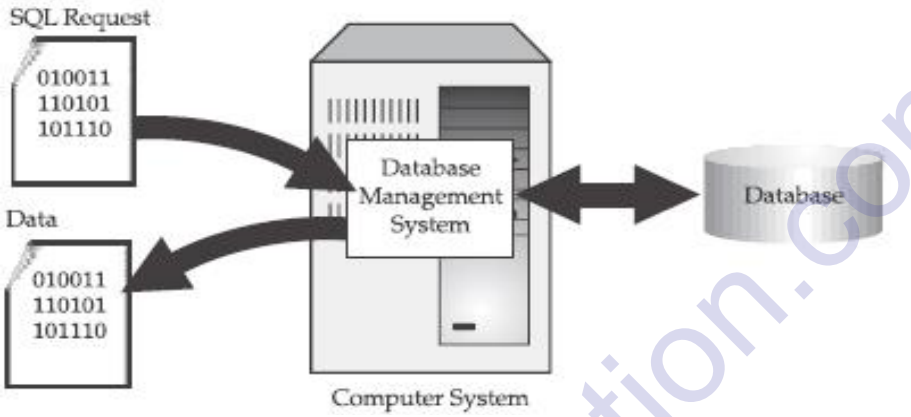
DBMS is a software package that carries out many different tasks including the provision of facilities to enable the user to access and modify information in the database. The database is an intermediate link between the physical database, computer and the operating system and the users. To provide the various facilities to different types of users, a DBMS normally provides one or more specialized programming languages called database languages



The DBMS must provide appropriate languages and interfaces for each category of users to express database queries and updates. Database Languages are used to create and maintain database on computer. There are large numbers of database languages like Oracle, MySQL, MS Access, dBase, FoxPro etc. SQL statements commonly used in Oracle and MS Access can be categorized as data definition language (DDL), data control language (DCL) and data manipulation language (DML).

Data Definition Language (DDL)

It is a language that allows the users to define data and their relationship to other types of data. It is mainly used to create files, databases, data dictionary and tables within databases. It is also used to specify the structure of each table, set of associated values with each attribute, integrity constraints, security and authorization information for each table and physical storage structure of each table on disk. The following table gives an overview about usage of DDL statements in SQL

S.No	Need and Usage	The SQL DDL statement
1	Create schema objects	CREATE
2	Alter schema objects	ALTER
3	Delete schema objects	DROP
4	Reneme schema objects	RENAME

Data Manipulation Language (DML):

It is a language that provides a set of operations to support the basic data manipulation operations on the data held in the databases. It allows users to insert, update, delete and retrieve

data from the database. The part of DML that involves data retrieval is called a query language. The following table gives an overview about the usage of DML statements in SQL:

S. No	Need and Usage	The SQL DML statement
1	Remove rows from tables or views	DELETE
2	Add new rows of data into table or view	INSERT
3	Retrieve data from one or more tables	SELECT
4	change column values In existing rows of a table or view	UPDATE

DML command

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

---

1) INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

INSERT into *table-name* values(data1,data2,..)

Lets see an example,

Consider a table **Student** with following fields.

S_id S_Name age

INSERT into Student values(101,'Adam',15);

The above command will insert a record into **Student** table.

S_id S_Name age

101 Adam 15

Example to Insert Null value to a column

Both the statements below will insert NULL value into **age** column of the Student table.

INSERT into Student(id,name) values(102,'Alex');

Or,

INSERT into Student values(102,'Alex',null);

The above command will insert only two column value other column is set to null.

S_id S_Name age

101 Adam 15

102 Alex

Example to Insert Default value to a column

INSERT into Student values(103,'Chris',default)

S_id S_Name age

101 Adam 15

102 Alex

103 chris 14

Suppose the **age** column of student table has default value of 14.
Also, if you run the below query, it will insert default value into the age column, whatever the default value may be.
INSERT into Student values(103,'Chris')

---

2) UPDATE command
Update command is used to update a row of a table. Following is its general syntax,
UPDATE *table-name* set column-name = value *where* **condition**;
Lets see an example,
update Student set age=18 where s_id=102;

S_id S_Name age

101 Adam 15

102 Alex 18

103 chris 14

Example to Update multiple columns
UPDATE Student set s_name='Abhi',age=17 where s_id=103;
The above command will update two columns of a record.

S_id S_Name age

101 Adam 15

102 Alex 18

103 Abhi 17

---

3) DELETE command
Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. Following is its general syntax,
DELETE from *table-name*;

Example to Delete all Records from a Table
DELETE from Student;
The above command will delete all the records from **Student** table.

Example to Delete a particular Record from a Table
Consider the following **Student** table

S_id S_Name age

101 Adam 15

102 Alex 18

103 Abhi 17

DELETE from Student where s_id=103;
The above command will delete the record where s_id is 103 from **Student** table.

S_id S_Name age

101 Adam 15
102 Alex 18

Data Control Language (DCL):

DCL statements control access to data and the database using statements such as GRANT and REVOKE. A privilege can either be granted to a User with the help of GRANT statement. The privileges assigned can be SELECT, ALTER, DELETE, EXECUTE, INSERT, INDEX etc. In addition to granting of privileges, you can also revoke (taken back) it by using REVOKE command. The following table gives an overview about the usage of DCL statements in SQL:

S. No.	Need And Usage	Age
1	Grant and take away priviliges and roles	Grant Revoke
2	Add a comment to the data dictionary	Comment

In practice, the data definition and data manipulation languages are not two separate languages. Instead, they simply form parts of a single database language such as Structured Query Language (SQL). SQL represents combination of DDL and DML, as well as statements for constraints specification and schema evaluation. Data Control Language (DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges. Privileges are of two types,

- **System :** creating session, table etc are all types of system privilege.
- **Object :** any command or query to work on tables comes under object privilege.
- **Grant :** Gives user access privileges to database.
- **Revoke :** Take back permissions from user.

To Allow a User to create Session
grant create session to *username*;

To Allow a User to create Table
grant create table to *username*;

To provide User with some Space on Tablespace to store Table
alter user *username* quota unlimited on system;

To Grant all privilege to a User
grant sysdba to *username*

To Grant permission to Create any Table
grant *create* any table to *username*

To Grant permission to Drop any Table
grant *drop* any table to *username*

To take back Permissions

revoke create table from *username*

Database Access for applications Programs

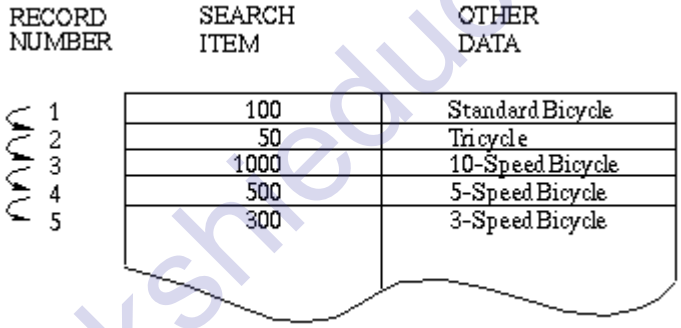
Data Access

Database access and manipulation is performed using the data manipulation statements. These statements, which are specifically designed to interact with an Eloquence database, are invoked through Eloquence language programs. These statements are structured so that each one suggests its function (for example, DBGET gets data from a data set). All data access is carried out at the data entry level (this is known as the "full record mode"). Data entries may be accessed in one of five modes: *serial, directed, chained, indexed* or *calculated*.

Serial Access

When accessing a data set in serial mode, Eloquence DBMS starts at the most recently accessed record (data entry), called the *current record* and sequentially examines records until the next, non-empty record is located. This record is then transferred to the data buffer and becomes the new current record. Serial access is often used to examine or list all entries in a data set.

The following example shows entries in the PRODUCT master data set. The record numbers are shown to the left of each entry. The arrows to the left of the record number show how entries will be retrieved in serial mode. If the current record is 4, for example, the next record accessed in serial mode will be record number 5.

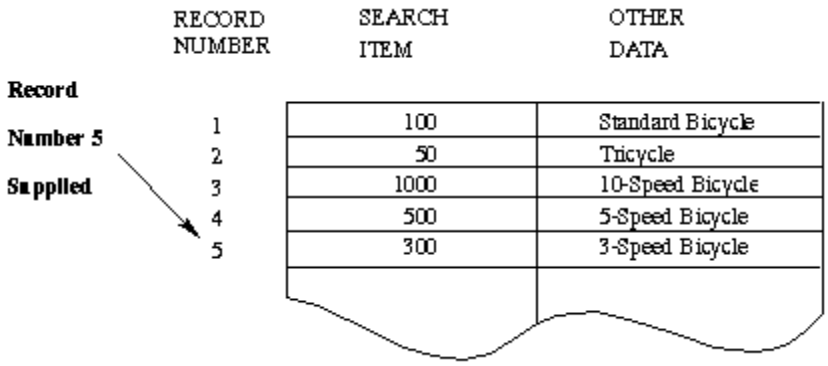


A Serial Access of the PRODUCT Master Data Set

Directed Access

A second method of accessing a data entry is directed access. With this method, Eloquence DBMS returns the record specified by a record number supplied by a program. If the specified record is non-empty the record is transferred to the data buffer. If the record is empty a status error is returned. In either case, the current record is set to the record specified. Directed access is used to read entries following a SORT or FIND operation.

The following example shows the retrieval of an entry using directed access. The record number 5, supplied by an application program, instructs Eloquence DBMS to retrieve record 5. Eloquence DBMS then copies the record into the data buffer and resets the current record to 5.



Directed Access of the PRODUCT Master Data Set

Chained Access

Chained access is used to retrieve detail data entries with common search item values. Eloquence DBMS supports chained access in a forward direction. Entries along a data chain may be accessed in a reverse direction, however, by using directed access and the status information returned by Eloquence DBMS. Chained access of detail data sets is often used for retrieving information about related events.

The following example shows the retrieval of detail entries using chained access. The corresponding chain pointer information, maintained by Eloquence DBMS, is shown along with the record number for the data set. Eloquence DBMS uses this pointer information to retrieve the next entry along the chain. The arrows to the left of the record numbers show how entries will be retrieved in chained mode. If the current record is 5, for example, the next record accessed in chained mode will be 7.

CUSTOMER (detail) Data Set

RECORD NUMBER	FORWARD CHAIN POINTER	BACKWARD CHAIN POINTER	SEARCH ITEM (PRODUCT-NO)	OTHER DATA (NAME)
1	0	0	100	Jimmy Dailing
2	5	0	50	Malcomb Gissing
3	0	0	500	Barton Decker
4	6	0	300	Sean Houseman
5	7	2	50	Sam Johnson
6	0	4	300	Bart Bekker
7	0	5	50	Thomas Smith

Figure 5 Chained Access of the CUSTOMER Detail Data Set

Calculated Access

Calculated access is based on a search item value, and may be used to access master data sets only. This access method involves assigning a search item to a data set, where the assignment of search item to data set is carried out by the ISAM software using index files.

The following example shows the retrieval of an entry using calculated access. The search item value 500, which is supplied by an application program, is used by Eloquence DBMS to locate record number 5. The record is copied into the data buffer and the current record is set to 5.
This example also shows two deleted (blank) records. When a record is deleted from a data file, the space taken by that record still remains in the file. When a new record is added, it takes an available space left by a deleted record. If there are no deleted (blank) records, the new record is added to the end of the file.

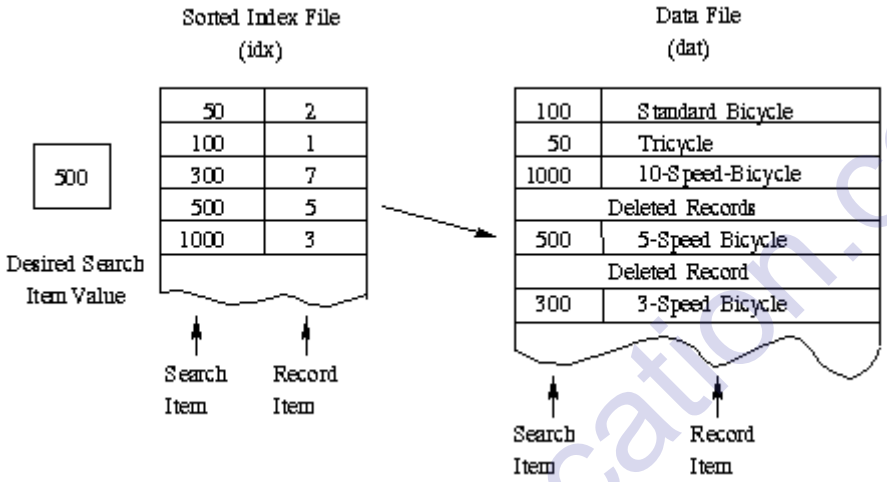
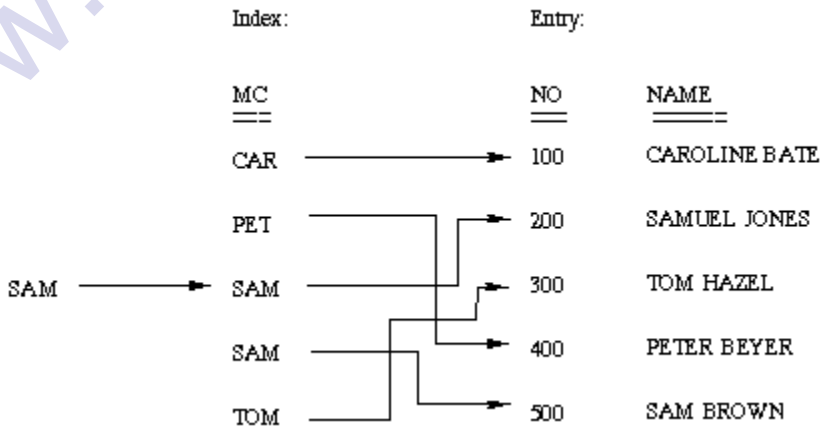


Figure 6 Calculated Access of the PRODUCT Master Data Set

Indexed Access

Indexed access is a further method of accessing individual data set entries or a group of data set entries. It is similar to chained access, but there are differences:

- No master set is required, so indexed access can be used for all data set types.
- Combined search-items are possible (e.g. the first four characters of a name plus date of birth). They don't have to be unique.
- Access via just part of the search-item is possible (e.g., all customers whose names begin with SAM).
- Data can be accessed in sorted form. By default, the index item values are sorted in ASCII sequence, but other collating sequences can be specified at the time the database is created.



Indexed Access

Database Users

Database users are the one who really use and take the benefits of database. There will be different types of users depending on their need and way of accessing the database.

1. **Application Programmers -** They are the developers who interact with the database by means of DML queries. These DML queries are written in the application programs like C, C++, JAVA, Pascal etc. These queries are converted into object code to communicate with the database. For example, writing a C program to generate the report of employees who are working in particular department will involve a query to fetch the data from database. It will include a embedded SQL query in the C Program.
2. **Sophisticated Users -** They are database developers, who write SQL queries to select/insert/delete/update data. They do not use any application or programs to request the database. They directly interact with the database by means of query language like SQL. These users will be scientists, engineers, analysts who thoroughly study SQL and DBMS to apply the concepts in their requirement. In short, we can say this category includes designers and developers of DBMS and SQL.
3. **Specialized Users -** These are also sophisticated users, but they write special database application programs. They are the developers who develop the complex programs to the requirement.
4. **Stand-alone Users -** These users will have stand –alone database for their personal use. These kinds of database will have readymade database packages which will have menus and graphical interfaces.
5. **Native Users -** these are the users who use the existing application to interact with the database. For example, online library system, ticket booking systems, ATMs etc which has existing application and users use them to interact with the database to fulfill their requests.

Database Administrators

The life cycle of database starts from designing, implementing to administration of it. A database for any kind of requirement needs to be designed perfectly so that it should work without any issues. Once all the design is complete, it needs to be installed. Once this step is complete, users start using the database. The database grows as the data grows in the database. When the database becomes huge, its performance comes down. Also accessing the data from the database becomes challenge. There will be unused memory in database, making the memory inevitably huge. These administration and maintenance of database is taken care by Data Base Administrator(DBA).

A DBA has many responsibilities. A good performing database is in the hands of DBA.

- **Installing and upgrading the DBMS Servers: -** DBA is responsible for installing a new DBMS server for the new projects. He is also responsible for upgrading these servers as there are new versions comes in the market or requirement. If there is any failure in upgradation of the existing servers, he should be able revert the new changes back to the older version, thus maintaining the DBMS working. He is also responsible for updating the service packs/ hot fixes/ patches to the DBMS servers.
- **Design and implementation: -** Designing the database and implementing is also DBA's responsibility. He should be able to decide proper memory management, file organizations, error handling, log maintenance etc for the database.

- **Performance tuning: -** Since database is huge and it will have lots of tables, data, constraints and indices, there will be variations in the performance from time to time. Also, because of some designing issues or data growth, the database will not work as expected. It is responsibility of the DBA to tune the database performance. He is responsible to make sure all the queries and programs works in fraction of seconds.
- **Migrate database servers: -** Sometimes, users using oracle would like to shift to SQL server or Netezza. It is the responsibility of DBA to make sure that migration happens without any failure, and there is no data loss.
- **Backup and Recovery: -** Proper backup and recovery programs needs to be developed by DBA and has to be maintained him. This is one of the main responsibilities of DBA. Data/objects should be backed up regularly so that if there is any crash, it should be recovered without much effort and data loss.
- **Security: -** DBA is responsible for creating various database users and roles, and giving them different levels of access rights.
- **Documentation: -** DBA should be properly documenting all his activities so that if he quits or any new DBA comes in, he should be able to understand the database without any effort. He should basically maintain all his installation, backup, recovery, security methods. He should keep various reports about database performance.

Types of DBA

There are different kinds of DBA depending on the responsibility that he owns.

- **Administrative DBA -** This DBA is mainly concerned with installing, and maintaining DBMS servers. His prime tasks are installing, backups, recovery, security, replications, memory management, configurations and tuning. He is mainly responsible for all administrative tasks of a database.
- **Development DBA -** He is responsible for creating queries and procedure for the requirement. Basically his task is similar to any database developer.
- **Database Architect -** Database architect is responsible for creating and maintaining the users, roles, access rights, tables, views, constraints and indexes. He is mainly responsible for designing the structure of the database depending on the requirement. These structures will be used by developers and development DBA to code.
- **Data Warehouse DBA -**DBA should be able to maintain the data and procedures from various sources in the datawarehouse. These sources can be files, COBOL, or any other programs. Here data and programs will be from different sources. A good DBA should be able to keep the performance and function levels from these sources at same pace to make the datawarehouse to work.
- **Application DBA -**He acts like a bridge between the application program and the database. He makes sure all the application program is optimized to interact with the database. He ensures all the activities from installing, upgrading, and patching, maintaining, backup, recovery to executing the records works without any issues.
- **OLAP DBA -** He is responsible for installing and maintaining the database in OLAP systems. He maintains only OLAP databases.

Transaction Management:

Every transaction, for whatever purpose it is being used, has the following four properties. Taking the initial letters of these four properties we collectively call them the ACID Properties**.** Here we try to describe them and explain them.

Atomicity: This means that either all of the instructions within the transaction will be reflected in the database, or none of them will be reflected.
Say for example, we have two accounts A and B, each containing Rs 1000/-. We now start a transaction to deposit Rs 100/- from account A to Account B.

```
ReadA;
A=A–100;
WriteA;
ReadB;
B=B+100;
Write B;
```

- Fine, is not it? The transaction has 6 instructions to extract the amount from A and submit it to B. The AFIM will show Rs 900/- in A and Rs 1100/- in B.
- Now, suppose there is a power failure just after instruction 3 (Write A) has been complete. What happens now? After the system recovers the AFIM will show Rs 900/- in A, but the same Rs 1000/- in B. It would be said that Rs 100/- evaporated in thin air for the power failure. Clearly such a situation is not acceptable.
- The solution is to keep every value calculated by the instruction of the transaction not in any stable storage (hard disc) but in a volatile storage (RAM), until the transaction completes its last instruction.
- When we see that there has not been any error we do something known as a COMMIT operation. Its job is to write every temporarily calculated value from the volatile storage on to the stable storage. In this way, even if power fails at instruction 3, the post recovery image of the database will show accounts A and B both containing Rs 1000/-, as if the failed transaction had never occurred.

Consistency: If we execute a particular transaction in isolation or together with other transaction, (i.e. presumably in a multi-programming environment), the transaction will yield the same expected result.
To give better performance, every database management system supports the execution of multiple transactions at the same time, using CPU Time Sharing. Concurrently executing transactions may have to deal with the problem of sharable resources, i.e. resources that multiple transactions are trying to read/write at the same time.
For example, we may have a table or a record on which two transaction are trying to read or write at the same time. Careful mechanisms are created in order to prevent mismanagement of these sharable resources, so that there should not be any change in the way a transaction performs. A transaction which deposits Rs 100/- to account A must deposit the same amount whether it is acting alone or in conjunction with another transaction that may be trying to deposit or withdraw some amount at the same time.

Isolation: In case multiple transactions are executing concurrently and trying to access a sharable resource at the same time, the system should create an ordering in their execution so that they should not create any anomaly in the value stored at the sharable resource. There are several ways to achieve this and the most popular one is using some kind of locking mechanism. Again, if you have the concept of Operating Systems, then you should remember the semaphores, how it is used by a process to make a resource busy before starting to use it, and how it is used to release the resource after the usage is over.
Other processes intending to access that same resource must wait during this time. Locking is almost similar. It states that a transaction must first lock the data item that it wishes to access, and release the lock when the accessing is no longer required. Once a transaction locks the data item, other transactions wishing to access the same data item must wait until the lock is released.

Durability: It states that once a transaction has been complete the changes it has made should be permanent.
As we have seen in the explanation of the Atomicity property, the transaction, if completes successfully, is committed. Once the COMMIT is done, the changes which the transaction has made to the database are immediately written into permanent storage. So, after the transaction has been committed successfully, there is no question of any loss of information even if the power fails.
Committing a transaction guarantees that the AFIM has been reached.There are several ways Atomicity and Durability can be implemented. One of them is called **Shadow Copy**. In this scheme a database pointer is used to point to the BFIM of the database. During the transaction, all the temporary changes are recorded into a Shadow Copy, which is an exact copy of the original database plus the changes made by the transaction, which is the AFIM. Now, if the transaction is required to COMMIT, then the database pointer is updated to point to the AFIM copy, and the BFIM copy is discarded.
On the other hand, if the transaction is not committed, then the database pointer is not updated. It keeps pointing to the BFIM, and the AFIM is discarded. This is a simple scheme, but takes a lot of memory space and time to implement. If you study carefully, you can understand that Atomicity and Durability is essentially the same thing, just as Consistency and Isolation is essentially the same thing.

Transaction States: There are the following six states in which a transaction may exist

Active: The initial state when the transaction has just started execution.

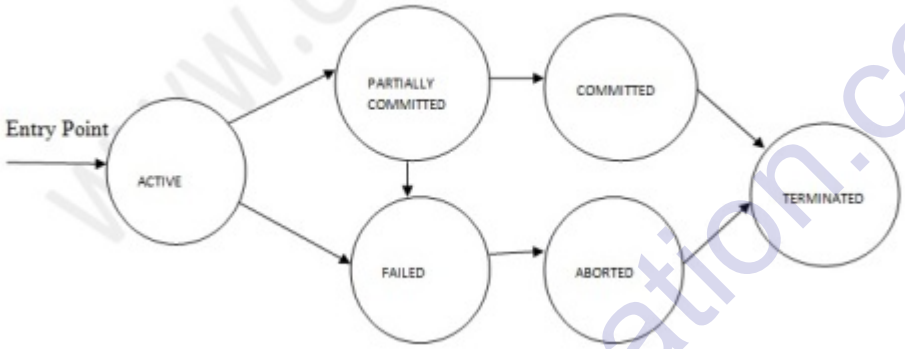
Partially Committed: At any given point of time if the transaction is executing properly, then it is going towards it COMMIT POINT. The values generated during the execution are all stored in volatile storage.

Failed: If the transaction fails for some reason. The temporary values are no longer required, and the transaction is set to **ROLLBACK**. It means that any change made to the database by this transaction up to the point of the failure must be undone. If the failed transaction has withdrawn Rs. 100/- from account A, then the ROLLBACK operation should add Rs 100/- to account A.

Aborted: When the ROLLBACK operation is over, the database reaches the BFIM. The transaction is now said to have been aborted.

Committed: If no failure occurs then the transaction reaches the COMMIT POINT. All the temporary values are written to the stable storage and the transaction is said to have been committed.

Terminated: Either committed or aborted, the transaction finally reaches this state.
The whole process can be described using the following diagram:



Concurrent Execution
A schedule is a collection of many transactions which is implemented as a unit. Depending upon how these transactions are arranged in within a schedule, a schedule can be of two types:

- **Serial:** The transactions are executed one after another, in a non-preemptive manner.
- **Concurrent:** The transactions are executed in a preemptive, time shared method.

In Serial schedule, there is no question of sharing a single data item among many transactions, because not more than a single transaction is executing at any point of time. However, a serial schedule is inefficient in the sense that the transactions suffer for having a longer waiting time and response time, as well as low amount of resource utilization.
In concurrent schedule, CPU time is shared among two or more transactions in order to run them concurrently. However, this creates the possibility that more than one transaction may need to access a single data item for read/write purpose and the database could contain inconsistent value if such accesses are not handled properly. Let us explain with the help of an example.
Let us consider there are two transactions T1 and T2, whose instruction sets are given as following. T1 is the same as we have seen earlier, while T2 is a new transaction.

T1
ReadA;
A=A–100;
WriteA;
ReadB;
B=B+100;
Write B;

T2
ReadA;
Temp=A*0.1;
ReadC;
C=C+Temp;
WriteC;

T2 is a new transaction which deposits to account C 10% of the amount in account A. If we prepare a serial schedule, then either T1 will completely finish before T2 can begin, or T2 will completely finish before T1 can begin. However, if we want to create a concurrent schedule, then some Context Switching need to be made, so that some portion of T1 will be executed, then some portion of T2 will be executed and so on. For example say we have prepared the following concurrent schedule.

T1	T2
Read A;	
A = A - 100;	
Write A;	
	Read A;
	Temp = A * 0.1;
	Read C;
	C = C + Temp;
	Write C;
Read B;	
B = B + 100;	
Write B;	

No problem here. We have made some Context Switching in this Schedule, the first one after executing the third instruction of T1, and after executing the last statement of T2. T1 first deducts Rs 100/- from A and writes the new value of Rs 900/- into A. T2 reads the value of A, calculates the value of Temp to be Rs 90/- and adds the value to C. The remaining part of T1 is executed and Rs 100/- is added to B.
It is clear that a proper Context Switching is very important in order to maintain the Consistency and Isolation properties of the transactions. But let us take another example where a wrong Context Switching can bring about disaster. Consider the following example involving the same

T1 and T2

T1	T2
Read A;	
A = A - 100;	
	Read A;
	Temp = A * 0.1;
	Read C;
	C = C + Temp;
	Write C;
Write A;	
Read B;	
B = B + 100;	
Write B;	

This schedule is wrong, because we have made the switching at the second instruction of T1. The result is very confusing. If we consider accounts A and B both containing Rs 1000/- each, then the result of this schedule should have left Rs 900/- in A, Rs 1100/- in B and add Rs 90 in C (as C should be increased by 10% of the amount in A). But in this wrong schedule, the Context Switching is being performed before the new value of Rs 900/- has been updated in A. T2 reads the old value of A, which is still Rs 1000/-, and deposits Rs 100/- in C. C makes an unjust gain of Rs 10/- out of nowhere.

In the above example, we detected the error simple by examining the schedule and applying common sense. But there must be some well formed rules regarding how to arrange instructions of the transactions to create error free concurrent schedules. This brings us to our next topic, the concept of Serializability.

Serializability

When several concurrent transactions are trying to access the same data item, the instructions within these concurrent transactions must be ordered in some way so as there are no problem in accessing and releasing the shared data item. There are two aspects of serializability which are described here:

- **Conflict Serializability**

  Two instructions of two different transactions may want to access the same data item in order to perform a read/write operation. Conflict Serializability deals with detecting whether the instructions are conflicting in any way, and specifying the order in which these two instructions will be executed in case there is any conflict. A conflict arises if at least one (or both) of the instructions is a write operation. The following rules are important in Conflict Serializability:

  1. If two instructions of the two concurrent transactions are both for read operation, then they are not in conflict, and can be allowed to take place in any order.
  2. If one of the instructions wants to perform a read operation and the other instruction wants to perform a write operation, then they are in conflict, hence their ordering is important. If the read instruction is performed first, then it reads the old value of the data item and after the reading is over, the new value of the data item is written. It the write instruction is performed first, then updates the data item with the new value and the read instruction reads the newly updated value.

3. If both the transactions are for write operation, then they are in conflict but can be allowed to take place in any order, because the transaction do not read the value updated by each other. However, the value that persists in the data item after the schedule is over is the one written by the instruction that performed the last write.

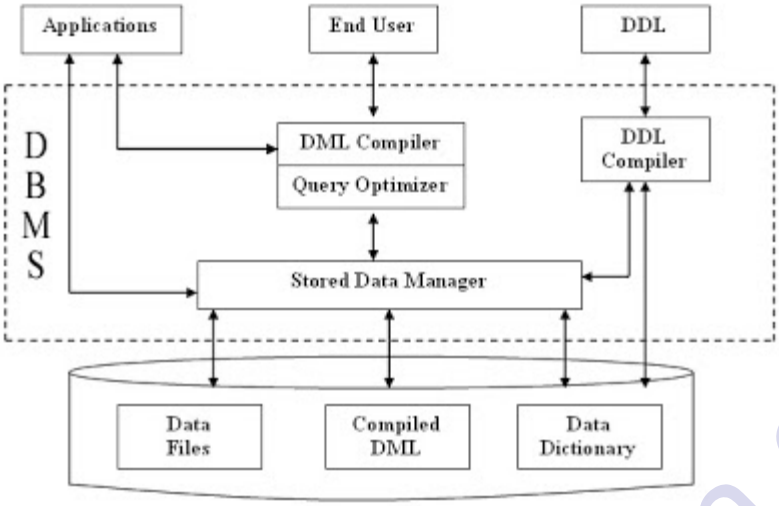
It may happen that we may want to execute the same set of transaction in a different schedule on another day. Keeping in mind these rules, we may sometimes alter parts of one schedule (S1) to create another schedule (S2) by swapping only the non-conflicting parts of the first schedule. The conflicting parts cannot be swapped in this way because the ordering of the conflicting instructions is important and cannot be changed in any other schedule that is derived from the first. If these two schedules are made of the same set of transactions, then both S1 and S2 would yield the same result if the conflict resolution rules are maintained while creating the new schedule. In that case the schedule S1 and S2 would be called **Conflict Equivalent**.

- **View Serializability:**
  This is another type of serializability that can be derived by creating another schedule out of an existing schedule, involving the same set of transactions. These two schedules would be called View Serializable if the following rules are followed while creating the second schedule out of the first. Let us consider that the transactions T1 and T2 are being serialized to create two different schedules S1 and S2 which we want to be **View Equivalent** and both T1 and T2 wants to access the same data item.
  1. If in S1, T1 reads the initial value of the data item, then in S2 also, T1 should read the initial value of that same data item.
  2. If in S1, T1 writes a value in the data item which is read by T2, then in S2 also, T1 should write the value in the data item before T2 reads it.
  3. If in S1, T1 performs the final write operation on that data item, then in S2 also, T1 should perform the final write operation on that data item.

Except in these three cases, any alteration can be possible while creating S2 by modifying S1.

Structure of DBMS

DBMS (Database Management System) acts as an interface between the user and the database. The user requests the DBMS to perform various operations (insert, delete, update and retrieval) on the database. The components of DBMS perform these requested operations on the database and provide necessary data to the users. The various components of DBMS are shown below: -



Structure Of DBMS

1. DDL Compiler - Data Description Language compiler processes schema definitions specified in the DDL. It includes metadata information such as the name of the files, data items, storage details of each file, mapping information and constraints etc.

2. DML Compiler and Query optimizer - The DML commands such as insert, update, delete, retrieve from the application program are sent to the DML compiler for compilation into object code for database access. The object code is then optimized in the best way to execute a query by the query optimizer and then send to the data manager.

3. Data Manager - The Data Manager is the central software component of the DBMS also knows as Database Control System.

The Main Functions Of Data Manager Are: –

• Convert operations in user's Queries coming from the application programs or combination of DML Compiler and Query optimizer which is known as Query Processor from user's logical view to physical file system.
• Controls DBMS information access that is stored on disk.
• It also controls handling buffers in main memory.
• It also enforces constraints to maintain consistency and integrity of the data.
• It also synchronizes the simultaneous operations performed by the concurrent users.
• It also controls the backup and recovery operations.

4. Data Dictionary - Data Dictionary is a repository of description of data in the database. It contains information about

• Data - names of the tables, names of attributes of each table, length of attributes, and number of rows in each table.
• Relationships between database transactions and data items referenced by them which is useful in determining which transactions are affected when certain data definitions are changed.
• Constraints on data i.e. range of values permitted.
• Detailed information on physical database design such as storage structure, access paths, files and record sizes.
• Access Authorization - is the Description of database users their responsibilities and their access rights.
• Usage statistics such as frequency of query and transactions.

Data dictionary is used to actually control the data integrity, database operation and accuracy. It may be used as a important part of the DBMS.

Importance of Data Dictionary -

Data Dictionary is necessary in the databases due to following reasons:

• It improves the control of DBA over the information system and user's understanding of use of the system.

• It helps in documentating the database design process by storing documentation of the result of every design phase and design decisions.

• It helps in searching the views on the database definitions of those views.

• It provides great assistance in producing a report of which data elements (i.e. data values) are used in all the programs.

• It promotes data independence i.e. by addition or modifications of structures in the database application program are not effected.

5. Data Files - It contains the data portion of the database.

6. Compiled DML - The DML complier converts the high level Queries into low level file access commands known as compiled DML.

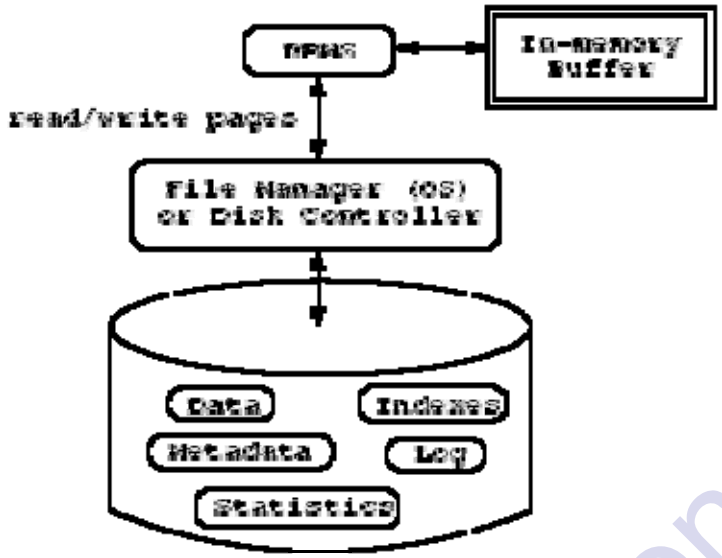
7. End Users - They are already discussed in previous section.

Storage management:

Storage management is important to the performance of DBMS. This paper gives a comprehensive overview of storage management in relational DBMS. The storage management is divided into three levels (logical, physical in-memory, and physical on-disk) and discussed separately.

The logical level caches logical units (tuples or index values) of the DBMS based on the logical information to improve buffer pool usage efficiency. The physical in-memory level caches physical pages directly in the buffer pool to reduce disk accesses. Many algorithms have been designed for the general buffer or for the buffer pool of DBMS. Complex algorithms can achieve better performance than simple LRU or CLOCK algorithms. However, the overhead is high and the advantage diminishes when the buffer is big.

The physical on-disk level borrows many techniques from file systems and storage systems. Because of the excellent potential write performance of log-structured organization, the Log-structured File System and related topics are discussed. Each level has its advantage and limitation for further improving performance. To achieve better performance of storage management systems of DBMS, all these three levels should be considered.



Query processing:

1. The retrieval of information from a database according to a set of retrieval criteria, the database itself remaining unchanged.
2. In the context of a specific query language, the technique of translating the retrieval criteria specified using the language into more primitive database-access software, including a selection among different methods to choose the most efficient in the particular circumstances
3. The scanning, parsing, and validating module produces an internal representation of the query. The query optimizer module devises an execution plan which is the execution strategy to retrieve the result of the query from the database files. A query typically has many possible execution strategies differing in performance, and the process of choosing a reasonably efficient one is known as query optimization.
4. The code generator generates the code to execute the plan.
5. The runtime database processor runs the generated code to produce the query result.

– SQL query is first translated into an equivalent extended relational algebra expression.

– SQL queries are decomposed into query blocks, which form the basic units that can be translated into the algebraic operators and optimized.

– Query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clauses.

– Nested queries within a query are identified as separate query blocks

Translating SQL Queries into Relational Algebra

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SALARY > (SELECT MAX(SALARY)
FROM EMPLOYEE
WHERE DNO=5);
```

• The inner block

(SELECT MAX (SALARY) FROM EMPLOYEE WHERE DNO=5)

– Translated in:

MAX SALARY (σDNO=5(EMPLOYEE))

• The Outer block

SELECT LNAME, FNAME FROM EMPLOYEE WHERE SALARY > C

– Translated in:

∏LNAME, FNAME (σSALARY>C(EMPLOYEE))

(C represents the result returned from the inner block.)

• The query optimizer would then choose an execution plan for each block.

• The inner block needs to be evaluated only once. (Uncorrelated nested query).