DATABASE MANAGEMENT SYSTEMS
UNIT V – TOPIC 1
Triggers

A trigger is a procedure which is automatically invoked by the DBMS in response to changes to the database, and is specified by the database administrator (DBA).

A database with a set of associated triggers is generally called an active database.

In MySQL, a trigger is a stored database object that is automatically executed or fired on occurrence of a DML command (also satisfying a specified condition.)

Triggers are used to enforce business rules, validate input data, and maintain an audit trail.

Parts of a Trigger:

A Trigger description contains three parts:

Event-Condition-Action (ECA)

Event is a change to the database which activates the trigger -

Trigger Event - INSERT | UPDATE | DELETE

Trigger Activation Time – BEFORE | AFTER

Condition is a query that is run when the trigger is activated - SQL condition

Action is a procedure that is executed when a trigger is activated due to meeting the specified condition

All data actions performed by the trigger, execute within the same transaction in which the trigger fires.

Cannot contain transaction control statements (COMMIT, SAVEPOINT, ROLLBACK)

Uses of Trigger:

1. **Enforcing Business Rules :**Triggers can ensure that certain business rules are automatically enforced within the database. For example, a trigger can ensure that an employee's salary cannot be decreased. Passwords must be changed every month, insertion not allowed beyond working hours.
2. **Validating Data:** Triggers can be used to validate data before it is inserted or updated in the database. For example, ensuring that an email address has a valid format.
3. **Maintaining Audit Trails:** Triggers can be used to keep an audit trail of changes made to important data. This is useful for tracking who changed what data and when.
4. **Synchronizing Tables:** Triggers can be used to synchronize data between tables. For example, if you have a master table and several dependent tables, you can use triggers to update dependent tables whenever the master table is modified.
5. **Preventing Invalid Transactions**: Triggers can be used to prevent certain transactions that could lead to invalid or inconsistent data. For example, preventing deletion of records that are referenced by other records.

6. **Automatic Calculations :** Triggers can be used to perform automatic calculations and updates. For example, updating the total amount in an order whenever an order line item is added or updated.
7. **Enforcing Referential Integrity:** Triggers can enforce referential integrity rules, such as cascading updates or deletes, beyond what foreign keys alone can do.
8. **Data Transformation :** Triggers can be used to transform data before it is stored. For example, automatically converting all text to uppercase.

Types of Trigger

In MySQL, triggers are always defined at the row level, meaning they execute once for each row affected by the triggering event.

We can define the maximum six types of actions or events in the form of triggers:

Before Insert: It is activated before the insertion of data into the table.

After Insert: It is activated after the insertion of data into the table.

Before Update: It is activated before the update of data in the table.

After Update: It is activated after the update of the data in the table.

Before Delete: It is activated before the data is removed from the table.

After Delete: It is activated after the deletion of data from the table.

Trigger Syntax:

```
CREATE TRIGGER trigger_name
   (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
       ON table_name FOR EACH ROW
       BEGIN
      --variable declarations
      --trigger code
      END;
```

NOTE: When a trigger code contains multiple statements, we can change the delimiter from the default ; to some other character.

Example-1:

Let us create a trigger that would make a log entry whenever a new record is inserted into the EMP table along with the timestamp.

1. CREATE A TABLE EMP_LOG as below

```
CREATE TABLE EMP_LOG (
LOG_ID INT AUTO_INCREMENT PRIMARY KEY,
EMP_ID INT, MSG VARCHAR(50),
CHANGE_TIME TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

2. Create the trigger as below

```
CREATE TRIGGER emp_insert_log
after INSERT ON emp
FOR EACH ROW
  INSERT INTO EMP_LOG (EMP_ID, MSG)
  VALUES (NEW.EMPNO, "ROW INSERTED AT");
```

3. TESTING: - run the following commands and check the output
    a. Insert record into emp table
    b. Select * from emp_log (check the new record inserted automatically)

Example-2:

Create a trigger for making the log entry whenever the salary is updated recording the empno, old and new salary

1. CREATE A TABLE SAL_UPDATE_LOG as below

```
CREATE TABLE SAL_UPDATE_LOG (
LOG_ID INT AUTO_INCREMENT PRIMARY KEY,
EMP_ID INT,
OLD_SALARY FLOAT(7,2), NEW_SALARY FLOAT(7,2),
CHANGE_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

2. CREATE THE TRIGGER

```
CREATE TRIGGE emp_log
after UPDATE ON emp
FOR EACH ROW
  INSERT INTO SAL_UPDATE_LOG (EMP_ID, OLD_SALARY, NEW_SALARY)
  VALUES (NEW.EMPNO, OLD.SAL, NEW.SAL);
```

3. TESTING:
    a. Update salary for an employee in emp table
    b. Select * from sal_update_log

Example – 3:

Create a trigger to be executed before insert into emp which ensures the minimum salary of the new employees is 30000

```
DELIMITER //
Create Trigger before_insert_empsalary
BEFORE INSERT ON emp FOR EACH ROW
BEGIN
IF NEW.sal < 30000 THEN SET NEW.sal = 30000;
END IF;
```

```
END //
DELIMITER ;
```

TESTING:

Insert a new record into emp table with salary less than 30000
Select * from emp
Ensure that the salary is set to 30000 in the newly entered row.

Triggers that raise the error:

In MySQL, you can create a trigger that will raise an error when a certain condition is met by using the SIGNAL SQL statement. The SIGNAL statement allows you to set an error condition, which effectively raises an error.

SIGNAL SQLSTATE '45000': Raises an error with a custom SQLSTATE value of '45000', which is a generic state indicating an error condition.

SET MESSAGE_TEXT = ‘xxxxxxxxx': Sets the error message that will be returned.

Example:

Let us assume we should not allow insertion into emp table beyond office hours [9am-6pm]

```
DELIMITER //

CREATE TRIGGER control_access
BEFORE INSERT ON emp
FOR EACH ROW
BEGIN
   DECLARE HR INT;
   SET HR = HOUR(CURTIME());
   IF HR < 9 AND HR >18 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'INSERTION IS PROHIBITED AT THIS TIME';
   END IF;
END //

DELIMITER ;
```

Listing the available Triggers:

List all the available triggers on a database

SHOW TRIGGERS;

```
----------------+
| Trigger                | Event  | Table | Statement                                                    | Timing | Created                                                                  |      sql_mode
| Definer       | character_set_client | collation_connection | Database Collation |
--+--------------------+
| before_insert_empsalary | INSERT | emp   | BEGIN
IF NEW.sal < 30000 THEN SET NEW.sal = 30000;
END IF;
END            |            BEFORE            |            2024-07-11            14:45:16.49            |
ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZER
O_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION            |
root@localhost | cp850              | cp850_general_ci     | utf8mb4_0900_ai_ci |
+------------------------+---------+-------+--
```

Deleting a Trigger:

We can drop/delete/remove a trigger in MySQL using the DROP TRIGGER statement.

DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;

Example:

DROP TRIGGER before_insert_empsalary;

DATABASE MANAGEMENT SYSTEMS
UNIT V – TOPIC 2
Stored Procedures

A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database.

It is a subroutine or a subprogram in the regular computing language. A procedure always contains a name, parameter lists, and SQL statements.

We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc.

Stored Procedure Features

- Stored Procedure increases the performance of the applications. Once stored procedures are created, they are compiled and stored in the database.
- Stored procedure reduces the traffic between application and database server. Because the application has to send only the stored procedure's name and parameters instead of sending multiple SQL statements.
- Stored procedures are reusable and transparent to any applications.
- A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permissions on the database tables.

Syntax :

The following syntax is used for creating a stored procedure in MySQL. It can return one or more value through parameters or sometimes may not return at all. By default, a procedure is associated with our current database. But we can also create it into another database from the current database by specifying the name as database_name.procedure_name.

```
DELIMITER //
CREATE PROCEDURE procedure_name [ [IN  | OUT  | INOUT]
parameter_name datatype [, parameter_name datatype]) ]
BEGIN
   Declaration_section
   Executable_section
END //
DELIMITER ;
```

Parameter Name	Descriptions
procedure_name	It represents the name of the stored procedure.
Parameter	presents the number of parameters. It can be one or more than one.
Declaration_section	It represents the declarations of all variables.
Executable_section	It represents the code for the function execution.

Different types of parameters used:

1. IN parameter

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

2. OUT parameters

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

3. INOUT parameters

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

Calling the Procedure:

We can use the CALL statement to call a stored procedure. This statement returns the values to its caller through its parameters (IN, OUT, or INOUT). The following syntax is used to call the stored procedure in MySQL:

CALL procedure_name (parameter(s))

Examples:

1. Procedures without Parameters

Suppose we want to display the list and count of employees who are earning commission.

```
DELIMITER //
CREATE PROCEDURE comm_details()
BEGIN
   SELECT * FROM emp WHERE comm  IS NOT NULL;
   SELECT COUNT(comm) AS 'Commission Earners' FROM emp;
END //
DELIMITER ;
```

Calling the procedure :

```
> CALL comm_details();
```

2. Procedures with IN Parameters

Suppose we want to display the list of employees who are from a specified department.

```
DELIMITER //
CREATE PROCEDURE dept_employees (IN dno int)
BEGIN
        SELECT * from EMP where DEPTNO=dno;
END //
DELIMITER ;
```

Calling the procedure :

```
> CALL dept_employees(20);
```

3. Procedures with OUT Parameters

Suppose we want to display the highest salary paid.

```
DELIMITER //
CREATE PROCEDURE display_max_sal(OUT highestsal INT)
BEGIN
   SELECT MAX(sal) INTO highestsal FROM emp;
END //
DELIMITER ;
```

NOTE: When we call the procedure, the OUT parameter tells the database systems that its value goes out from the procedures. Now, we will pass its value to a session variable @M in the CALL statement as follows:

Calling the procedure :

```
> CALL display_max_sal (@X);
> SELECT @X;
```

4. Procedures with IN & OUT Parameters

Suppose we want to display the highest salary paid in the specified department

```
DELIMITER //
CREATE PROCEDURE display_deptmax_sal(OUT highestsal INT, IN dno INT)
BEGIN
   SELECT MAX(sal) INTO highestsal FROM emp WHERE deptno=dno;
END //
DELIMITER ;
```

Calling the procedure :

```
> CALL display_deptmax_sal (@M, 10);
```

6. Procedures with IN & OUT Parameters

Suppose we want to display the salary earned by a selected employee

```
DELIMITER &&
CREATE PROCEDURE get_emp_sal (INOUT data INT)
BEGIN
   SELECT sal INTO data FROM emp WHERE empno = data;
END &&
DELIMITER ;
```

Calling the procedure :

```
> SET @M = 7900
> CALL get_emp_sal(@M);
> SELECT @M;
```

Displaying the list of All procedures

When we have several procedures in the MySQL server, we can list all procedure stored on the current MySQL server as follows:

SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE search_condition]

Example:

SHOW PROCEDURE STATUS WHERE DB='IT2B';

Deleting Stored Procedures:

MySQL also allows a command to drop the procedure. When the procedure is dropped, it is removed from the database server also. The following statement is used to drop a stored procedure in MySQL:

DROP PROCEDURE [IF EXISTS] procedure_name;

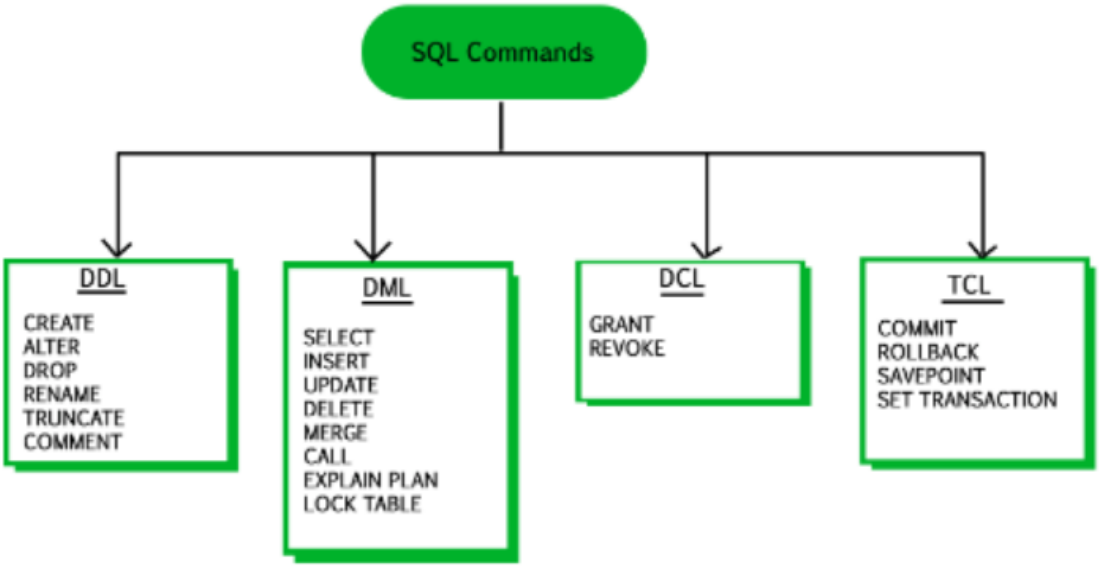
Example:

DROP PROCEDURE comm_details;

DATABASE MANAGEMENT SYSTEMS
UNIT V – TOPIC 3
Data Control Language

Data Controlling Language:

Data Controlling Language (DCL) helps users to retrieve and modify the data stored in the database with some specified queries. Grant and Revoke belong to these types of commands of the Data controlling Language. DCL is a component of SQL commands.



Two types of DCL commands can be used by the user in SQL. These commands are useful, especially when several users access the database. It enables the administrator to manage access control. The two types of DCL commands are as follows:

- GRANT
- REVOKE

Grant Privileges on Table

This command allows the administrator to assign particular privileges or permissions over a database object, such as a table, view, or procedure.

It enables system administrators to assign privileges and roles to the MySQL user accounts so that they can use the assigned permission on the database whenever required.

Syntax

```
GRANT privilege_name(s)
ON object
TO user_account_name;
```

Parameter	Description
privilege_name(s)	**It specifies the access rights or grant privilege to user accounts. If we want to give multiple privileges, then use a comma operator to separate them.**
object	**It determines the privilege level on which the access rights are being granted. It means granting privilege to the table; then the object should be the name of the table.**
user_account_name	**It determines the account name of the user to whom the access rights would be granted.**

The privileges to assign are listed below. It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
ALL	ALL does not revoke all permissions for the table. Rather, it revokes the ANSI-92 permissions which are SELECT, INSERT, UPDATE, DELETE, and REFERENCES.

Object

The name of the database objects that you are granting permissions for. In the case of granting privileges on a table, this would be the table name.

User_account_name

The name of the user who will be granted these privileges.

Example

Let's look at some examples of how to grant privileges on tables in SQL Server.

For example, if you wanted to grant SELECT, INSERT, UPDATE, and DELETE privileges on a table called employees to a user name smithj, you would run the following GRANT statement:

GRANT SELECT,INSERT,UPDATE, DELETE ON employees TO smithj;

You can also use the ALL keyword to indicate that you wish to grant the ANSI-92 permissions

(ie: SELECT, INSERT, UPDATE, DELETE, and REFERENCES) to a user named smithj.

```
GRANT ALL ON employees TO smithj;
```

If you wanted to grant only SELECT access on the employees table to all users, you could grant the privileges to the public role.

```
GRANT SELECT ON employees TO public;
```

Revoke Privileges on Table:

As the name suggests, revoke is to take away.

The REVOKE command enables the database administrator to remove the previously provided privileges or permissions from a user over a database or database object, such as a table, view, or procedure.

The REVOKE commands prevent the user from accessing or performing a specific operation on an element in the database.

In simple language, the REVOKE command terminates the ability of the user to perform the mentioned SQL command in the REVOKE query on the database or its component.

The primary reason for implementing the REVOKE query in the database is to ensure the data's security and integrity.

Syntax

The syntax for revoking privileges on a table in SQL Server is:

```
REVOKE privileges
ON object
FROM user_account_name;
```

Example

Let's look at some examples of how to revoke privileges on tables in SQL Server.

For example, if you wanted to revoke DELETE privileges on a table called employees from a user named anderson, you would run the following REVOKE statement:

```
REVOKE DELETE ON employees FROM anderson;
```

If you wanted to revoke ALL ANSI-92 permissions

(ie: SELECT, INSERT, UPDATE, DELETE, and REFERENCES) on a table for a user named anderson, you could use the ALL keyword as follows:

```
REVOKE ALL ON employees FROM anderson;
```

If you had granted SELECT privileges to the public role (ie: all users) on the employees table and you wanted to revoke these privileges, you could run the following REVOKE statement:

```
REVOKE SELECT ON employees FROM public;
```

Differences between Grant and Revoke commands:

S.NO	Grant	Revoke
1	This DCL command grants permissions to the user on the database objects.	This DCL command removes permissions if any granted to the users on database objects.
2	It assigns access rights to users.	It revokes the user access rights of users.
3	For each user you need to specify the permissions.	If access for one user is removed; all the particular permissions provided by that users to others will be removed.
4	When the access is decentralized granting permissions will be easy.	If decentralized access removing the granted permissions is difficult.

Advantages of DCL commands:

- **Security:** the primary reason to implement DCL commands in the database is to manage the access to the database and its object between different users. It ensures the security and integrity of the data stored in the database.
- **Granular control:** DCL commands provide granular control to the data administrator over the database. Thus, it enables the admin to create different levels of access to the database.
- **Flexibility:** The data administrator can implement DCL commands on specific commands and queries in the database. It allows the administrator to grant or revoke user permissions and privileges as per their needs. It provides flexibility to the administrator that allows them to manage access to the database.

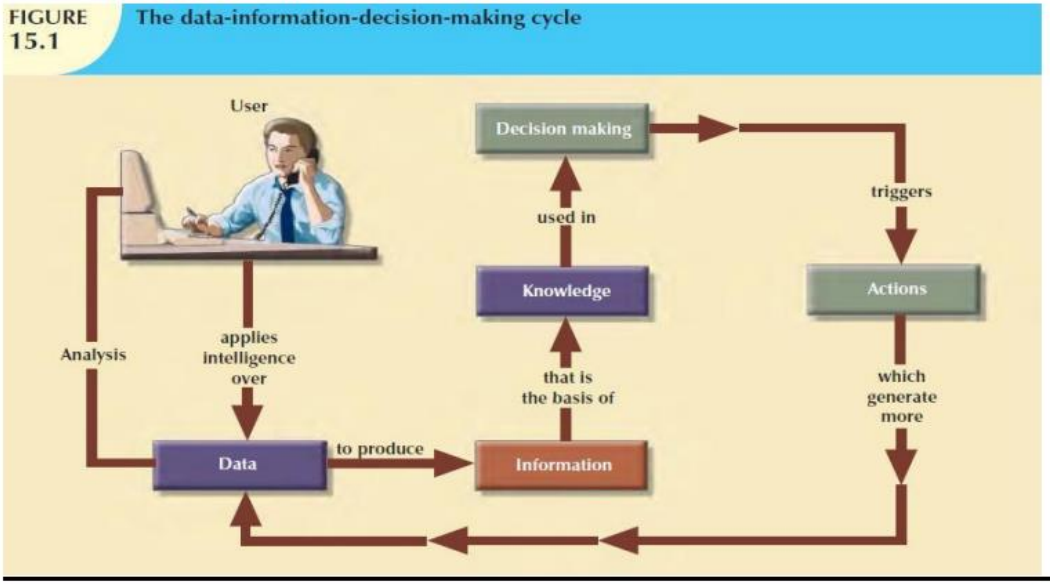
Disadvantages of DCL commands:

- **Complexity:** It increases the complexity of database management. If many users are accessing the database, keeping track of permission and privileges provided to every user in the database becomes very complex.
- **Time-Consuming:** It is time-consuming to assign the permissions and privileges to each user separately.
- **Risk of human error:** Human administrators execute DCL commands and can make mistakes in granting or revoking privileges. Thus, giving unauthorized access to data or imposing unintended restrictions on access.
- **Lack of audit trail:** There may be no built-in mechanism to track changes to privileges and permissions over time. Thus, it is extremely difficult to determine who has access to the data and when that access was granted or revoked.

DATABASE MANAGEMENT SYSTEMS
UNIT V – TOPIC 4
Data Base Administrator

Data as a Corporate Asset:

Data are a valuable resource that can translate into information. If the information is accurate and timely, it is likely to trigger actions that enhance the company's competitive position and generate wealth. In effect, an organization is subject to a data-information-decision cycle; that is, the data user applies intelligence to data to produce information that is the basis of knowledge used in decision making by the user. This cycle is illustrated in Figure below.



The decisions made by high-level managers trigger actions within the organization's lower levels. Such actions produce additional data to be used for monitoring company performance. Thus, data form the basis for decision making, strategic planning, control, and operations monitoring. To manage data as a corporate asset, managers must understand the value of information—that is, processed data.

Role of a Database in an Organization:

Data are used by different people in different departments for different reasons. Therefore, data management must address the concept of shared data. The DBMS facilitates:

- Interpretation and presentation of data in useful formats
- Distribution of data and information to the right people at the right time.
- Data preservation and monitoring the data usage for adequate periods of time.
- Control over data duplication and use, both internally and externally.
- Data Security and Concurrent access.

The database's predominant role is to support managerial decision making at all levels in the organization while preserving data privacy and security.

DataBase Administration Function:

The person responsible for the control of the centralized and shared database became known as the **DataBase Administrator (DBA).**

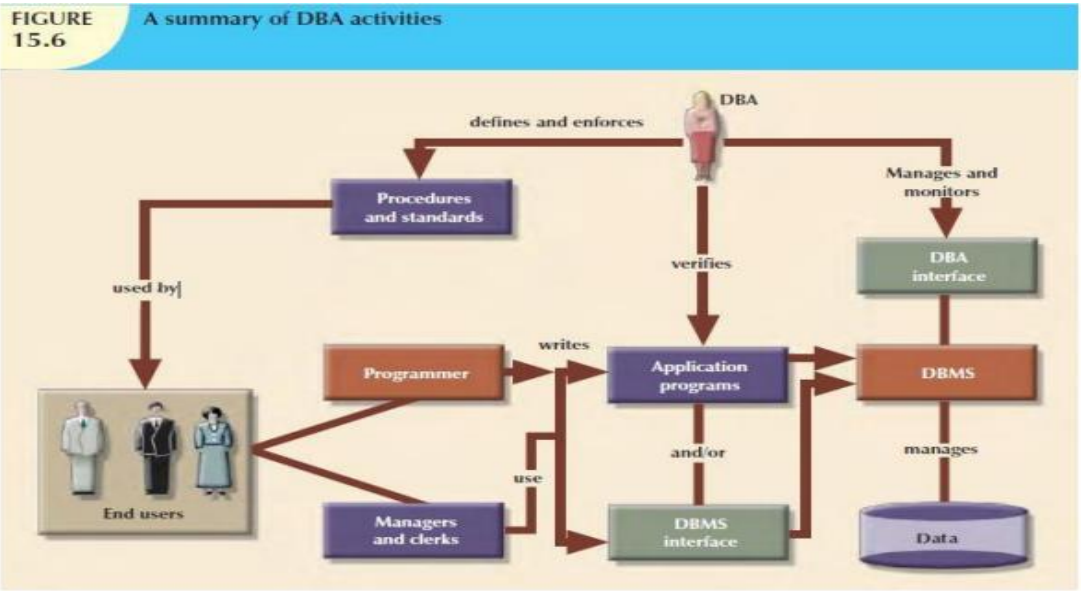
It is common practice to define the DBA function by dividing the DBA operations according to the Database Life Cycle (DBLC) phases. If that approach is used, the DBA function requires personnel to cover the following activities:

- Database planning, including the definition of standards, procedures, and enforcement.
- Database requirements gathering and conceptual design.
- Database logical and transaction design.
- Database physical design and implementation.
- Database testing and debugging.
- Database operations and maintenance, including installation, conversion, and migration.
- Database training and support.
- Data quality monitoring and management.

Figure below represents an appropriate DBA functional organization according to that model.

The interactions between the people and data in an organization, places the DBA in the dynamic environment as explored in the Figure below. The DBA is the focal point for data/user interaction. The DBA defines and enforces the procedures and standards to be used by programmers and end users during their work with the DBMS.

The DBA also verifies that programmer and end-user access meets the required quality and security standards.



DBA skills are be divided into two categories **Managerial Role**
Technical Role

The DBA's Managerial Role

As a manager, the DBA must concentrate on the control and planning dimensions of database administration. Therefore, the DBA is responsible for:

- Coordinating, monitoring, and allocating database administration resources: i.e. people and data.
- Defining goals and formulating strategic plans for the database administration function.

1. End-User Support

The DBA interacts with the end user by providing data and information support services to the organization's departments.

- Gathering user requirements.
- Building end-user confidence.
- Resolving conflicts and problems.
- Finding solutions to information needs.
- Ensuring quality and integrity of data and applications.
- Managing the training and support of DBMS users.

2. Policies, Procedures and Standards

A prime component of a successful data administration strategy is the continuous enforcement of the policies,procedures, and standards for correct data creation, usage, distribution, and deletion within the database.

- **Policies** are general statements of direction or action that communicate and support DBA goals.

- **Standards** describe the minimum requirements of a given DBA activity; they are more detailed and specificthan policies.

- **Procedures** are written instructions that describe a series of steps to be followed duringthe performance ofa given activity.

3. Data Security, Privacy and Integrity

The security, privacy, and integrity of the data in the database are of great concern to DBAs who manage current DBMS installations. The distribution of data across multiple sites, has made data maintenance, security, and integrity very critical. The DBAs must team up with Internet security experts to build security mechanisms to safeguard data from possible attacks or unauthorized access.

4. Data Backup and Recovery

The DBA must also ensure that the data in the database can be fully recovered in case of physical data loss or loss of database integrity. Data loss can be partial or total. A partial loss is caused by a physical loss of part of the database or when part of the database has lost integrity. A total loss might mean that the database continues to exist but its integrity is entirely lost or that the entire database is physically lost.

Disaster management includes all of the DBA activities designed to secure data availability following a physical disaster or a database integrity failure.

The DBA's Technical Role

The DBA's technical activities include the selection, installation, operation, maintenance, and upgrading of the DBMS and utility software, as well as the design, development, implementation, and maintenance of the application programs that interact with the database.

The technical aspects of the DBA's job are rooted in the following areas of operation:

1. **Evaluating, selecting, and installing the DBMS and related utilities**.
   Selecting the database management system, utility software, and supporting hardware to be used in the organization. The selection plan is based on organization's needs and the features like DBMS model, storage capacity, backup-recovery, concurrency control, performance, portability, cost etc.

2. **Designing and implementing databases and applications.**

   The DBA function usually requires that several people be dedicated to database modeling and design activities. The DBA also works with applications programmers to ensure the quality and integrity of

database design and transactions. Such support services include reviewing the database application design to ensure that transactions are correct, efficient and compliant.

3. Testing and evaluating databases and applications.

The DBA must also provide testing and evaluation services for all of the database and enduser applications. Testing starts with the loading of the tested database. That database contains test data for the applications, and its purpose is to check the data definition and integrity rules of the database and application programs. The testing and evaluation of a database application cover all aspects of the system technical, evaluation of written documentation, observance of standards for naming, documenting, and coding, Data duplication conflicts with existing data, the enforcement of all data validation rules.

4. Operating the DBMS, utilities, and applications.

DBMS operations can be divided into four main areas:

- System support.
- Performance monitoring and tuning.
- Backup and recovery.
- Security auditing and monitoring.

5. Training and supporting users.

Training people to use the DBMS and its tools is included in the DBA's technical activities. The DBA provides technical training in the use of the DBMS and its utilities for the applications programmers.

6. Maintaining the DBMS, utilities, and applications.

Maintenance activities are dedicated to the preservation of the DBMS environment. Periodic DBMS maintenance includes management of the physical or secondary storage devices, upgrading the DBMS and utility software, migration and conversion services for data in incompatible formats or for different DBMS software.