enter password : *********

        mysql > revoke select on db_emp.employee from user1@localhost ;
```

<u>Login to “user1” user and try “select” command</u>

```
        mysql > exit
        c:\>    mysql –u user1 –p
                Enter password : *********

                mysql > select * from db_emp.employee ;
                  Error : SELECT command denied to user “user1”@”localhost” for
                  table “employee”
```

“select” command will not work, since it is also revoked.

<u>Sample “revoke” statements</u>

<u>Command to revoke all privileges on “db_emp” database on “employee” table from “user1”</u>

```
mysql >revoke all on db_emp.employeefrom user1@localhost ;
```

<u>Command to revoke all privileges on “db_emp” database on all tables from “user2”</u>

```
mysql >revoke all on db_emp.*from user2@localhost ;
```

<u>Command to revoke all privileges on all databases on all tables from “user3”</u>
mysql >revoke all on *.*from user3@localhost ;

– TRANSACTION MANAGEMENT

Transaction

A transaction is a sequential group of database operations, executed as if it were one single work unit, which changes the content of the database. The changes made to the database are committed when all the database operations in the transaction succeed. If any operation within the transaction fails, the entire transaction is rolled back.

<u>Example</u>

Fund transfer in banking application. If a debit is made successfully from one account, the corresponding credit is made to the other account.

Transaction States

A transaction goes through different states during its execution. They are

(1) Active
(2) Partially committed
(3) Committed
(4) Failed
(5) Aborted

<u>(1) Active</u>
This is the initial state and the transaction stays in this state while it is running.

<u>(2) Partially committed</u>
Transaction goes to this state when the final statement in the transaction unit has been executed.

<u>(3) Committed</u>
A transaction that completes its execution successfully and committed the transaction is said to be in “committed” state.

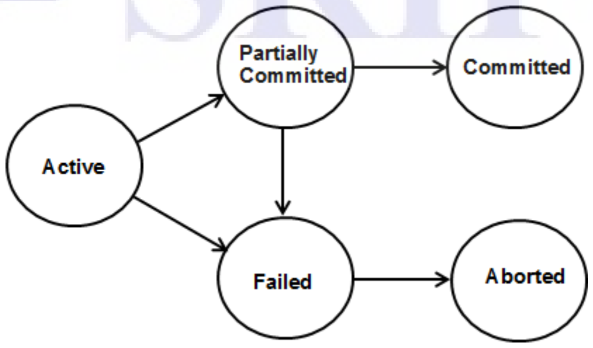


Fig. 3.1 - Transaction States

(4) Failed

The transaction goes to this state when normal execution of the transaction can no longer be proceeded.

(5) Aborted

The transaction goes to this state, after the transaction has been rolled back in failed condition.

Implementing Transaction

The following commands are used in implementing transaction in MySQL.

(1) start transaction (or) begin
(2) commit
(3) rollback
(4) savepoint

(1) start transaction (or) begin

This command is used to start a new transaction. The transaction gets completed when the command "commit" or "rollback" without savepoint is executed. Any DDL (Data Definition Language) command will also implicitly end the transaction by committing the changes.

(2) commit

The full syntax of the command is

commit [and [no] chain] [[no] release]

This command is used to save the changes to the database permanently and ends the transaction. When the command is used with "and chain" attribute, a new transaction will begin as soon as the current transaction ends.The attribute "release"will disconnect the current session from the database server, after committing and terminating the transaction.

Example

```
mysql > commit ;
```

(3) rollback

The full syntax of the command is

rollback [and [no] chain] [[no] release] [to savepoint *identifier* **]**

This command is used to undo the changes to the database and ends the transaction. When the command is used with "and chain" attribute, a new transaction will begin as soon as the current transaction ends. The attribute "release" will disconnect the current session from the database server, after rollback and terminating the transaction. The command "rollback to savepoint" is used to undo the transaction upto the savepoint given by the identifier.

Example

```
mysql > rollback ;
```

(4) savepoint

The commands used in savepoint are

- **savepoint***identifier*
- **rollback to savepoint***identifier*
- **release savepoint***identifier*

The “savepoint” command is used to set a point in the transaction using which we can later rollback. If the current transaction has a savepoint with the same name, the old savepoint is deleted and the new one is set.
The “rollback to savepoint” statement rolls back the transaction upto the named savepoint without terminating the transaction. Database modifications made after the named savepoint by the current transaction are undone in the rollback.

The “release savepoint” statement deletes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist.All savepoints of the current transaction are deleted if we execute “commit” or “rollback”without savepoint.

<u>Example</u>

```
mysql > savepoint s1 ;
mysql > rollback to savepoint s1 ;
mysql > release savepoint s1 ;
```

Working of Transaction commands

Consider the table “employee”

empno	**name**
11	arun
22	babu

```
mysql > start transaction ;
mysql > savepoint s1 ;
mysql > insert into employee values (33, ”chitra”) ;
mysql > select * from employee ;
```

empno	**name**
11	arun
22	babu
33	chitra

```
mysql > savepoint s2 ;
mysql > insert into employee values (44, ”daniel”) ;
```

mysql > select * from employee ;

empno	**name**
11	arun
22	babu
33	chitra
44	daniel

mysql > rollback to savepoint s2 ;

mysql > select * from employee ;

empno	**name**
11	arun
22	babu
33	chitra

mysql > release savepoint s2 ;

mysql >commit ;

mysql > select * from employee ;

empno	**name**
11	arun
22	babu
33	chitra

mysql > rollback to savepoint s1 ;

Error : savepoint s1 does not exist.

Note : Once the commit command is given, all the savepoints defined before are deleted.

Summary

- A database index is a data structure that improves the speed of operations in a table.
- Sequence is a list of integers generated in ascending order.
- Join is used to retrieve records from two or more logically related tables based on common set of values.
- Union is used to combine the result sets of two or more “select” statements into a single result set.
- View is created from tables by using query to view selected portion of the table.
- Users are created in MySQL using “create user” command.
- “grant” command is used to provide privileges to the users.
- "revoke” command is used to withdraw the granted permissions.
- "commit" command is used to save the changes to the database permanently.
- "rollback" command is used to undo the changes to the database.

Review Questions

PART A

1. Define index.
2. What is full text indexing?
3. How do you drop index?
4. Define sequence.
5. How do you delete sequence?
6. Define Join.
7. What is aliasing?
8. List the different types of Join.
9. Define Union.
10. Define View.
11. How do you delete View?
12. What is the use of grant and revoke command?
13. What is the use of commit and rollback commands?
14. What is savepoint?

PART B

1. How will you create an index using Primary key
2. How will you create an index using Foreign key
3. Explain full text indexing
4. Explain about limit handling
5. What are the advantages of View?
6. Explain grant command
7. Explain revoke command
8. Explain inner join
9. Explain commit statement
10. Explain self-join

PART C

1. What is index? How do you create index?
2. Discuss leftmost indexing.
3. How do you create sequence?
4. How do you alter sequence?
5. What is Join? How do you create Join?
6. What is Union? How do you create Union?
7. Discuss the types of Union with example.
8. Discuss limit handling in Union.
9. Explain various types of Join with sample queries.
10. What is View? How do you create View?
11. How do you update View?
12. How do you create and delete user?
13. Discuss grant and revoke commands?
14. Explain transaction handling using commit and rollback.

UNIT- IV
Storage Engines, Stored Program concept , Optimization & API's

OBJECTIVES

- To know about Mysql Storage engines – MyISAM, InnoDB, Memory types.
- To create, invoke, drop the Mysql Stored Procedure and function.
- To get information about the various types of Mysql triggers and how to create, use & delete them .
- To describe about cursor with its creation and deletion procedure.
- To optimize Mysql queries with the use Explain command.
- To understand the need of writing own mysql programs and API's.

Storage Engines

Definition :

A storage engine is a software that is used by a database management system to create, read, and update data from a database. Most DBMS use APIs (Application Programming Interface) to enable interactions of users with the storage engines.

MySQL Storage Engines:

MySQL Storage engines are MySQL components that handle the SQL operations for different table types. MySQL storage engines include both those that handle transaction-safe tables and those that handle non transaction-safe tables.

Transaction-safe tables (TSTs) have several advantages over non-transaction-safe tables (NTSTs):

- They are safer even after MySQL crashes or hardware problems by using automatic recovery or from a backup plus the transaction log.
- Many statements can be combined and accepted at the same time with the COMMIT statement (if autocommit is disabled).
- You can execute ROLLBACK to ignore the changes made.
- If an update fails, all the changes made are reverted.

- Transaction-safe storage engines can provide better concurrency for tables that get many updates concurrently with reads.

ii) Non-Transactional safe tables:

In non-transaction safe tables, all changes that take place in the tables are permanent. In order to perform rollback operation, the user will need to do it manually with codes.

Its advantages are:

- Much faster
- Lower disk space requirements
- Less memory required to perform updates

Types of Storage engines:

There are many storage engines available in MySQL and they are used for different purposes.

MySQL supports the following storage engines:

- MyISAM
- InnoDB
- Memory (Heap)
- Merge
- ARCHIVE
- CSV
- FEDERATED

Let us study the first three storage engines.

i) MyISAM storage engine

MyISAM was the default storage engine for the MySQL relational database management system versions prior to 5.5.

Features :

- MyISAM is supported in all MySQL configurations.
- This storage engine, manages non transactional tables,
- This provides high-speed storage and retrieval, supports full text searching.
- MyISAM storage engine is used the most in Web, data warehousing, and other application environments with heavy read operations. This is because of the structure of its Indexes.
- Tables can be compressed.

Advantages of MyISAM :

1. **Simpler to design and create**, thus better for beginners. No worries about the foreign relationships between tables.
2. **Faster than InnoDB on the whole** as a result of the simpler structure thus much less costs of server resources.
3. supports FULLTEXT indexing and OpenGIS data types.
4. Especially good for **read-intensive (select) tables**.

Disadvantages of MyISAM :

1. **No data integrity** (e.g. relationship constraints) check, which then comes a responsibility and overhead of the database administrators and application developers.
2. **Doesn't support transactions** which is essential in critical data applications such as that of banking.

ii) InnoDB storage engine :

This is the default storage engine for MySQL 5.5 and higher. InnoDB is a storage engine for MySQL that balances high reliability and high performance.

Features :

It provides transaction-safe (ACID compliant) tables

Supports FOREIGN KEY referential-integrity constraints.

It supports commit, rollback, and crash-recovery capabilities to protect data.

It also support row-level locking.

It's "consistent nonlocking reads" increases performance when used in a multiuser environment.

It stores data in clustered indexes which reduces I/O for queries based on primary keys.

Advantages of InnoDB storage engine

- InnoDB has maximum performance when processing large data volumes.
- Its DML operations (add, update and delete data) is ACID (atomic, consistent, isolated and durable) model compatible, with transactions featuring commit, rollback, and crash-recovery capabilities to protect user data.

- Row-level locking (locks are placed on single records (rows)) system increase multi-user concurrency and performance.
- InnoDB tables arrange the user data on disk to optimize queries based on primary keys.
- InnoDB supports FOREIGN KEY constraints to maintain data integrity.
- It is possible to mix InnoDB tables with tables from other MySQL storage engines within the same statement using 'join' .

Limitation: InnoDB table :

- Maximum 1017 columns are allowed in a table
- Maximum 64 secondary indexes (subset of table columns)are allowed in a table.
- By default, an index key for a single-column index can be up to 767 bytes.
- The InnoDB internal maximum key length is 3500 bytes.
- The maximum row length except for variable-length columns (VARBINARY, VARCHAR, BLOB and TEXT), is about 8000 bytes for the default pae size of 16KB.
- The maximum table space size is 64TB database pages (64TB) and the minimum table space size nearly 10MB.

iii) MEMORY storage engine

Provides in-memory tables, formerly known as HEAP. The MEMORY storage engine creates tables that are stored in memory. It stores all data in RAM for faster access than storing data on disks. Useful for quick looks up of reference and other identical data. This is because that Memory tables support HASH indexes, which are very fast for lookup queries. Memory tables can be up to an order of magnitude faster than MyISAM tables. Because the data can be crashed due to hardware or power issues, these tables are used only as temporary work areas or read-only caches for data pulled from other tables. When the MySQL server halts or restarts, the data in MEMORY tables is lost.

Creating Memory storage engine:

```
CREATE TABLE t3 (i INT) ENGINE = MEMORY;
```

MySQL Stored Procedures & Functions

MySQL: Stored Procedures

Definition : In MySQL, a procedure is a stored program which passes parameters into. It does not return a value like a function does.

Create Procedure:

The syntax to create a procedure in MySQL is:

```
CREATE PROCEDURE procedure_name [ (parameter datatype [, parameter
datatype]) ]
BEGIN
  declaration_section
  executable_section
END;
```

Where

procedure_name - The name to assign to this procedure in MySQL.

Parameter - Optional. One or more parameters passed into the procedure. When creating a procedure, there are three types of parameters that can be declared:

1. **IN** - The parameter can be referenced by the procedure. The value of the parameter can not be overwritten by the procedure.
2. **OUT** - The parameter can not be referenced by the procedure, but the value of the parameter can be overwritten by the procedure.
3. **IN OUT** - The parameter can be referenced by the procedure and the value of the parameter can be overwritten by the procedure.

declaration_section - The place in the procedure where you declare local variables.

executable_section - The place in the procedure where you enter the code for the procedure.

Example

Let's look at an example that shows how to create a procedure in MySQL:

```
DELIMITER //
CREATE procedure CalcIncome ( OUT ending_value INT )
BEGIN
  DECLARE income INT;
  SET income = 50;
  label1: WHILE income <= 3000 DO
    SET income = income * 2;
  END WHILE label1;
  SET ending_value = income;
END; //
DELIMITER ;
```

Invoking Procedure :

The Stored Procedure is executed or invoked as follows:

```
CALL Procedure_name (@variable_name);

SELECT @variable_name;
```

4.2.1.3. **Droping procedure**

This is used to remove the stored procedure from the database.

Syntax

The syntax to a drop a procedure in MySQL is:

```
DROP procedure [ IF EXISTS ] procedure_name;
```

Where procedure_name - The name of the procedure that you wish to drop.

Example

To drop the above CalcIncome procedure in MySQL, the following command is used:

```
DROP procedure CalcIncome;
```

MySQL stored function

Stored Functions:

Definition : A stored function is a special kind stored program that returns a single value. It is used to encapsulate common formulas or business rules that are reusable among SQL statements or stored programs.

A stored function is invoked in SQL statements wherever an expression is used. This helps to improve the readability and maintainability of the procedural code.

Creating Stored Function

The syntax for creating a new stored function is:

```
CREATE FUNCTION function_name [ (parameter datatype [, parameter datatype]) ]
RETURNS return_datatype
BEGIN
  declaration_section
  executable_section
END;
```

where function_name : The name to assign to this function in MySQL.
Parameter : One or more parameters passed into the function. When creating a function, all parameters are considered to be **IN parameters** (not OUT or INOUT parameters) where the parameters can be referenced by the function but can not be over written by the function.
return_datatype : The data type of the function's return value.
declaration_section : The place in the function where you declare local variables.
executable_section : The place in the function where you enter the code for the function.

Example

Let's look at an example that shows how to create a function in MySQL:

```
DELIMITER //
CREATE FUNCTION CalcIncome ( starting_value INT )
RETURNS INT
BEGIN
  DECLARE income INT;
  SET income = 0;
  label1: WHILE income <= 3000 DO
    SET income = income + starting_value;
  END WHILE label1;
  RETURN income;
END; //
DELIMITER ;
```

Calling a Stored Function

The following command is to call or reference the above stored function in Mysql :

```
SELECT CalcIncome (1000);
```

Deleting a Stored FunctionDrop

Function

Drop function is used to remove the already created MySQL function from its database.

Syntax

The syntax to a drop a function in MySQL is:

```
DROP FUNCTION [ IF EXISTS ] function_name;
```

Where function_name : The name of the function that is to be dropped.

Example

DROP FUNCTION CalcIncome;

Advantages and disadvantages of using stored procedures

Advantages:

1. Reduce network usage between clients and servers .
2. Improved security – database administrator can control the users who access the stored
   Procedure.
3. Reduced development cost and increased reliability
4. Access to other database objects in a secure and uniform way.
5. Can prevent SQL injection attacks.
6. Encapsulation of business logic – less chances to data become corrupted through faulty client
   programs.

MySQL trigger & Cursor

MySQL trigger

The MySQL trigger is a database object that is associated with a table. It will be activated when a defined action is executed for the table. The trigger can be executed when any one of the following MySQL statements are run on the table:

INSERT, UPDATE and DELETE. It can be invoked before or after the event. Triggers are available in MySQL 5.0.2 and later.

Uses for triggers :

- Enforce business rules
- Validate input data
- Generate a unique value for a newly-inserted row in a different file.
- Write to other files for audit trail purposes
- Query from other files for cross-referencing purposes
- Access system functions
- Replicate data to different files to achieve data consistency

Creating triggers

A trigger is a named database object that is associated with a table, and it activates when a particular event (e.g. an insert, update or delete) occurs for the table. The statement CREATE TRIGGER creates a new trigger in MySQL. Here is the syntax :

Syntax :

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
trigger_body
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```

Explanation :

DEFINER clause : The DEFINER clause specifies the MySQL account to be used when checking access privileges at trigger activation time. If a user value is given, it should be a MySQL account specified as 'user_name'@'host_name' CURRENT_USER, or CURRENT_USER().
The default DEFINER value is the user who executes the CREATE TRIGGER statement.

trigger_name : All triggers must have unique names within a schema. Triggers in different schemas can have the same name.

trigger_time : trigger_time is the trigger action time. It can be BEFORE or AFTER to indicate that the trigger activates before or after each row to be modified.

trigger_event : trigger_event indicates the kind of operation that activates the trigger. These trigger_event values are permitted:

- The trigger activates whenever a new row is inserted into the table; for example, through INSERT, LOAD DATA, and REPLACE statements.
- The trigger activates whenever a row is modified; for example, through UPDATE statements.
- The trigger activates whenever a row is deleted from the table; for example, through DELETE and REPLACE statements.
- **tbl_name :** The trigger becomes associated with the table named tbl_name, which must refer to a permanent table. You cannot associate a trigger with a TEMPORARY table or a view.

trigger_body : trigger_body is the statement to execute when the trigger activates. To execute multiple statements, use the BEGIN ... END compound statement construct.

Types of trigger

A trigger can be defined to be invoked either before or after the data is changed by INSERT, UPDATE or DELETE statement. There are six types of triggers for each table. They are :

- BEFORE INSERT – activated before data is inserted into the table.
- AFTER INSERT – activated after data is inserted into the table.
- BEFORE UPDATE – activated before data in the table is updated.
- AFTER UPDATE – activated after data in the table is updated.
- BEFORE DELETE – activated before data is removed from the table.
- AFTER DELETE – activated after data is removed from the table

Example 1 :

```
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account FOR EACH
ROW SET
        @sum = @sum + NEW.amount;
```

In the above example, there is new keyword '**NEW**' which is a MySQL extension to triggers. There is two MySQL extension to triggers '**OLD**' and '**NEW**'. OLD and NEW are not case sensitive.

- Within the trigger body, the OLD and NEW keywords enable you to access columns in the rows affected by a trigger
- In an INSERT trigger, only NEW.col_name can be used.
- In a UPDATE trigger, you can use OLD.col_name to refer to the columns of a row before it is updated and NEW.col_name to refer to the columns of the row after it is updated.

- In a DELETE trigger, only OLD.col_name can be used; there is no new row.

A column named with OLD is read only. You can refer to it (if you have the SELECT privilege), but not modify it. You can refer to a column named with NEW if you have the SELECT privilege for it. In a BEFORE trigger, you can also change its value with SET NEW.col_name = value if you have the UPDATE privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or used to update a row.

Example 2 :

Here is another example of a MySQL trigger:

- First we will create the table for which the trigger will be set:

mysql> CREATE TABLE people (age INT, name varchar(150));

- Next we will define the trigger. It will be executed before every INSERT statement for the people table:

mysql> delimiter //
mysql> CREATE TRIGGER agecheck BEFORE INSERT ON people FOR EACH ROW IF NEW.age < 0 THEN SET NEW.age = 0; END IF;//

mysql> delimiter ;

- We will insert two records to check the trigger functionality.

mysql> INSERT INTO people VALUES (-20, 'Sid'), (30, 'Josh');
Query OK, 2 rows affected
Records: 2 Duplicates: 0 Warnings: 0

- At the end we will check the result.

*mysql> SELECT * FROM people;*

```
- - - - - - - - - - -
| age | name  |
- - - - - - - - - - -
| 0   |   Sid     |
| 30 |   Josh   |
```

2 rows in set

MySQL SHOW TRIGGERS statement

Another quick way to display triggers in a particular database is to use SHOW TRIGGERS statement as follows:

SHOW TRIGGERS [FROM|IN] database_name

[LIKE expr | WHERE expr];

To view all triggers in the current database:

SHOW TRIGGERS;

To get all triggers in a specific database:

SHOW TRIGGERS FROM classicmodels;

It returns all triggers in the classicmodels database.

Deleting triggers

To destroy the trigger, a DROP TRIGGER statement is used. You must specify the schema name if the trigger is not in the default schema:

mysql> **DROP TRIGGER agecheck;**

If a table is dropped , any triggers for the table are also dropped.

Cursor

Introduction to MySQL cursor

A cursor is used to handle a result set inside a stored procedure . A cursor allows us to iterate a set of rows returned by a query and process each row accordingly.

MySQL supports cursors inside stored programs. The syntax is as in embedded SQL. Cursors have these properties:

- Asensitive: The server may or may not make a copy of its result table
- Read only: Not updatable
- Nonscrollable: Can be traversed only in one direction and cannot skip rows

Cursor declarations must appear before handler declarations and after variable and condition declarations. MySQL cursors are used in stored procedures, stored functions, and triggers.

Creation of Cursors

i) Cursor DECLARE Syntax

DECLARE *cursor_name* CURSOR FOR *select_statement;*

This statement declares a cursor and associates it with a SELECT statement that retrieves the rows to be traversed by the cursor. To fetch the rows later, use a FETCH statement. The number of columns retrieved by the SELECT statement must match the number of output variables specified in the FETCH statement.

Cursor declarations must appear before handler declarations and after variable and condition declarations. A stored program may contain multiple cursor declarations, but each cursor declared in a given block must have a unique name.

ii) Cursor OPEN Syntax

OPEN *cursor_name ;*

This statement opens a previously declared cursor.

iii) Cursor FETCH Syntax

FETCH [[NEXT] FROM] *cursor_name* INTO *var_name* [, *var_name*] ...

This statement fetches the next row for the SELECT statement associated with the specified cursor (which must be open), and advances the cursor pointer. If a row exists, the fetched columns are stored in the named variables. The number of columns retrieved by the SELECT statement must match the number of output variables specified in the FETCH statement.

iv) Close syntax

Finally, we close the cursor using the CLOSE statement:

CLOSE ***cursor_name*** ;

Cursor Example:

```
CREATE PROCEDURE processorders()
BEGIN
  -- Declare local variables
  DECLARE done BOOLEAN DEFAULT 0;
  DECLARE o INT;
  DECLARE t DECIMAL(8,2);
  -- Declare the cursor
  DECLARE ordernumbers CURSOR
  FOR
  SELECT order_num FROM orders;
  -- Declare continue handler
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done=1;
  -- Create a table to store the results
  CREATE TABLE IF NOT EXISTS ordertotals
    (order_num INT, total DECIMAL(8,2));
  -- Open the cursor
  OPEN ordernumbers;
  -- Loop through all rows
  REPEAT
    -- Get order number
    FETCH ordernumbers INTO o;
    -- Get the total for this order
    CALL ordertotal(o, 1, t);
    -- Insert order and total into ordertotals
    INSERT INTO ordertotals(order_num, total)
    VALUES(o, t);
  -- End of loop
  UNTIL done END REPEAT;
  -- Close the cursor
  CLOSE ordernumbers;
END;
```

In this example, we've added another variable named t (this will store the total for each order). The stored procedure also creates a new table on the fly (if it does not exist) named ordertotals. This table will store the results generated by the stored procedure. FETCH fetches each order_num as it did before, and then used CALL to execute another stored procedure (the one we created in the previous tutorial) to calculate the total with tax for each order (the result of which is stored in t). And then finally, INSERT is used to save the order number and total for each order.

This stored procedure returns no data, but it does create and populate another table that can then be viewed using a simple SELECT statement:

```
SELECT * FROM ordertotals;
- - - - - - - - - - - - - - -
| order_num | total  |
- - - - - - - - - - - - - - -
|     20005 |  158.86 |
|     20006 |   58.30  |
|     20007 | 1060.00|
|     20008 |  132.50 |
|     20009 |   40.78  |
- - - - - - - - - - - - - - -
```

Cursor Deletion

The cursor is deleted using the following :

DELETE cursor_name;

MySQL Optimizations

Query optimization using EXPLAIN command.

Optimization : Database performance depends on several factors at the database level, such as tables, queries, and configuration settings. These software constructs result in CPU and I/O operations at the hardware level which must be minimized and made as efficient as possible.

Query Optimization **:** Depending on the details of the MySQL tables, columns, indexes, and the conditions in the WHERE clause, the MySQL optimizer considers many techniques to efficiently perform the lookups involved in an SQL query. Using query optimization, a query on a huge table can be performed without reading all the rows; a join involving several tables can be performed without comparing every combination of rows. The set of operations that the optimizer chooses to perform the most efficient query is called the "**query execution plan**", also known as the **EXPLAIN** plan.

Optimizing Queries with EXPLAIN

The EXPLAIN statement provides information about how MySQL executes statements:

- EXPLAIN works with SELECT, DELETE, INSERT, REPLACE, and UPDATE statements.
- When EXPLAIN is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order
- When EXPLAIN is used with FOR CONNECTION *connection_id* rather than an explainable statement, it displays the execution plan for the statement executing in the named connection.
- For SELECT statements, EXPLAIN produces additional execution plan information.
- EXPLAIN is useful for examining queries involving partitioned tables.
- The FORMAT option can be used to select the output format. TRADITIONAL presents the output in tabular format. This is the default if no FORMAT option is present.

EXPLAIN is used to find where to add indexes to tables so that the statement executes faster by using indexes to find rows. Also it is used to check whether the optimizer joins the tables in an optimal order.

EXPLAIN Output Format

EXPLAIN returns a row of information for each table used in the SELECT statement. It lists the tables in the output in the order that MySQL would read them while processing the statement. MySQL resolves all joins using a nested-loop join method. This means that MySQL reads a row from the first table, and then finds a matching row in the second table, the third table, and so on. When all tables are processed, MySQL outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

EXPLAIN is used to get the query execution plan as follows:

EXPLAIN SELECT * FROM Users WHERE uid = 1;

Example :
To illustrate, consider the following simple example

```
mysql> EXPLAIN SELECT city.name, city.district FROM city, country WHERE
city.countrycode = country.code AND country.code = 'IND';
-----------------------------------------------------------------
-----
| id | select_type | table     | type   | possible_keys  | key        | key_len | ref     |
rows| Extra      |
-----------------------------------------------------------------
-----
|  1 | SIMPLE      | country | const  | PRIMARY       | PRIMARY | 3          | const  |
1     | Using index  |
|  1 | SIMPLE      | city       | ALL   | NULL             | NULL       | NULL   | NULL |
4079| Using where |
-----------------------------------------------------------------
---------2 rows in set (0.00 sec)
```

Here, the query is structured as a join between two tables and the *EXPLAIN* keyword describes how MySQL will process the join. It should be clear the current design will require MySQL to process only one record in the *country* table (which is indexed) but all 4079 records in the *city* table (which isn't).

MySQL and web:

Need for own MySQL programs

MySQL includes a set of client programs which are programs intended as a small, focused program with a specific, limited function. For example, mysqlimport loads data files into tables, mysqladmin performs administrative operations, and mysql allows to interact with the server to execute arbitrary SQL statements . The standard client programs handle only most common tasks that MySQL users need to perform, but applications sometimes have requirements that are not addressed by the capabilities of those clients.

Hence there is a need for writing our own MySQL-based programs for accessing our own databases. To make this possible, MySQL includes a client-programming library that gives the flexibility to satisfy the specialized requirements of the our applications. By providing access to the MySQL server, the client library opens up possibilities limited only by our own imagination.

The following are the benefits we gain by writing our own programs considering the capabilities of the mysql client and its interface to the MySQL server:

- **Customizing input handling :** With mysql, raw SQL statements can be entered . With our own programs, we can provide input methods for the user that are more intuitive and easier to use. Our program can eliminate the need for its users to know SQL. Input collection can be easy such as a command-line interface that prompts the user and reads a value like a form in a Web page.

  Also, the input collection provided by the user could be validated in a easy manner. For
  example, it checks date format of the user enter with format of MySQL. This enhances
  the safety and security of our applications.

  Some applications might not even involve a human user, such as when input for MySQL is generated by another program. You might configure your Web server to write log entries to MySQL rather than to a file.

- **Customizing the output :** The output can be customized to have nice-looking
  reports. It should be formatted by the user by including several specialized elements:
  - Customized headers.
  - Suppression of repeating values so that the values are printed only when they change.
  - Subtotal and total calculations.
  - Formatting of numbers, such as 94384.24, to print as dollar amounts, such as $94,384.24.

  Another common type of task involving complex formatting is invoice production, where you need to associate each invoice header with information about the customer and about each item ordered.

- **Working around constraints imposed by the nature of SQL itself :** SQL is not a procedural language with a set of flow control structures such as conditionals, loops, and subroutines. If you execute a file of SQL queries using mysql in batch mode, mysql either quits after the first error. By writing our own program, it's possible to selectively adapt to the success of queries by providing flow control around statement-execution operations.

MySQL 5.0 introduces support for stored functions and procedures. These can use flow control and error-handling constructs, which provides additional flexibility at the SQL level.

- **Integrating MySQL into any application :** It's possible to achieve "integration" of MySQL into an application by using a ***shell script*** that invokes ***mysql*** with an input file containing SQL statements, and then postprocessing the output using other utilities.. It can be more effective to use the ***client library*** to interact with the ***MySQL server directly***, extracting exactly the information we want at each phase of the application's execution.

Writing MySQL Programs Using C

MySQL provides a client library written in the C programming language that you can use to write client programs that access MySQL databases. This library defines an application programming interface that includes the following facilities:

- Connection management routines that establish and terminate a session with a server.
- Routines that construct SQL statements, send them to the server, and process the results.
- Status-checking and error-reporting functions for determining the exact reason for an error when an API call fails.
- Routines that help you process options given in option files or on the command line.

MySQL's Application Programming Interfaces.

Definition :

API means ***Application Program Interface*** and it is a set of programming instructions (through classes, methods, functions and variables) and standards for accessing a web-based software application or web tool. An **application programming interface** (API) is a source code interface that a computer system or program library provides to support requests for services to be made of it by a computer program.

An API differs from an application binary interface in that it is specified in terms of a programming language that can be compiled when an application is built, rather than an explicit low level description of how data is laid out in memory.

The MySQL Connectors and APIs are the drivers and libraries which are used to connect applications in different programming languages to MySQL database servers. The application and database server can be on the same machine, or communicate across the network.

APIs can be ***procedural*** or ***object-oriented.*** A procedural API is used for calling functions to carry out tasks, with the object-oriented API, we instantiate classes and then call methods on the resulting objects.

The software that provides the functionality described by an API is said to be an *implementation* of the API. The API itself is abstract, in that it specifies an interface and does not get involved with implementation details.

A good example of an API would be a web service interface, such as the API provided by Google for its mapping service.

Application programming interfaces

Many programming languages with language-specific APIs include libraries for accessing MySQL databases. Some of them are :

- MySQL Connector/Net for integration with Microsoft's Visual Studio (languages such as C# and VB are most commonly used)
- JDBC driver for Java.
- An ODBC interface called MySQL Connector/ODBC allows additional programming languages that support the ODBC interface to communicate with a MySQL database, such as ASP or ColdFusion.
- The HTSQL – URL-based query method also ships with a MySQL adapter, allowing
  direct interaction between a MySQL database and any web client via structured URLs.
- The C client library API - Primary programming interface to MySQL. It's used, for
  example, to implement the standard clients in the MySQL distribution, such as mysql,
  mysqladmin and mysqldump.
- The DBI (Database Interface) API for Perl. DBI is implemented as a Perl module that interfaces with other modules at the DBD (Database Driver) level, each of which provides access to a specific type of database engine.
- The PHP API. PHP is a server-side scripting language that provides a convenient way of embedding programs in Web pages.. For example, when the following short PHP script is embedded in a Web page, it displays the IP address of the client host that requested the page:

Choosing an API:

Choosing an API for various types of applications is done by comparing the capabilities of the API's like C, DBI, and PHP APIs which will give idea about their relative strengths and weaknesses, and when to choose one over another.

The following factors are considered while selecting API's for a particular task:

- **Intended execution environment:** The context in which the application is to be used.
- **Performance :** Efficient performance of applications when writing the API language.
- **Ease of development :** Easiness in making application writing while using API and its language.
- **Portability :** Usage of the application for database systems other than MySQL.

<u>Summary :</u>

- MySQL Storage engines are MySQL components that handle the SQL operations for different table types.
- There are many storage engines available in MySQL. The important among them are MyISAM, InnoDB and Memory storage engines.
- **MyISAM** was the default storage engine for the MySQL relational database management system versions prior to 5.5
- MyISAM provides high-speed storage and retrieval, supports full text searching.
- **InnoDB** storage engine is the default storage engine for MySQL 5.5 and higher.
- InnoDB is a storage engine for MySQL that balances high reliability and high performance.
- The MEMORY storage engine creates tables that are stored in memory. It stores all data in RAM for faster access than storing data on disks. Useful for quick looks up of reference and other identical data.
- The right storage engine is selected based on the features like Storage limits, Locking granularity ,Hash indexes etc.,
- In MySQL, a stored procedure is a program which passes parameters into. It does not return a value like a function does.
- A stored function is a special kind stored program that returns a single value. It is used to encapsulate common formulas or business rules that are reusable among SQL statements or stored programs.

- ❖ A trigger is a named database object that is associated with a table, and it activates when a particular event (e.g. an insert, update or delete) occurs for the table.
- ❖ A cursor is used to handle a result set inside a stored procedure . A cursor allows us to iterate a set of rows returned by a query and process each row accordingly.
- ❖ **Query Optimization :** Depending on the details of the MySQL tables, columns, indexes, and the conditions in the WHERE clause, the MySQL optimizer considers many techniques to efficiently perform the lookups involved in an SQL query The set of operations that the optimizer chooses to perform the most efficient query is called the "**query execution plan**", also known as the **EXPLAIN** plan.
- ❖ We need to write our own MySQL-based programs for accessing our own databases and applications.
- ❖ API means ***Application Program Interface*** and it is a set of programming instructions and standards for accessing a web-based software application or web tool.

Review Questions

Part – A (2 marks)

1. What is a storage engine in MySql?
2. What do you mean by Transactional Storage engines.
3. List some of the Mysql storage engines you know.
4. List the important features of MyISAm.
5. What are the advantages of InnoDB storage engine.
6. How will you create a Memory storage engine.
7. Write the syntax for creating a stored function in Mysql.
8. How will you invoke a stored procedure?
9. What is a Trigger?
10. Mention some of the uses of Mysql trigger.
11. What is the syntax for creating a trigger?
12. What is the use SHOW trigger statement?
13. List the properties of Mysql cursor.
14. Define : Query optimization in Mysql.
15. Write the syntax for Explain output format.
16. What is the necessity of writing own programs in mysql?
17. What is an API?
18.

Part – B (3 marks)

1. List the advantages of Non-Transactional storage engines.
2. List the advantages of MyISAM.
3. Mention the limitations of InnoDB storage engine.
4. What are the disadvantages of Memory storage engine.
5. Write about the three parameters used in stored procedure.

6. What are the various types of triggers available in mysql?
7. Write about Fetch statement used in cursor with its syntax .
8. How explain is used in optimizing queries in Mysql.
9. List the benefits of writing own programs in Mysql.
10. Mention some of the API's you know in Mysql.

Part – C (5 or 10 marks)

1. Discuss about any two of the Mysql's storage engines. (10 marks)
2. How will you choose the right storage engine? (5 marks)
3. With an example, explain Stored function of Mysql. (10 marks)
4. Discuss about the use of stored procedure in Mysql with example. (10 marks)
5. Write about the creation of triggers with an example. (10 marks)
6. How cursor is created in mysql. Explain with an example. (10 marks)
7. Discuss about query optimization using Explain . (10 marks)
8. Write short notes on :
   a) Use of various API's b) Selecting various API's . (10 marks)

UNIT – V DATA WAREHOUSING & INTRODUCTION TO BIG DATA

Learning Objectives

At the end of the unit, the students will be able to

- Define data warehousing and data mining
- Understand the functions of Warehousing
- Understand the architecture of data warehousing
- List the applications of data warehousing
- Explain the characteristics and applications of Big data
- Understand the various technologies and tools used for Big data
- Understand the concepts of NoSQL and different types of data stores
- Differentiate RDBMS and NoSQL
- Understand the types of Data stores in NoSQL
- Write queries using NoSQL

Data Warehousing

Definition

Data Warehousing is defined as a method of collecting information from many sources and storing it under a unique model at a single site. Data warehouse is a database which is kept separate from operational database. It stores consolidated historical data used to analyze and take business decisions in an organization.

Functions of Data Warehouse

The following are the important functions of data warehouse. They are

- **Gathering or collecting data**
- **Integration of schema or model**
- **Data Transformation and cleaning**
- **Updating data**
- **Summarizing data**

<u>Gathering or collecting data</u>

Data warehouse collects data from various data sources such as relational databases, flat files and on-line records. The collected data are stored in databases inside the warehouse. The type of data collection used depends on the architecture of the warehouse. In a source driven architecture, the data are collected continuously or periodically. But in destination – driven architecture, the warehouse periodically sends request to the data source to send data.

Integration of schema or model

Each and every data source uses different schema. But data warehouse gets data from different sources with schema and converts the data from various sources into a **common integrated schema.** The data warehouse won't copy the data from the sources. But it copy, converts and gives a view of data.

Data Transformation and cleaning

The task of correcting and preparing the data is called data cleaning. Data sources deliver data with many inconsistencies. Before storing the data into the database of data warehouse it should be corrected.

For example,

- Address list collected from many sources may have duplicates. The duplicates must be deleted. The operation of removing duplicates is called deduplication.
- Sometimes, it is necessary to change the units of measure of data to suit the integrated schema.

Data Warehouses have many graphical tools to support data transformation.

Updating Data

Updates on tables at the data sources must be sent to the data warehouse. If the tables in data warehouse are same as sources, the updation is easy.

Summarizing Data

The raw data generated by a transaction may be too large to store on line. Therefore, we can use summary of transactions for easy querying. For example, instead of storing data about every sale of clothing, we can store total sales of clothing by item name and category.

The different steps involved in getting data into a data warehouse are called extract, transform, and load or ETL tasks; extraction refers to getting data from the sources, correcting and preprocessing data is called **data cleansing (transform)**, while load refers to loading the data into the data warehouse.

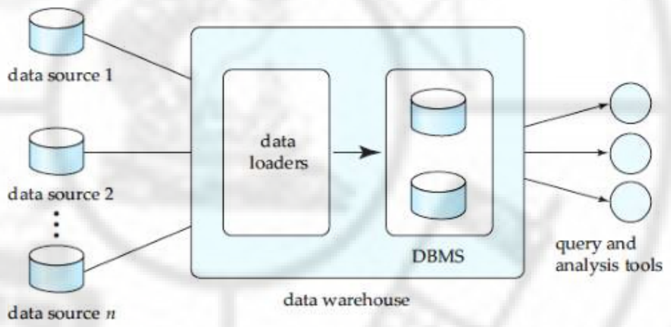
Data Warehouse Architecture

Different data warehousing systems have different structures. Some may have an operational data store (ODS), while some may have multiple data marts. Some may have a small number of data sources, while some may have dozens of data sources.

In general, all data warehouse systems have the following layers:

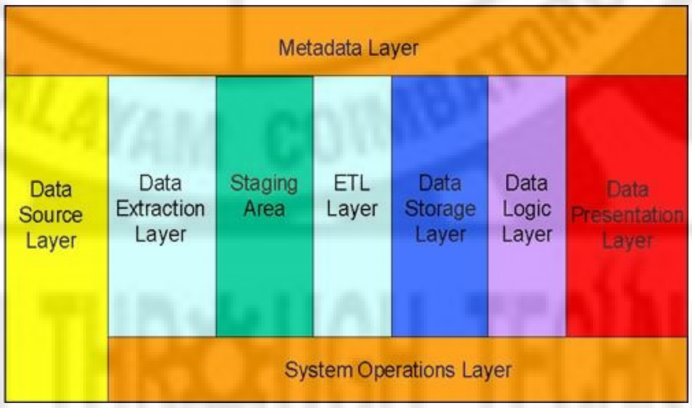
- Data Source Layer

- Data Extraction Layer
- Staging Area
- ETL Layer (Extract, Transform, Load)
- Data Storage Layer
- Data Logic Layer
- Data Presentation Layer
- Metadata Layer
- System Operations Layer



Data Warehouse Architecture

The picture below shows the relationships among the different components of the data warehouse architecture:



Different layers of Data Warehouses

Each component is discussed individually below:
<u>Data Source Layer</u>
This represents the different data sources that feed data into the data warehouse. The data source can be of any format -- plain text file, relational database, other types of database, Excel file, etc., can all act as a data source.

Many different types of data can be a data source:

- Operations - such as sales data, HR data, product data, inventory data, marketing data, systems data.
- Web server logs with user browsing data.
- Internal market research data.
- Third-party data, such as census data, demographics data, or survey data.

All these data sources together form the Data Source Layer.

Data Extraction Layer

Data gets pulled from the data source into the data warehouse system. There is likely some minimal data cleansing, but there is unlikely any major data transformation.

Staging Area

This is where data sits prior to being scrubbed and transformed into a data warehouse / data mart. Having one common area makes it easier for subsequent data processing / integration.

ETL Layer

This is where data gains its "intelligence", as logic is applied to transform the data from a transactional nature to an analytical nature. This layer is also where data cleansing happens. The ETL design phase is often the most time-consuming phase in a data warehousing project, and an ETL tool is often used in this layer.

Data Storage Layer

This is where the transformed and cleansed data sit. Based on scope and functionality, 3 types of entities can be found here: data warehouse, data mart, and operational data store (ODS). In any given system, you may have just one of the three, two of the three, or all three types.

Data Logic Layer This is where business rules are stored. Business rules stored here do not affect the underlying data transformation rules, but do affect what the report looks like.

Data Presentation Layer This refers to the information that reaches the users. This can be in a form of a tabular / graphical report in a browser, an emailed report that gets automatically generated and sent everyday, or an alert that warns users of exceptions, among others. Usually an OLAP tool and/or a reporting tool is used in this layer.

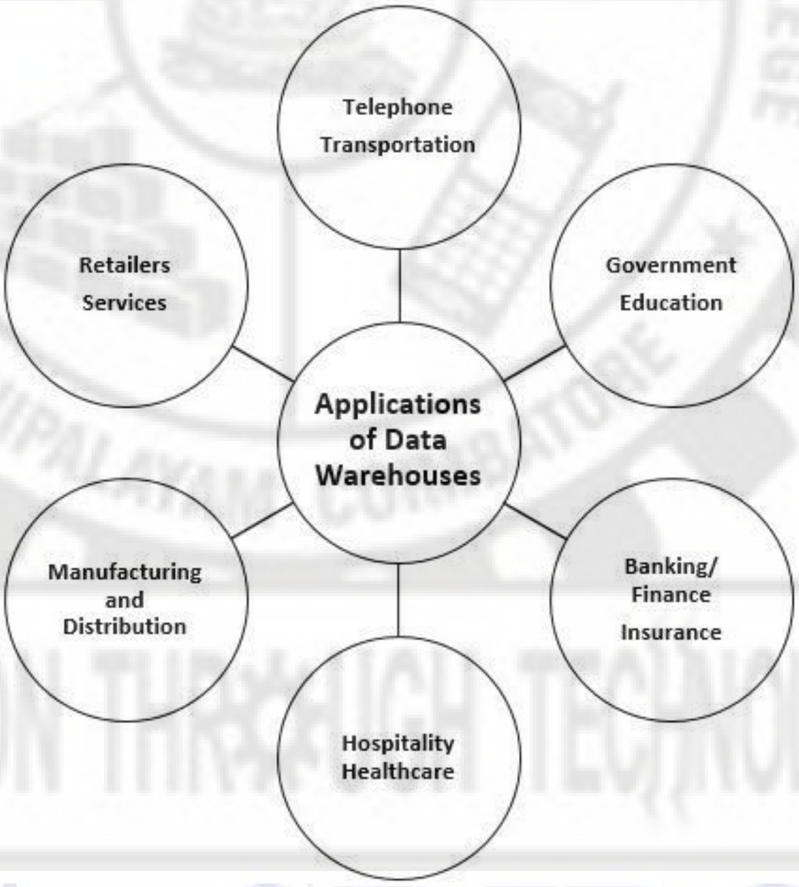
Metadata Layer This is where information about the data stored in the data warehouse system is stored. A logical data model would be an example of something that's in the metadata layer. A metadata tool is often used to manage metadata.

System Operations Layer This layer includes information on how the data warehouse system operates, such as ETL job status, system performance, and user access history.

Applications of Data warehouse

A data warehouse helps business executives to organize, analyze, and use the data for decision making. Data warehouses are widely used in the following fields

1. Financial services
2. Banking services
3. Consumer goods
4. Health care
5. Retail sectors
6. Insurance
7. Service sector
8. Telephone industry
9. Transportation industry
10. Education



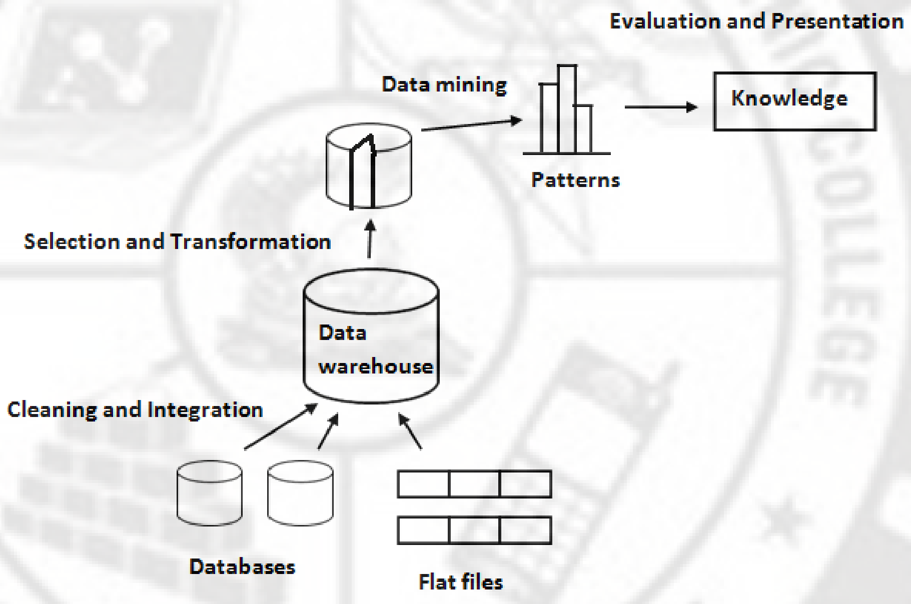
Applications of Data Warehouses

Data mining concepts

The process of extracting and finding hidden knowledge from large database is called **data mining**. This technology is used to help companies to find the most important hidden information from their data warehouse and used to plan business strategies.

The term **data mining** refers to the process of semi-automatically analyzing large databases to find useful patterns. Like knowledge discovery in artificial intelligence (also called machine learning) or statistical analysis, data mining attempts to discover rules and patterns from data. However, data mining differs from machine learning and statistics in that it deals with large volumes of data, stored primarily on disk. That is, data mining deals with "knowledge discovery in databases."

Steps in data mining



Data Mining

Various steps in data mining can be specified as follows

1) Data cleaning
    - Errors and inconsistencies in the data are removed.
2) Data Integration
    - Data from multiple sources are combined.
3) Data selection
    - The relevant data are retrieved from the database for analysis.
4) Data transformation
    - The selected data are converted to common schema using aggregation operations.
5) Data mining
    - Intelligent methods are used to extract hidden information as data patterns.
6) Pattern evaluation
    - The data patterns are evaluated based on certain criteria.
7) Knowledge presentation
    - The mined data is presented to the user using knowledge representation techniques.

Advantages of data mining

Industry

- help companies to take business decisions and strategies
- increase company revenue

Manufacturing

- improve product safety, usability and comfort.

Shopping and Advertising

- predict future trends, customer purchase habits
- create targeted advertising campaigns

Education

- analyse student learning behaviour and student performance

Text Mining

- classify documents, books, e-mail and web pages for search engines

Image Recognition

- recognize characters, identify human faces, etc.

Web Mining

- increase website optimization.
- analyse e-commerce websites and offer customized pages for customers

Fraud Detection

- find out those fraudulent acts and products available in the market
- detect fraudulent transactions in financial institutions

Big Data

Definition

This is the era of Big Data and these are undoubtedly revolutionary times. Massive amounts of data are being generated by the hour, from social media and from enterprises. It would be extremely foolish to waste this treasure trove by simply doing nothing about it. Enterprises have learnt to harvest Big Data to earn higher profits, offer better services and gain a deeper understanding of their target clientele.

Big Data basically refers to the huge amounts of data, both organised and unorganised, that enterprises generate on a day-to-day basis. In this context, the volume of data is not as relevant as what organisations do with the data. Analysis of Big Data can lead to insights that improve strategic business decision-making.

Examples of Big Data

The automotive industry: Ford's modern-day hybrid Fusion model yields up to 25GB of data per hour. This data can be used to interpret driving habits and patterns in order to prevent accidents, deflect collisions, etc.

Entertainment: The video game industry is using Big Data for examining over 500GB of organised data and 4TB of functional backlogs, each day.

The social media effect: About 500TB of fresh data gets added into the databases of social media site Facebook daily.

Characteristics of Big DataThe

four Vs of Big Data

Some of the common characteristicsof Big Data are depicted.

1. Volume: The volume of data is an important factor in deciding on its value. Hence, volume is one property that needs to be considered while handling Big Data.

2. Variety: This refers to assorted data sources and the nature of data, both structured and unstructured. Previously, spreadsheets and databases were the only origins of data considered in most of the practical applications. But these days, data in the form of e-mails, pictures, recordings, monitoring devices, etc., are also being considered in investigation applications.

3. Velocity: This term refers to how swiftly date is generated. How fast the data is created and refined to meet a particular need, determines its real potential. The velocity of Big Data is the rate at which date follows from sources like business procedures, application logs, websites, etc. The speed at which Big Data flows is very high and virtually non-stop.

4. Veracity: This refers to the incompatibility between the various formats that the data is being generated in, thus constraining the process of mining or managing the data profitably.

Various Technologies used

As we know the subject of Big Data is very broad and permeates many new technology developments. Here is an overview of some of the technologies that help users monetise Big Data.

1. **MapReduce:** This allows job implementation, with scalability crossing thousands of servers.

Map: Input dataset transforms into a different set of values.

Reduce: Many outputs of the Map task are united to form a reduced set of values.

2. **Hadoop**: This is the most admired execution of MapReduce, being a completely open source platform for handling Big Data. Hadoop is flexible enough to be able to work with many data sources, like aggregating data in order to do large scale processing, reading data from a database, etc.

3. **Hive:** This is an SQL-like link that allows BI applications to run queries beside a Hadoop cluster. Having been developed by Facebook, it has been made open source for a little while and is a higher-level concept of the Hadoop framework. Also, it allows everyone to make queries against data stored in a Hadoop cluster and has improved on Hadoop's functionality, making it ideal for BI users.

Applications

1. Banking and Securities

Applications of big data in the banking and securities industry

Retail traders, Big banks and other financial markets use big data for trade analytics, high frequency trading, pre-trade decision-support analytics, sentiment measurement, predictive analytics etc.

2. Communications, Media and Entertainment

Since consumers expect rich media on-demand in different formats and in a variety of devices, some big data challenges in the communications, media and entertainment industry include:

- Collecting, analyzing, and utilizing consumer insights
- Leveraging mobile and social media content
- Understanding patterns of real-time, media content usage
- Applications of big data in the Communications, media and entertainment industry Organizations in this industry simultaneously analyze customer data along with behavioral data to create detailed customer profiles that can be used to:
- Create content for different target audiences
- Recommend content on demand
- Measure content performance

3. Healthcare Providers

Some hospitals are using data collected from a cell phone app, from millions of patients, to allow doctors to use evidence-based medicine as opposed to administering several medical/lab tests to all patients who go to the hospital

4. Education

Big data is used quite significantly in higher education. An Australian university with over 26000 students has deployed a Learning and Management System that tracks among other things, when a student logs onto the system, how much time is spent on different pages in the system, as well as the overall progress of a student over time.

It is also used to measure teacher's effectiveness to ensure a good experience for both students and teachers. Teacher's performance can be fine-tuned and measured against student numbers, subject matter, student demographics, student aspirations, behavioural classification and several other variables.

5. Manufacturing and Natural Resources

In the natural resources industry, big data allows for predictive modelling to support decision making that has been utilized to ingest and integrate large amounts of data from geospatial data, graphical data, text and temporal data. Areas of interest where this has been used include; seismic interpretation and reservoir characterization.

6. Government

In public services, big data has a very wide range of applications including: energy exploration, financial market analysis, fraud detection, health related research and environmental protection.

Big data is being used in the analysis of large amounts of social disability claims, made to the Social Security Administration (SSA), that arrive in the form of unstructured data. The analytics are used to process medical information rapidly and efficiently for faster decision making and to detect suspicious or fraudulent claims.

7. Insurance

Big data has been used in the industry to provide customer insights for transparent and simpler products, by analyzing and predicting customer behaviour through data derived from social media, GPS-enabled devices and CCTV footage. The big data also allows for better customer retention from insurance companies.

When it comes to claims management, predictive analytics from big data has been used to offer faster service. Fraud detection has also been enhanced.

8. Retail and Whole sale trade

Big data from customer loyalty data, POS (Point of Sale), store inventory, local demographics data continues to be gathered by retail and wholesale stores.

9. Transportation

Governments' use of big data: traffic control, route planning, intelligent transport systems, congestion management (by predicting traffic conditions)

Private sector use of big data in transport: revenue management, technological enhancements, logistics and for competitive advantage (by consolidating shipments and optimizing freight movement)

Individual use of big data: includes route planning to save on fuel and time, for travel arrangements in tourism etc.

10. Energy and Utilities

Smart meter readers allow data to be collected almost every 15 minutes as opposed to once a day with the old meter readers. This granular data is being used to analyze consumption of utilities better which allows for improved customer feedback and better control of utilities use.

In utility companies the use of big data also allows for better asset and workforce management which is useful for recognizing errors and correcting them as soon as possible before complete failure is experienced.

The accessibility of Big Data, inexpensive product hardware, and new information managing and analytics software have come together to create a unique moment in the history of data analysis. We now have the capability that is necessary to examine these amazing data sets rapidly and cost-effectively, for the first time in history.

Overview of NoSQL

NoSQL database, also called **Not Only SQL**, is an approach to data management and database design that is useful for very large sets of distributed data. NoSQL, which encompasses a wide range of technologies and architectures, seeks to solve the scalability and big data performance issues that relational databases were not designed to address.

NoSQL is especially useful when an enterprise needs to access and analyse massive amounts of unstructured data or data that is stored remotely on multiple virtual servers in the cloud.

NoSQL technology was originally created and used by Internet leaders such as Facebook, Google, Amazon and others, who required database management systems that could write and read data anywhere in the world, while scaling and delivering performance across massive data sets and millions of users.

Difference between RDBMS and NoSQL

RDBMS

- **Stands for Relational Database Management System**
- It is completely a structured way of storing data.
- The amount of data stored in RDBMS depends on physical memory of the system or in other words it is vertically scalable.
- In RDBMS schema represents logical view in which data is organized and tells how the relation are associates.
- It is a mixture of open and closed development models. like oracle, apache and so on.
- RDBMS databases are table based databases This means that SQL databases represent data in form of tables which consists of n number of rows of data
- RDBMS have predefined schemas.
- For defining and manipulating the data RDBMS use structured query language i.e. SQL which is very powerful.
- **RDBMS database examples:** MySql, Oracle, Sqlite, Postgres and MS-SQL.
- RDBMS database is well suited for the complex queries as compared to NoSQL.
- If we talk about the type of data then RDBMS are not best fit for hierarchical data storage
- **Scalability:** RDBMS database is vertically scalable so to manage the increasing load by increase in CPU, RAM, and SSD on a single server.
- RDBMS is best suited for high transactional based application and it is more stable and promise for the atomicity and integrity of the data.
- RDBMS support large scale deployment and get support from there vendors.
- **Properties:** ACID properties (Atomicity, Consistency, Isolation, Durability).

NoSQL

- **Stands for Not Only SQL**
- It is completely a unstructured way of storing data.
- While in NoSQL there is no limit you can scale it horizontally.
- Work on only open source development models.
- NoSQL databases are document based, key-value pairs, graph databases or wide-column stores whereas NoSQL databases are the collection of key-value pair, documents, graph databases or wide-column stores which do not have standard schema definitions which it needs to adhered to.
- NoSQL have dynamic schema with the unstructured data.
- It uses UnQL i.e. unstructured query language and focused on collection of documents and vary from database to database.
- **NoSQL database examples:** MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j and CouchDb

- NoSQL is note well suited for complex queries on high level it dose not have standard interfaces to perform that queries.
- NoSQL is best bit for hierarchical data storage because it follows the key-value pair way of data similar to JSON. Hbase is the example for the same.
- **Scalability:** as we know NoSQL database is horizontally scalable so to handle the large traffic you can add few servers to support that.
- NoSQL is still rely on community support and for large scale NoSQL deployment only limited experts are available.
- **Properties:** Follow Brewers CAP theorem (Consistency, Availability and Partition tolerance).

Tools used in Big Data

The following are the top 10 tools used for Big Data.

- **Hadoop**: Hadoop is an open source software framework for storing and processing large scale of distributed data sets. Hadoop is known for the ability to process extremely large data in both, structured and unstructured formats, reliably replicating chunks of data to nodes in the cluster and making it available locally on the processing machine.

- **MapReduce**: MapReduce was originally developed by Google. As a programming model and software framework for writing applications, MapReduce rapidly process vast amounts of data in parallel on large clusters of computer nodes. Widely used by Hadoop and many other data processing applications.

- **GridGain**: GridGain is a Java based middleware for faster in-memory processing of Big Data in real time. GridGain is compatible with the Hadoop Distributed File System (HDFS). GridGain requires Windows, Linux or Mac OS operating system. It offers an alternative to MapReduce.

- **HPCC**: It is developed by LexisNexis Risk Solutions, HPCC is short for "high performance computing cluster". HPCC Systems delivers on a single platform, a single architecture and a single programming language for data processing. HPCC claims to offer superior performance than Hadoop.

- **Storm**: Storm is different from other tools with its distributed, real-time, fault-tolerant processing system, unlike the batch processing systems of Hadoop. With real-time computation capabilities, Storm is fast and highly scalable, often being described as the "Hadoop of real-time". Storm is fault-tolerant and works with nearly all programming languages, though typically Java is used. Descending from the Apache family, Storm is now owned by Twitter

- **Cassandra**: It is a highly scalable NoSQL database to monitor massive data across multiple data centers and the cloud. Apache Cassandra is used by many organizations with large, active datasets, including Netflix, Twitter, Urban Airship, Constant Contact, Reddit, Cisco and Digg. Its commercial support and services are available through third-party vendors. Originally developed by Facebook, it is now managed by the Apache Foundation.

- **HBase**: It is the non-relational data store for Hadoop. Being a column-oriented database management system, HBase is well suited for sparse data sets and is written in Java. Supports writing applications such as Avro, REST and Thrift. Developed as part of the Apache Hadoop project, HBase runs on top of Hadoop distributed file system.

- **MongoDB**: MongoDB was originally developed by 10gen, and was designed to support humongous databases. It's a NoSQL database written in C++ with document-oriented storage, full index support, replication and high availability, which scales horizontally without compromising on functionality. Commercial support is available through 10gen MongoDB. It is literally derived from the term 'humongous' and is the most popular NoSQL database system.

- **Neo4j**: It boasts performance improvements of up to 1000x or more when in comparison with relational databases. Stores data structured in graphs instead of tables and is a disk-based, fully transactional Java engine. Organizations can purchase advanced and enterprise versions from Neo Technology Developed by Neo Technologies, which is the world's leading graph database.

- **CouchDB**: CouchDB stores data in JSON documents that can be accessed via the web or query using JavaScript. It offers distributed scaling with fault-tolerant storage. Its Key featured include: On-the-fly document transformation, real-time change notifications, easy-to-use web administration.

Scalability

Scalability refers to the increase in application workload due to increase in traffic. Applications should be designed to handle and perform well when the usage of the application increases. From the NoSQL point of view, it means that collections and data entities should be modelled based on the current and future demand for the application. There should not be non-availability or degradation of performance due to an increase in the number of users or transactions in the database.

Data scalability is the ability of a system to store, manipulate, analyze, and process ever increasing amounts of data without reducing overall system availability, performance, or throughput.

Data scalability is achieved by a combination of more powerful processing capabilities and larger and efficient storage mechanisms.

Relational and hierarchical databases scale up by adding more processors, more storage and caching systems.

Data integrity and schemas are suited for handling transactional, normalized, uniform data. They handle unstructured or rapidly evolving data structures with difficulty or exponentially larger costs.

Often, a data store is busy because different people are accessing different parts of the dataset. In these circumstances we can support horizontal scalability by putting different parts of the data onto different servers—a technique that's called **sharding.**

Understanding storage architecture

A large number of data-storage systems on the cloud have been built in recent years, in response to data storage needs of extremely large-scale Web applications. These data-storage systems allow scalability to thousands of nodes, with geographic distribution, and high availability. Current data-storage systems also do not support SQL, and provide only a simple put()/get() interface. While cloud computing is attractive even for traditional databases, there are several challenges due to lack of control on data placement and geographic replication.

Big Data architecture

Big Data architecture comprises consistent, scalable and completely computerised data pipelines. The data pipelines collect raw data and transform it into something of value. Meanwhile, the Big Data engineer has to plan what happens to the data, the way it is stored in the cluster, how access is approved internally, what equipment to use for processing the data, and finally, the mode of providing access to the outside world.

Cloud-Based Databases

Cloud computing is a relatively new concept in computing that emerged in the late 1990s and the 2000s, first under the name (SAAS) **software as a service**. Initial vendors of software services provided specific customizable applications that they hosted on their own machines. The concept of cloud computing developed as vendors began to offer generic computers as a service on which clients could run software applications. A client can make arrangements with a cloud-computing vendor to obtain a certain number of machines of a certain capacity as well as a certain amount of data storage. Both the number of machines and the amount of storage can grow and shrink as needed. In addition to providing computing services, many vendors also provide other services such as data storage services, map services, and other services that can be accessed using a Web-service application programming interface.

Further, as the needs of the enterprise grow, more resources (computing and storage) can be added as required; the cloud-computing vendor generally has very large clusters of computers, making it easy for the vendor to allocate resources on demand. A variety of vendors offer cloud services. They include traditional computing vendors as well as companies, such as amazon and Google. Web applications that need to store and retrieve data for very large numbers of users (ranging from millions to hundreds of millions) have been a major driver of cloud-based databases.

Thus, cloud data-storage systems are based on two primitive functions, put(key, value), used to store values with an associated key, and get(key), which retrieves the stored value associated with the specified key. Some systems such as Big table additionally provide range queries on key values. In Big table, a record is not stored as a single value, but is split into component attributes that are stored separately. Thus, the key for an attribute value conceptually consists of (record-identifier, attribute-name). Each attribute value is just a string as far as Big table is concerned. To fetch all attributes of a record, a range query, or more precisely a prefix-match query consisting of just the record identifier, is used. The get () function returns the attribute names along with the values. For efficient retrieval of all attributes of a record, the storage system stores entries sorted by the key, so all attribute values of a particular record are clustered together. In fact, the record identifier can be structured hierarchically. In Big table the record identifier is just a string.

Data Storage Systems on the Cloud

Applications on the Web have extremely high scalability requirements. Popular applications have hundreds of millions of users, and many applications have seen their load increase many fold within a single year, or even within a few months. To handle the data management needs of such applications, data must be partitioned across thousands of processors. A number of systems for data storage on the cloud have been developed and deployed over the past few years to address data management requirements of such applications; these include Big table from Google, Simple Storage Service (S3) from Amazon, which provides a Web interface to Dynamo, which is a key value storage system, Cassandra, from FaceBook, which is similar to Big table, and Sherpa/PNUTS from Yahoo!, the data storage component of the Azure environment from Microsoft, and several other systems.

Data-storage systems typically allow multiple versions of data items to be stored. Versions are often identified by timestamp, but may be alternatively identified by an integer value that is incremented whenever a new version of a data item is created. In Big table, for example, a key consists of three parts record-identifier, attribute-name and timestamp.

Partitioning and Retrieving Data

Partitioning of data is the key to handling extremely large scale in data-storage systems. Unlike regular parallel databases, it is usually not possible to decide on a partitioning function ahead of time. Further, if load increases, more servers need to be added and each server should be able to take on parts of the load incrementally.

To solve both these problems, data-storage systems typically partition data into relatively small units (in the order of hundreds of megabytes). These partitions are often called tablets. Each tablet is a fragment of a table. The partitioning of data should be done on the search key, so that a request for a specific key value is directed to a single tablet; otherwise each request would require processing at multiple sites, increasing the load on the system greatly. Two approaches are used: either range partitioning is used directly on the key, or a hash function is applied on the key, and range partitioning is applied on the result of the hash function.

The partitioning of data into tablets happens dynamically. As data are inserted, if a tablet grows too big, it is broken into smaller parts. Further, even if a tablet is not large enough to merit being broken up, if the load (get/put operations) on that tablet is excessive, the tablet may be broken into smaller tablets, which can be distributed across two or more sites to share the load. Usually the number of tablets is much larger than the number of sites, for the same reason that virtual partitioning is used in parallel databases.

Transactions and Replication

A data-storage system on the cloud must be able to continue normal processing even with many sites down. Such systems replicate data (such as tablets) to multiple machines in a cluster, so that a copy of the data is likely to be available even if some machines of a cluster are down. (A **cluster** is a collection of machines in a data center.) For example, the *Google File System* (GFS), which is a distributed fault-tolerant file system, replicates all file system blocks at three or more nodes in a cluster. Normal operation can continue as long as at least one copy of the data is available (key system data, such as the mapping of files to nodes, is replicated at more nodes, a majority of which need to be available). In addition, replication is also used across geographically distributed clusters.

Although a database system provides a high-level view of data, ultimately data have to be stored as bits on one or more storage devices. A vast majority of databases today store data on magnetic disk (and, increasingly, on flash storage) and fetch data into main memory for processing, or copy data onto tapes and other backup devices for archival storage. The physical characteristics of storage devices play a major role in the way data are stored, because access to a random piece of data on disk is much slower than memory access: Disk access takes tens of milliseconds, whereas memory access takes a tenth of a microsecond.

Web applications that need to store and retrieve data for very large numbers of users (ranging from millions to hundreds of millions) have been a major driver of cloud-based databases. The needs of these applications differ from those of traditional database applications, since they value availability and scalability over consistency. Several cloud-based data-storage systems have been developed in recent years to serve the needs of such applications.

Applications on the Web have extremely high scalability requirements. Popular applications have hundreds of millions of users, and many applications have seen their load increase many fold within a single year, or even within a few months. To handle the data management needs of such applications, data must be partitioned across thousands of processors.

Types of Data stores in NoSQL

Introduction

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These databases can broadly be categorised into four types. Each of these categories has its own specific attributes and limitations. There is no a single solution which is better than all the others; however there are some databases that are better to solve specific problems. The most common categories are:

- Column oriented data store
- Document oriented store
- Key-value store
- Graph store

Column store NoSQL database

In column-oriented NoSQL databases, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at runtime or while defining the schema. Read and write is done using columns rather than rows. Column families are groups of similar data that is usually accessed together. As an example, we often access customers' names and profile information at the same time, but not the information on their orders.

The main advantages of storing data in columns over relational DBMS are fast search/access and data aggregation. Relational databases store a single row as a continuous disk entry. Different rows are stored in different places on the disk while columnar databases store all the cells corresponding to a column as a continuous disk entry thus making the search/access faster.

Each column family can be compared to a container of rows in an RDBMS table, where the key identifies the row and the row consists of multiple columns. The difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add them to other rows.

Use cases: Developers mainly use column databases in:

- Content management systems
- Blogging platforms
- Systems that maintain counters
- Services that have expiring usage
- Systems that require heavy write requests (like log aggregators)

Column store databases should be avoided if you have to use complex querying or if your querying patterns frequently change. Also avoid them if you don't have an established database requirement. Examples of column store NoSQL databases are **Cassandra** and Apache **Hadoop, HBase**.

Document store NoSOL database

Document store NoSQL databases are similar to key-value databases in that there's a key and a value. Data is stored as a value. Its associated key is the unique identifier for that value. The difference is that, in a document database, the value contains structured or semi-structured data. This structured/semi-structured value is referred to as a document and can be in XML, JSON or BSON format.

Use cases: Document store databases are preferable for:

- E-commerce platforms
- Content management systems.
- Analytics platforms
- Blogging platforms

Document store NoSQL databases are not the right choice if you have to run complex search queries or if your application requires complex multiple operation transactions. Examples of document store NoSQL databases are **MongoDB, Apache CouchDB and Elasticsearch.**

Key-value store NoSQL database

From an API perspective, key-value stores are the simplest NoSQL data stores to use. The client can get the value for the key, assign a value for a key or delete a key from the data store. The value is a blob that the data store just stores, without caring or knowing what's inside; it's the responsibility of the application to understand what was stored. Since key- value stores always use primary-key access, they generally have great performance and can be easily scaled. The key- value database uses a hash table to store unique keys and pointers (in some databases it's also called the inverted index) with respect to each data value it stores. There are no column type relations in the database; hence, its implementation is easy. Key-value databases give great performance and can be very easily scaled as per business needs.

Use cases: Here are some popular use cases of the key-value databases:

- For storing user session data
- Maintaining schema-less user profiles
- Storing user preferences
- Storing shopping cart data

However key-value databases are not the ideal choice for every use case when:

- We have to query the database by specific data value.
- We need relationships between data values.
- We need to operate on multiple unique keys.
- Our business needs updating a part of the value frequently.

Examples of this database are **Redis, MemcacheDB and Riak.**

Graph based NoSQL database

Graph databases are basically built upon the Entity - Attribute - Value model. Entities are also known as nodes, which have properties. It is a very flexible way to describe how data relates to other data. Nodes store data about each entity in the database, relationships describe a relationship between nodes, and a property is simply the node on the opposite end of the relationship. Whereas a traditional database stores a description of each possible relationship in foreign key fields or junction tables, graph databases allow for virtually any relationship to be defined on-the-fly.

Use cases: Graph base NoSQL databases are usually used in:

- Fraud detection
- Graph based search
- Network and IT operations
- Social networks, etc

CRUD (Create, Read, Update, Delete) operations

In computer programming **create, read, update, and delete** (as an acronym **CRUD**) are the four basic functions in data bases. Alternate words are sometimes used when defining the four basic functions of *CRUD*, *retrieve* instead of *read*, *modify* instead of *update*, or *destroy* instead of *delete*. *CRUD* is also sometimes used to describe user interface conventions that facilitate viewing, searching, and changing information; using computer-based forms and reports. The term was first popularized by James Martin in his 1983 book *Managing the Data-base Environment*. The acronym may be extended to CRUDL to cover ***listing*** of large data sets.

The acronym CRUD refers to all of the major functions that are implemented in relational database applications (SQL).

Operation	**SQL**

Create	INSERT
Read (Retrieve)	SELECT
Update (Modify)	UPDATE
Delete (Destroy)	DELETE

CRUD is also relevant at the user interface level of most applications. For example, in address book software, the basic storage unit is an individual contact entry. As a bare minimum, the software must allow the user to Create or add new entries

Read, retrieve, search, or view existing entries

Update or edit existing entries

Delete/deactivate/remove existing entries

Without these four operations, the software cannot be considered complete.

Create Operation

Create or insert operation adds new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

MongoDB provides the following methods to insert documents into a collection:

- db.collection.insertOne()
- db.collection.insertMany()

In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

db.inventory.insertOne({ item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } })

Insert Multiple Documents

db.collection.insertMany() can insert *multiple* documents into a collection. Pass an array of documents to the method.
The following example inserts three new documents into the inventory collection. If the documents do not specify an _id field, MongoDB adds the _id field with an ObjectId value to each document. See Insert Behavior.

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm"
} }
])
```

Read/access Operation

Read operation retrieves documents from a collection; i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

- db.collection.find()

You can specify query filters or criteria that identify the documents to return.

Select All Documents in a Collection

To select all documents in the collection, pass an empty document as the query filter parameter to the find method. The query filter parameter determines the select criteria:

db.inventory.find({})

This operation corresponds to the following SQL statement:

SELECT * FROM inventory

Although you can express this query using the $or operator, use the $in operator rather than the $or operator when performing equality checks on the same field.

The operation corresponds to the following SQL statement:

SELECT * FROM inventory WHERE status in ("A", "D")

Update Operations

Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

- db.collection.updateOne()
- db.collection.updateMany()
- db.collection.replaceOne()

In MongoDB, update operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as read operations.

Examples for using update in the mongo shell:

- db.collection.updateOne(<filter>, <update>, <options>)

- db.collection.updateMany(<filter>, <update>, <options>)
- db.collection.replaceOne(<filter>, <replacement>, <options>)

The following examples use the inventory collection. To create and/or populate the inventory collection, run the following:

```
db.inventory.insertMany( [
   { item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status: "A" },
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "A" },
   { item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" }, status: "P" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" },
   { item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" }, status: "A" }
]);
```

Update Documents in a Collection

To update a document, MongoDB provides update operators, such as $set, to modify field values.

To use the update operators, pass to the update methods an update document of the form:

```
{
  <update operator>: { <field1>: <value1>, ... },
  <update operator>: { <field2>: <value2>, ... },
  ...
}
```

Some update operators, such as $set, will create the field if the field does not exist. See the individual update operator reference for details.

Update a Single Document

The following example uses the db.collection.updateOne() method on the inventory collection to update the *first* document where item equals "paper":

```
db.inventory.updateOne(
   { item: "paper" },
   {
     $set: { "size.uom": "cm", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

The update operation:

- uses the $set operator to update the value of the size.uom field to "cm" and the value of the status field to "P",
- uses the $currentDate operator to update the value of the lastModified field to the current date. If lastModified field does not exist, $currentDate will create the field. See $currentDate for details.

Delete Operations

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

db.collection.deleteOne()

db.collection.deleteMany()

In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as read operations.

Delete All Documents

To remove all documents from a collection, pass an empty filter document {} to the db.collection.deleteMany() method.

The following example deletes all documents from the inventory collection:

db.inventory.deleteMany({})

Querying NoSQL Stores

There are substantial differences in the querying capabilities of different NoSQL data stores. The key/value stores design often provide only a lookup by primary key or some id field and lack capabilities to query any further fields, other data stores like the document databases CouchDB and MongoDB allow for complex queries—at least static ones predefined on the database nodes (as in CouchDB). In the design of many NoSQL databases rich dynamic querying features have been omitted in favor of performance and scalability. On the other hand, also when using NoSQL databases, there are use-cases requiring at least some querying features for non-primary key attributes.

All **key-value stores** can query by the key. If you have requirements to query by using some attribute of the value column, it's not possible to use the database: Your application needs to read the value to figure out if the attribute meets the conditions.

While using key-value stores, lots of thought has to be given to the design of the key. Can the key be generated using some algorithm? Can the key be provided by the user (user ID, email, etc.)? Or derived from timestamps or other data that can be derived outside of the database? These query characteristics make key-value stores likely candidates for storing session data (with the session ID as the key), shopping cart data, user profiles, and so on. The expiry_secs property can be used to expire keys after a certain time interval, especially for session/shopping cart objects.

Document databases provide different query features. CouchDB allows you to query via views—complex queries on documents which can be either materialized or dynamic. With CouchDB, if you need to aggregate the number of reviews for a product as well as the average rating, you could add a view implemented via map-reduce to return the count of reviews and the average of their ratings.

One of the good features of document databases, as compared to key-value stores, is that we can query the data inside the document without having to retrieve the whole document by its key and then introspect the document. This feature brings these databases closer to the RDBMS query model.

In **Column store** (Cassandra) the columns and column families are optimized for reading the data, as it does not have a rich query language; as data is inserted in the column families, data in each row is sorted by column names. If we have a column that is retrieved much more often than other columns, it is better to use that value for the row key instead.

Cassandra has a query language that supports SQL-like commands, known as Cassandra Query Language (CQL).

Basic queries that can be run using a Cassandra client include the GET, SET, and DEL. Before starting to query for data, we have to issue the keyspace command use ecommerce;. This ensures that all of our queries are run against the keyspace that we put our data into. Before starting to use the column family in the keyspace, we have to define the column family. CQL has many more features for querying data, but it does not have all the features that SQL has. CQL does not allow joins or subqueries, and its 'where clauses' are typically simple.

Graph databases are supported by query languages such as Gremlin. Gremlin is a domain specific language for traversing graphs; it can traverse all graph databases that implement the Blueprints property graph. Neo4J also has the Cypher query language for querying the graph. Outside these query languages, Neo4J allows you to query the graph for properties of the nodes, traverse the graph, or navigate the nodes relationships using language bindings.

Properties of a node can be indexed using the indexing service. Similarly, properties of relationships or edges can be indexed, so a node or edge can be found by the value. Indexes should be queried to find the starting node to begin a traversal. Let's look at searching for the node using node indexing.

NoSQL in cloud

Cloud computing is popular in recent years for all the benefits it offers. Companies have been increasingly turning to cloud environments to host their applications and databases to take advantage of:

- **Faster time to market.** You can get your system up in minutes or hours as opposed to days and months
- **Cost savings.** Pay only for what you need with no upfront investment in infrastructure or personnel
- **Flexibility.** You can easily adjust cloud resources to match your demand
- **Resource efficiency.** You can tailor the amount of bandwidth, processing and storage capability to your needs
- **Reliability.** Distributed servers located across the globe spells better disaster recovery and lower latency for locally served users

MongoDB, the leading NoSQL database according to DB-Engines rankings and over 10 million downloads, is especially well-suited for the cloud. With a native scale-out architecture enabled by a feature called “sharding,” you can easily grow your MongoDB deployment to meet additional demand. Replica sets, or redundant servers, in MongoDB help ensure high availability and data integrity even if individual cloud instances are taken offline.

When it comes to choosing a NoSQL database in the cloud, many companies go with MongoDB for its extensive partner network which includes leading cloud service providers such as Amazon. The Amazon Web Services solution is especially favoured by those who require high-performance operations on large datasets.

A key disadvantage of SQL Databases is the fact that SQL Databases are at a high abstraction level. This is a disadvantage because to do a single statement, SQL often requires the data to be processed multiple times. This takes more time. For instance, multiple queries on SQL Data occur when there is a ‘Join’ operation. *Cloud computing environments need high-performing and highly scalable databases.*

The general definition of a NOSQL data store is that it manages data that is not strictly tabular and relational, so it does not make sense to use SQL for the creation and retrieval of the data. NOSQL data stores are usually non-relational, distributed, open-source, and horizontally scalable.

A major advantage of NoSQL Databases is the fact that Data replication can be done more easily than it would be with SQL Databases.

The disadvantage of SQL databases is the fact that there is always a schema involved. Over time, requirements will definitely change and the database somehow has to support this new requirements. This can lead to serious problems. “Just imagine” the fact that applications need two extra fields to store data. Solving this issue with SQL Databases might get very hard. NoSQL databases support a changing environment for data and are a better solution in this case as well.

SQL Databases have the advantage over NoSQL Databases to have better support for “Business Intelligence”.

Cloud Computing Platforms are made for a great number of people and potential customers. This means that there will be millions of queries over various tables, millions or even billions of read and write operations within seconds. SQL Databases are built to serve another market: the “business intelligence”, where fewer queries are executed.

Amazon SimpleDB

Amazon SimpleDB is a highly available NoSQL data store that off-loads the work of database administration. Developers simply store and query data items via web services requests and Amazon SimpleDB does the rest. Amazon SimpleDB is a distributed database written in Erlang by Amazon.com.

Amazon SimpleDB creates and manages multiple geographically distributed replicas of data automatically to enable high availability and data durability. The service, charges only for the resources actually consumed in storing data and serving requests. Data model can be changed on the fly, and data is automatically indexed.

With Amazon SimpleDB, user can focus on application development without worrying about infrastructure provisioning, high availability, software maintenance, schema and index management, or performance tuning.

Benefits

Low touch
Amazon SimpleDB automatically manages infrastructure provisioning, hardware and software maintenance, replication and indexing of data items, and performance tuning.

Highly available
Amazon SimpleDB automatically creates multiple geographically distributed copies of each data item we store. This provides high availability and durability – if one replica fails, Amazon SimpleDB can change to another replica in the system.

Flexible
As business changes or application evolves user can simply add another attribute to the Amazon SimpleDB data set when needed.

Simple to use
Amazon SimpleDB provides streamlined access to the store and query functions that are traditionally achieved using a relational database cluster – while leaving out other complex, often-unused database operations. The service allows you to quickly add data and easily retrieve or edit that data through a simple set of API calls.

- Monitoring or tracking
- Metering
- Trend of business analysis

- Auditing
- Archival or regulation compliance

Online Games

For developers of online games on any platform, Amazon SimpleDB offers a highly-available, scalable, and administration-free database solution for user and game data.
Common data online games can store, index, and query with Amazon SimpleDB includes:

- User scores and achievements
- User settings or preferences
- Information about a player's items or user-generated content
- Game session state (when play is saved or interrupted)
- Dynamic game content.
- Indexed metadata for large objects used by the game and stored in Amazon S3

Review Questions

PART A

1. Define big data.
2. What is data warehousing?
3. List any three layers of data ware housing architecture
4. list the characteristics of Big Data.
5. What is veracity in Big data characteristics.
6. Define scalability of NoSQL.
7. What is Hadoop?
8. What is Hive?
9. Define NoSQL.
10. What is sharding?
11. What are the tools used in Big Data?
12. Define column oriented data store.
13. List any three examples for document store data bases.
14. List the examples for column store NoSQL database.
15. What is CRUD in NoSQL?
16. What is MongoDB?
17. What is Amazon Simple DB?

PART B

1. Explain any three functions of warehousing
2. Draw the diagram of data warehouse architecture.
3. List out any three applications of data warehousing.
4. What are the technologies used for Big Data?
5. List any three major differences between the RDBMS and NoSQL
6. List out the types of data stores in NoSQL.
7. List the commands for creating documents in a MongoDB.
8. Explain creatin database in NoSQL.
9. Explain pretty() and find () methods in NoSQL

PART C

1. Explain the architecture of data warehouse.
2. Explain the functions of data warehouse.
3. Explain the applications of data warehouse.
4. What is data mining? Explain the advantages of data mining.
5. Explain the various steps in data mining.
6. Explain the advantages of data mining.
7. Explain the characteristics of big data.
8. List and explain the technologies used for big data.
9. Discuss the applications of big data in detail.
10. Discuss the benefits of NoSQL.
11. Explain the difference between the RDBMS and NoSQL.
12. List and explain the tools used in big data.
13. Explain the storage architecture of NoSQL.
14. Explain the different types of data stores in NoSQL.
15. Explain the basic operations of NoSQL.
16. Explain the various types of queries in NoSQL.
17. What is Amazon Simple DB? Explain the benefits of SimpleDB.