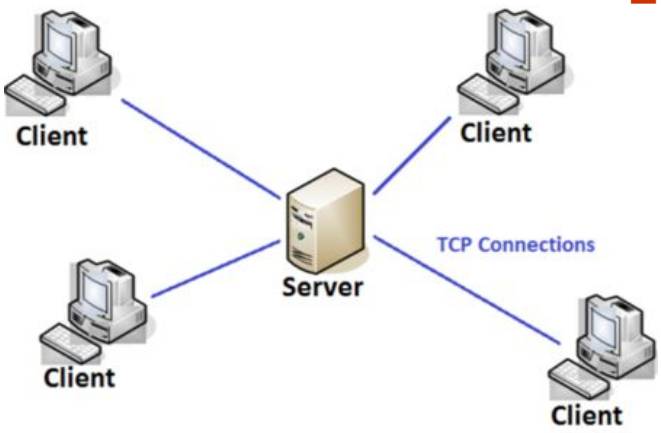
Chapter 2: SMBD ARCHITECTURE updated: Sonia Ordóñez

Database System Concepts, 6th Ed.

SMBD ARCHITECTURE

- Database management systems have been growing along with computers.
- Many of the developments in computers are due to data management needs.
- Many of the more sophisticated software developments (today they are critical) were developed due to limitations in hardware.
- The architecture of the SMBD, is made up of components in hardware and software that together are seen as a great information system.
- The SMBD, like any information system, has 3 basic elements:
  - The client
  - The server
  - The storage of information.

SMBD ARCHITECTURE



CLIENT

- The client is in charge of receiving the required user applications through commands or orders and sending them to the server in a suitable format.

  Some engines allow the client to execute some compilation and interpretation functions.

SERVER

- The Server is the component that is in charge of processing the requests or tasks sent by the client.

SOFTWARE ARCHITECTURE

- Logical layers are merely a way of organizing your code
- Three layers involved in the application:
  - Presentation Layer
  - Business Layer
  - Data Layer

- In no way is it implied that these layers might run on different computers or in different processes on a single computer or even in a single process on a single computer.

SOFTWARE ARCHITECTURE

PRESENTATION LAYER

- It is also known as Client layer.
- Top most layer of an application.
- This is the layer we see when we use a software.
- By using this layer we can access the webpages.
- The main functionality of this layer is to communicate with Application layer.
- This layer passes the information which is given by the user in terms of keyboard actions, mouse clicks to the Application Layer.
- For example, login page of Gmail where an end user could see text boxes and buttons to enter user id, password and to click on sign-in.
- In a simple words, it is to view the application.

SOFTWARE ARCHITECTURE

APPLICATION LAYER

- It is also known as Business Logic Layer which is also known as logical layer.
- As per the Gmail login page example, once user clicks on the login button, Application layer interacts with data layer and sends required information to the Presentation layer.
- It controls an application's functionality by performing detailed processing.
- This layer acts as a mediator between the Presentation and the Data layer.
- In a simple words, it is to perform operations on the application..

SOFTWARE ARCHITECTURE

DATA LAYER

- Data requirements are made through this level
- Application layer communicates with Data layer to retrieve the data.
- It contains methods that connects the SMBD and performs required action e.g.: insert, update, delete etc.
- In a simple words, it is to share and retrieve the data.

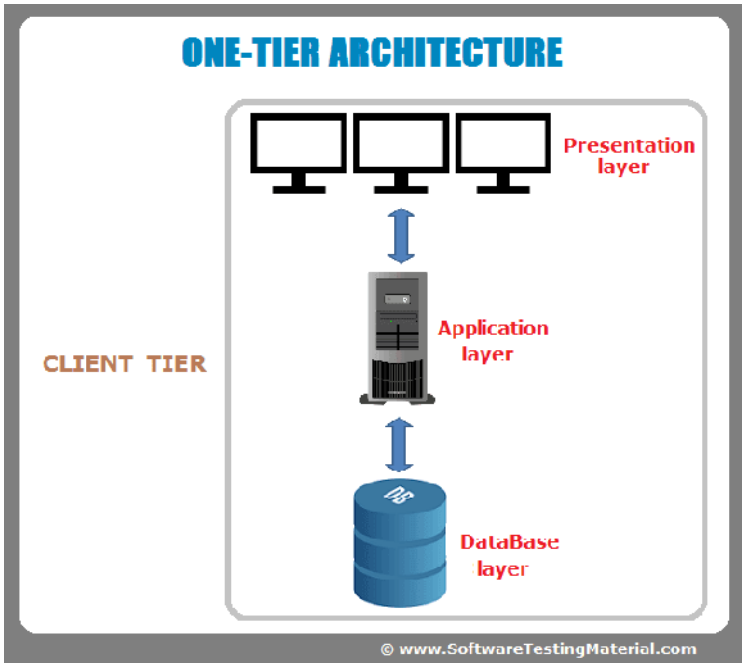
TYPES OF SYSTEM ARCHITECTURE

TIER

- Physical tiers, are only about where the code runs.
- Specifically, tiers are places where layers are deployed and where layers run.
- In other words, tiers are the physical deployment of layers..

ONE TIER ARCHITECTURE

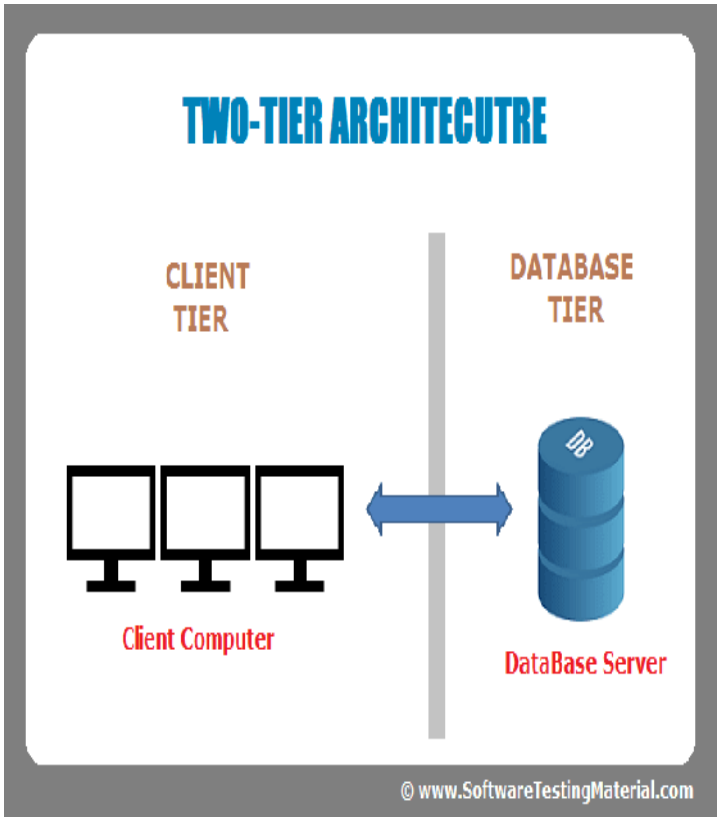
- One tier architecture has all the layers such as Presentation, Business, Data Access layers in a single software package.
- Applications which handles all the three tiers such as MP3 player, MS Office are come under one tier application.
- The data is stored in the local system or a shared drive.



TYPES OF SYSTEM ARCHITECTURE

TWO-TIER ARCHITECTURE

- The Two-tier architecture is divided into two parts:
  - Client Application (Client Tier)
  - Database (Data Tier)
- Client system handles both Presentation and Application layers Server system handles Database layer.
- It is also known as client server
- The communication takes place between the Client and the Server Database.
- Client system sends the request to the Server Database system and the Server system processes the request and sends back the data to the Client System



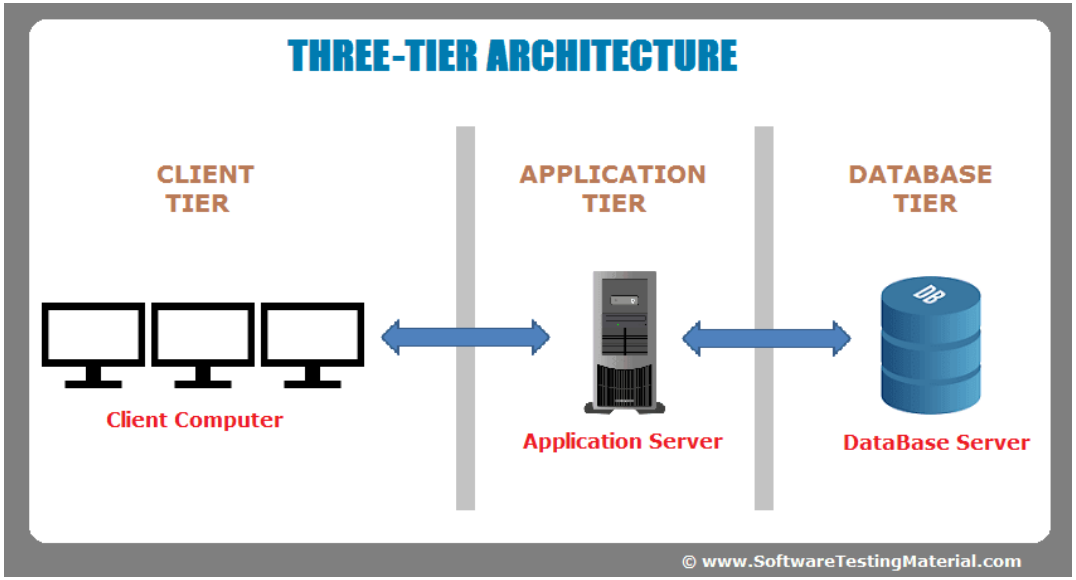
TYPES OF SYSTEN ARCHITECTURE

Three-Tier Architecture

The Three-tier architecture is divided into three parts:

- Presentation layer (Client Tier)
- Application layer (Business Tier)
- Database layer (Data Tier)

- Client system handles Presentation layer
- Application server handles Application layer
- SMBD handles Database layer



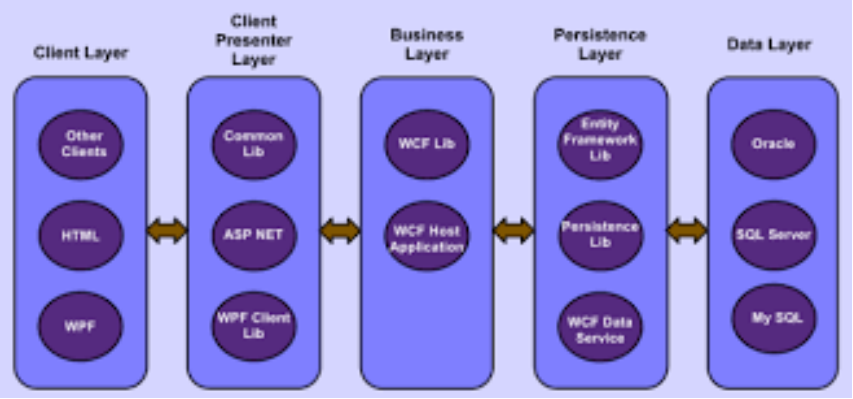
Fuente parcial: https://www.softwaretestingmaterial.com/software-architecture/

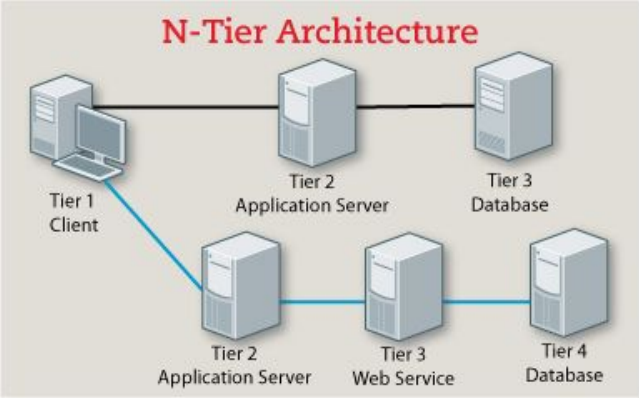
TYPES OF SYSTEM ARCHITECTURE

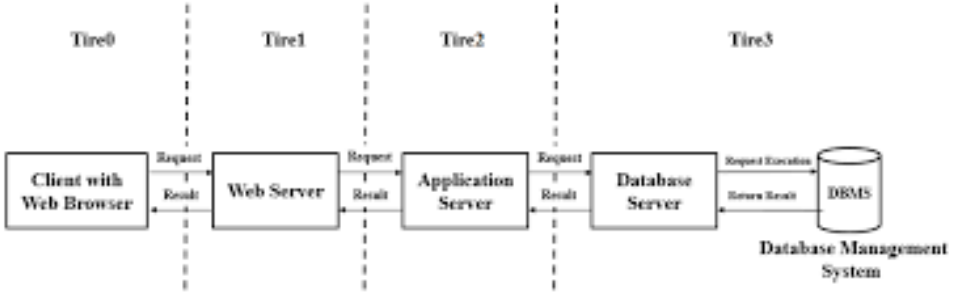
Layer is N-Tier

N-Tier application. Distributed application.

It is similar to three tier architecture but number of application servers are increased and represented in individual tiers in order to distributed the business logic.

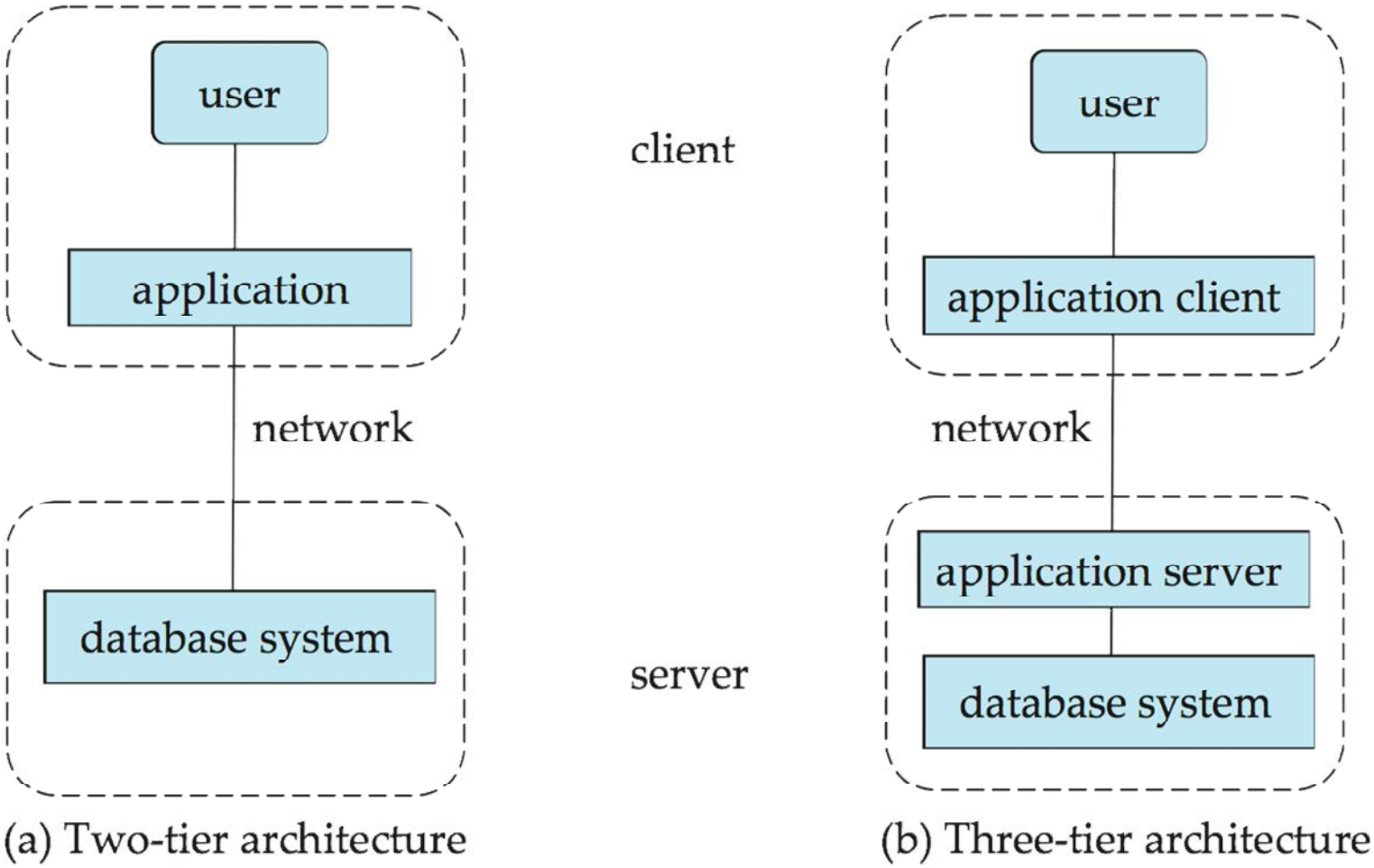






Fuente parcial: https://www.softwaretestingmaterial.com/software-architecture/

ARCHITECTURE SMBD



(a) Two-tier architecture

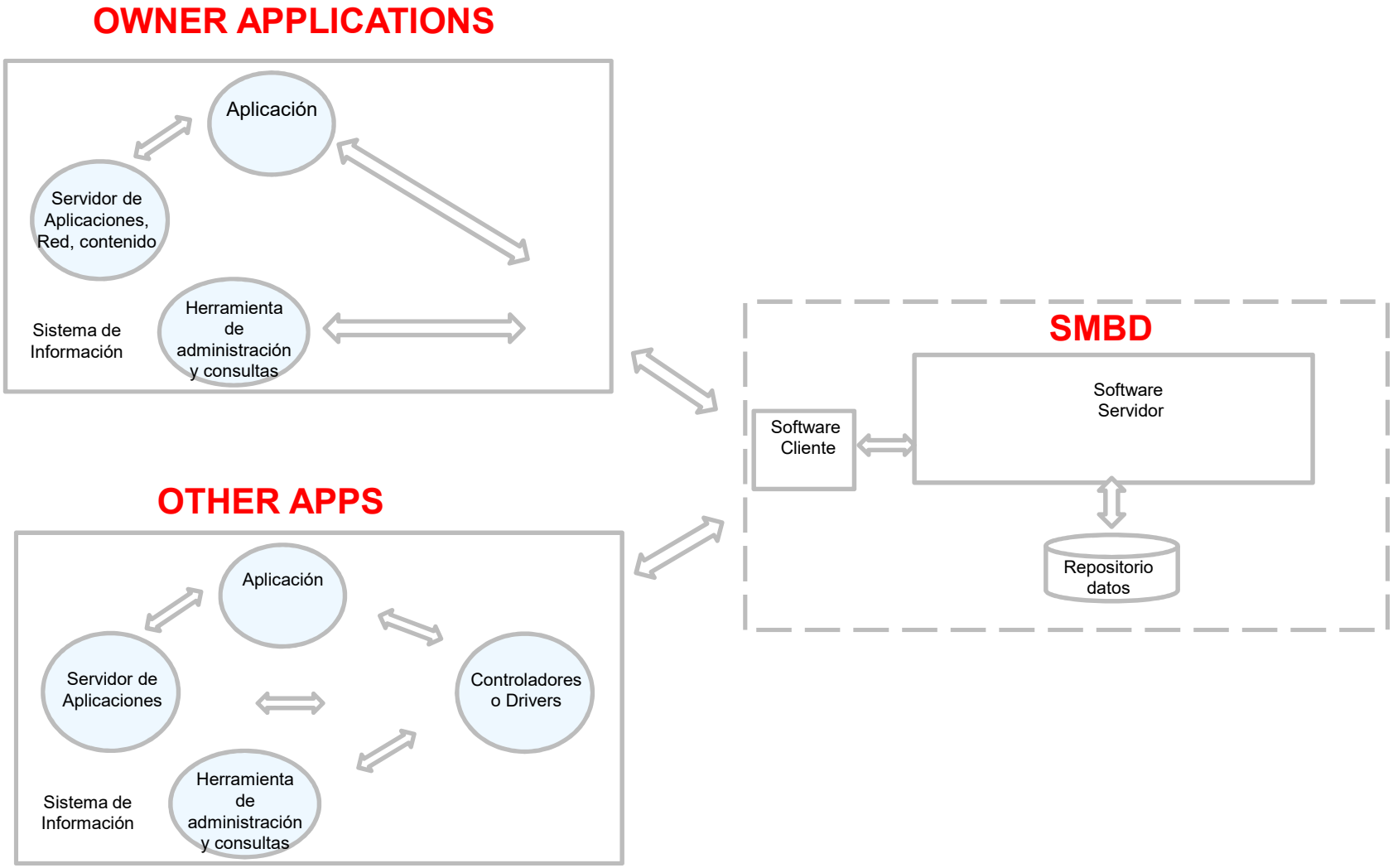
(b) Three-tier architecture

SMBD ARCHITECTURE

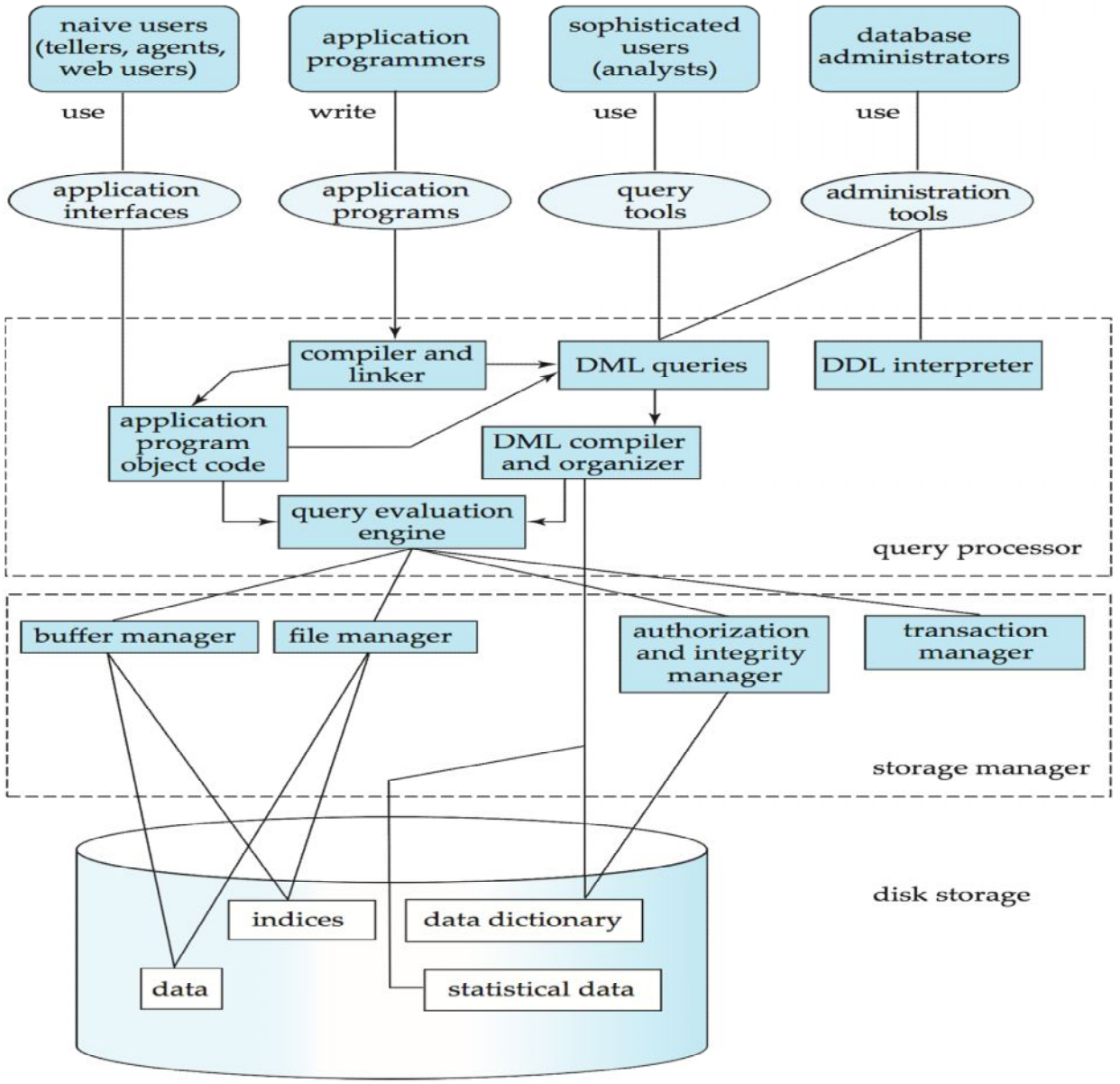
USER APPLICATIONS OF AN SMBD

- Different types of applications interact with SMBDs.
- Most applications or languages for communication with the SMBD use a standard application programming interface (API) and the implementation of that interface (controller).
- The interfaces are provided by Microsoft ODBC and by Java JDBC.
- Drivers or controllers are implementations of these interfaces and are provided by the Database engines.
- Drivers establish the logical connection (bridge) between an application or language and the client.
- The Api and the controller allow interoperability between different products despite the fact that the client behaves like a closed box for all users.

SMBD ARCHITECTURE

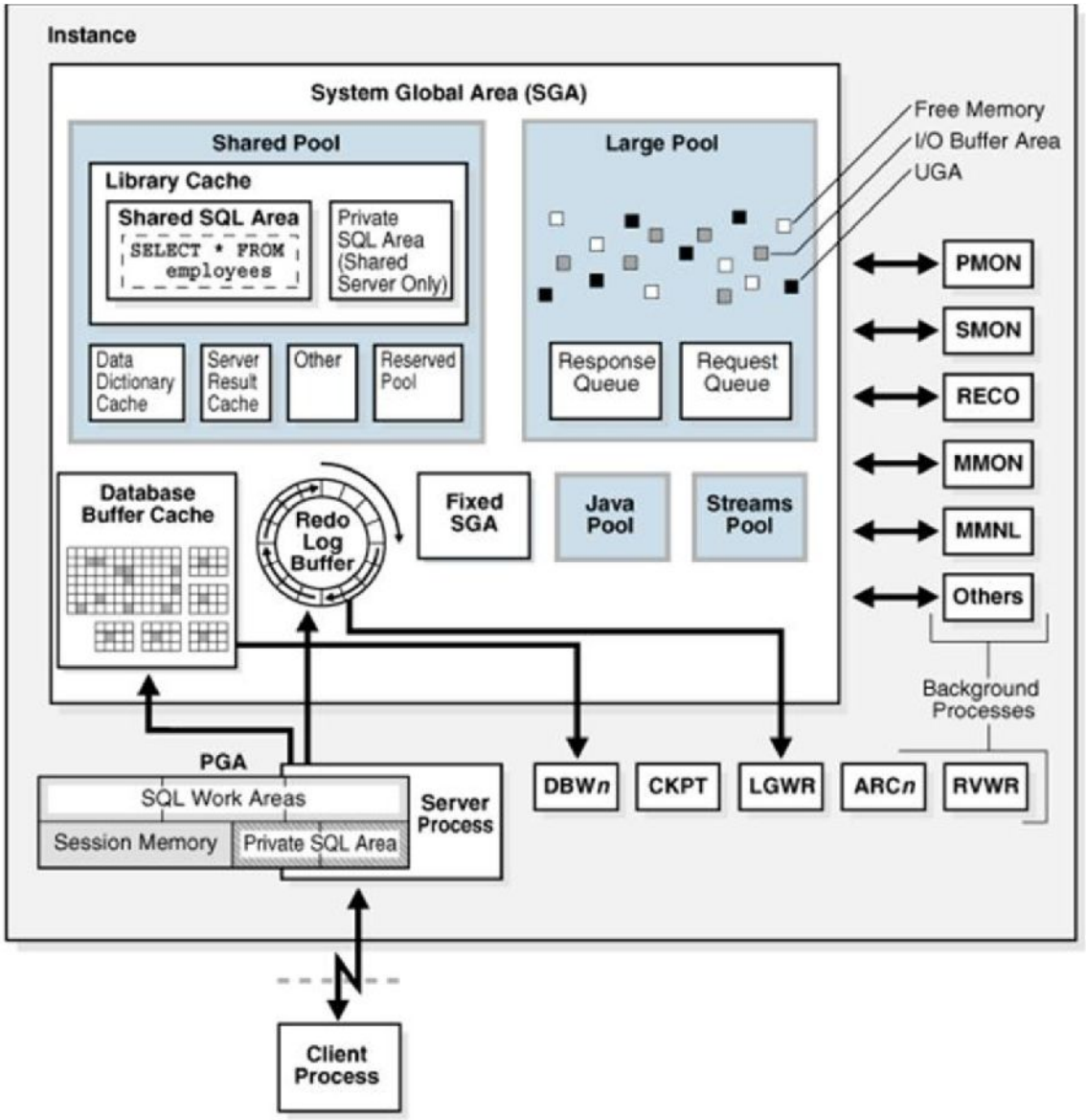


Overall Database System



Overall Database System

ORACLE



Overall Database System

The basic memory structures associated with Oracle Database include:

Program global area (PGA)

A PGA is a nonshared memory region that contains data and control information exclusively for use by an Oracle process. The PGA is created by Oracle Database when an Oracle process is started.

System global area (SGA)

The SGA is a group of shared memory structures, that contain data and control information for one Oracle Database instance.

User Global Area (UGA)

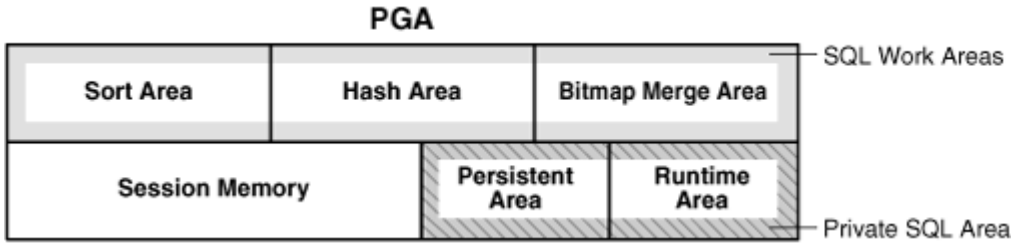
The UGA is memory associated with a user session. A small fraction of memory that it associates with the user's session.

Software code areas

Software code areas are portions of memory used to store code that is being run or can be run. Oracle Database code is stored in a software area that is typically at a different location from user programs—a more exclusive or protected location.

Overall Database System

Program Global Area (PGA)



- One PGA exists for each server process and background process. The collection of individual PGAs is the total instance PGA
- The server process allocates required memory structures when a user or pool of users connects
- The PGA is a fraction of temporary memory that contains control variables.
- It is for the exclusive use of the process that created it.

Overall Database System

System global area (SGA)

The SGA is shared by all server and background processes.

The SGA consists of several memory partitions that allocate and de-allocate *space* in contiguous memory units called granules (The granule size is platform specific and is determined by the total SGA size).

Database Buffer Cache

- When a user executes an operation that requires data registers, the blocks that contain these registers must be read from the hard disk and copied to a memory structure.
- The cache buffer is the memory area that stores the copies of the blocks recently read
- All users simultaneously connected to an instance of the database share access to this buffer.
- This buffer is managed through a list that uses a less recently used (LRU algorithm).

Overall Database System

System global area (SGA) cont.

I/O buffer

- This Buffer works as a backup to the Buffer cache and is managed through a queue.
- When the buffer cache space becomes scarce, a process checks which blocks are ready to be saved in DD and passes them to the I/O buffer.

Reading Buffer

- In this buffer, the headers of the buffers that are in the I / O buffer are stored to prevent new I / O operations on already used blocks.

Buffer to redo

- The buffer to redo is a sequential circular buffer that stores the entries that describe the changes made to a data block.
- Redo entries contain the information necessary to rebuild or redo the changes made to the database using DML or DDL operations.

Overall Database System

System global area (SGA). Cont.

Shared Pool

This buffer includes several areas: SQL Libraries, Cahe Data Dictionary, Results Area and Reserve Area.

Cache Data Dictionary

- The data dictionary is a collection of tables and views containing: Reference information about the database, its structures, objects and its users.
- The Oracle database accesses the data dictionary frequently during parsing of SQL statements.
- The data dictionary persistently remains in the DD and once an instance is started a copy is created in this cache area.

Overall Database System

System global area (SGA). Cont.

Shared Pool (Cont)

SQL libraries

- The SQL libraries store the executable code (shared SQL and PL, in case of shared and private servers), as well as the control structures.
- When an instruction is run the first time, an analysis is performed consisting of:
  - Determine what type of declaration is (SQL, DML, DDL)
  - Check the syntactic that consists of determining if it is the valid declaration. If it makes sense given the grammar, that is if you follow all the rules
- Analyze the semantics that consists of determining if:
  - The declaration is valid in light of the objects in the database (the tables and columns exist).
  - You have access to objects.
  - Comply with privileges.
  - The statement is ambiguous

Overall Database System

System global area (SGA). Cont.

Shared Pool (Cont) *SQL libraries (Cont.)*

- The database tries to reuse the code that has already been validated (smooth analysis) and that is in the library. Otherwise, the database must create a new executable version of the code, allocating new processing and storage spaces within the buffer.
- An item can be removed according to the Least Recently Used, LRU algorithm.
- The elements used by many sessions remain in this buffer

SQL query results cache

- The database stores the results of queries and query fragments in this cache.
- Use the stored results for future reference
- This performance improvement, the database avoids the costly operation of rereading blocks and recalculating the results.
- If the results are cached, they are returned immediately.
- The database invalidates the results when a transaction modifies the data or metadata of the objects used to build that cached result.

Overall Database System

System global area (SGA). Cont.

Large Pool

The database administrator can optionally configure this memory area for:

- when transactions interact with more than one database (parallel query buffers)
- I/O server processes.
- Backup and restore operations of the Oracle database.

Java Pool

- Java Memory is used for session-specific Java data and codes within the JVM, as well as statistics that may affect performance.

Streams Pool

- This structure provides memory for Oracle Streams (capture and application processes).

Overall Database System

BACKGROUND PROCESSES

User processes

User processes are created when the user starts tools such as SQl * plus, forms, developer, reports, Oracle Enterprise Manager among others.

These processes provide user interfaces that allow you to enter commands that interact with the DB.

Server Process

The Server Process accepts commands from the user processes and executes the necessary steps to complete the user's requirements.

Overall Database System

BACKGROUND PROCESSES

Database Writer (DBWn)

- Write unused buffers from buffer caches to data files.
- This activity is asynchronous.
- It ensures having a sufficient number of free buffers.
- Buffers can be overwritten when the Server Process needs to read blocks from data files
- Through the event checkpoint (trigger that runs in various situations, synchronously) writes from the buffer cache to the data files.
- Additional processes can be configured to improve writing performance
- The closed database contains only saved data unless a “shutdown abort” or some other option has been used.

Overall Database System

BACKGROUND PROCESSES

Database Writer (DBWn)

- If the data is not saved it is because an error occurs and the changes are “rollback” and the checkpoint is used to force the changes to be saved to disk.
- If the instance is downloaded normally and users have saved the data, then the data files will contain only saved data.
- When an instance is started, it may contain data that has not been saved. This happens when the data has been changed but not saved (the changed data is in the cache), which requires more disk space and forces a no save of the data.
- In the event of a failure, during the recovery sequence, redo logs and rollback segments are used to synchronize the data files.

Overall Database System

BACKGROUND PROCESSES

Log writer LGWR

The LGWR writes to the redo log buffer when:

- The redo log buffer has reached a third of its capacity
- An out of time occurs (every three seconds)
- There is one MB of redo
- Before the DBWn writes modified blocks from the buffer cache to the database.
- A transaction gets a commit

Overall Database System

BACKGROUND PROCESSES

CHECKPOINT (CKPT)

- It is responsible to signal to DBWn which is the most recent checkpoint.
- Every three seconds the CHKPT records a mark in the control file.
- When a log switch is triggered, the CHKPT writes the necessary information to the headers of the data files.

CHECKPOINT POSITION

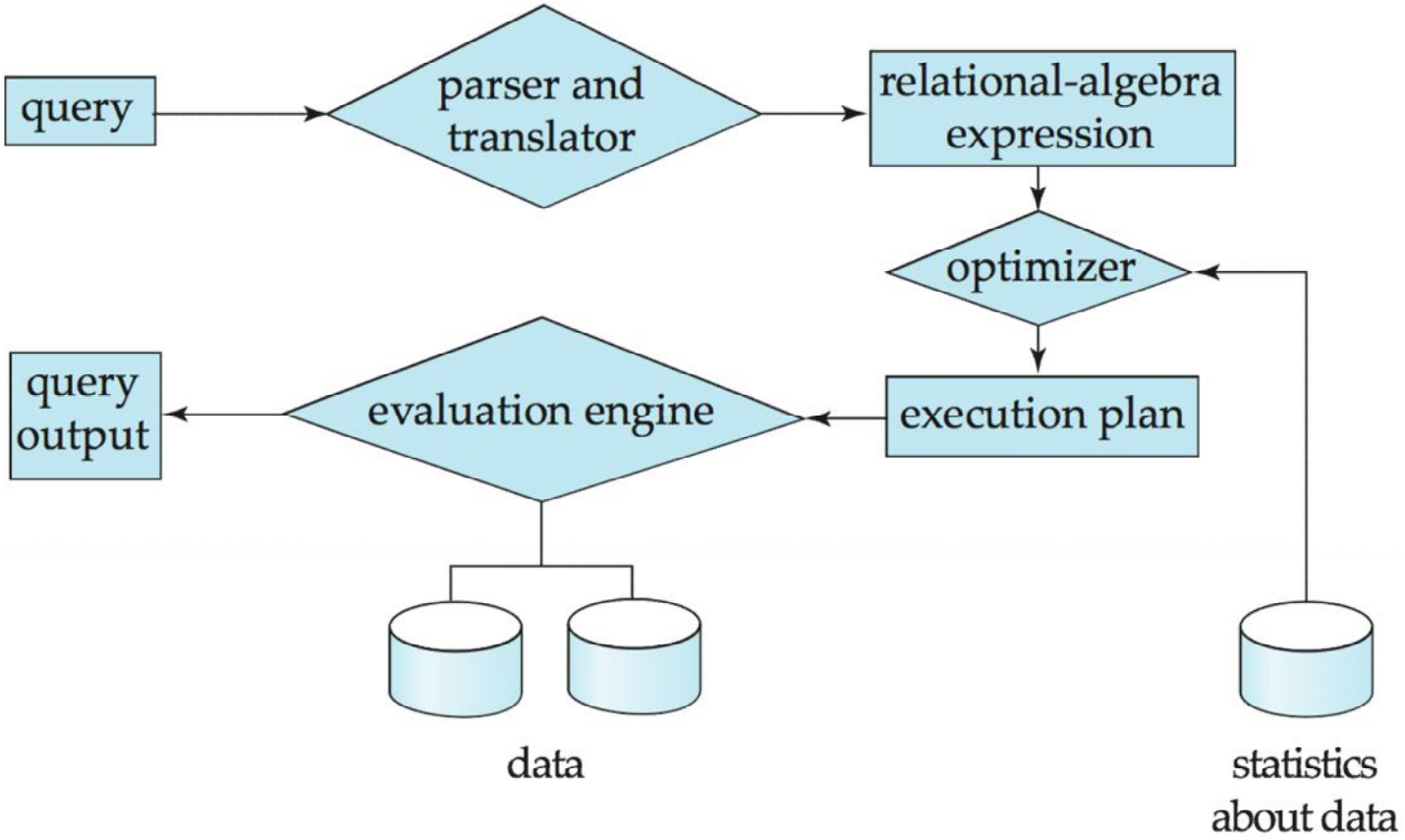
- It is the position that allows us to know from where to start a recovery.
- All blocks referenced before this point have already been written to disk by DBWn.
- This is defined as a distance between it and the end of the log redo

CHECKPOINT QUEUE

Each entry in the checkpoint queue includes the block identifier (file number and block number) and the location in the redo log of the modified data block

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Basic Steps in Query Processing (Cont.)

- Parsing and translation
  - translate the query into its internal form.
  - This is then translated into relational algebra.
  - Parser checks syntax, verifies relations
- Optimization
  - Enumerate all possible query-evaluation plans
  - Compute the cost for the plans
  - Pick up the plan having the minimum cost
- Evaluation
  - The query-execution engine takes a query-evaluation plan,
  - executes that plan,
  - and returns the answers to the query.

Query Processing (Cont.)

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
- Cost difference between a good and a bad way of evaluating a query can be enormous
- Need to estimate the cost of operations
  - Depends critically on statistical information about relations which the database must maintain
  - Need to estimate statistics for intermediate results to compute cost of complex expressions

Basic Steps in Query Processing : Optimization

- A relational algebra expression may have many equivalent expressions
  - $E_1 \cup E_2 \approx E_2 \cup E_1$
  - $(E_1 \cup E_2) \cup E_3 \approx E_1 \cup (E_2 \cup E_3)$

- Each relational algebra operation can be evaluated using one of several different algorithms
  - Correspondingly, a relational-algebra expression can be evaluated in many ways.

Basic Steps: Optimization (Cont.)

Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**.

- E.g., can use an index on *salary* to find instructors with salary < 75000,
- or can perform complete relation scan and discard instructors with salary ≥ 75000

- **Query Optimization**
  - Amongst all equivalent evaluation plans choose the one with lowest cost.
  - Cost is estimated using statistical information from the database catalog
    - e.g. number of tuples in each relation, size of tuples, etc.

1.9 Transaction Management

- What if the system fails?
- What if more than one user is concurrently updating the same data?
- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Example of Fund Transfer

Transaction to transfer $50 from account A to account B:

1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B*)

1. **read**(*A*)
2. *A* := *A* – 50
3. **read**(*B*)
4. *B* := *B* + 50
5. **write**(*A*)
6. **write**(*B*)

Atomicity requirement

if the transaction fails after step 3 and before step 6, money will be "lost" leading to an inconsistent database state

Failure could be due to software or hardware

the system should ensure that updates of a partially executed transaction are not reflected in the database

Durability requirement

Once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures.

Example of Fund Transfer (Cont.)

Consistency requirement

- the sum of A and B is unchanged by the execution of the transaction
- In general, consistency requirements include
  - Explicitly specified integrity constraints such as primary keys and foreign keys
  - Implicit integrity constraints
    - e.g. sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
- A transaction must see a consistent database.
- During transaction execution the database may be temporarily inconsistent.
- When the transaction completes successfully the database must be consistent
  - Erroneous transaction logic can lead to inconsistency

Example of Fund Transfer (Cont.)

- **Isolation requirement** — if between steps 3 and 6, another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database (the sum $A + B$ will be less than it should be).

T1	T2
1. **read**(A)	
2. $A := A - 50$	
3. **write**(A)	
	read(A), read(B), print(A+B)
4. **read**(B)	
5. $B := B + 50$	
6. **write**(B	

- Isolation can be ensured trivially by running transactions **serially**
  - that is, one after the other.
- However, executing multiple transactions concurrently has significant benefits, as we will see later.

ACID Properties

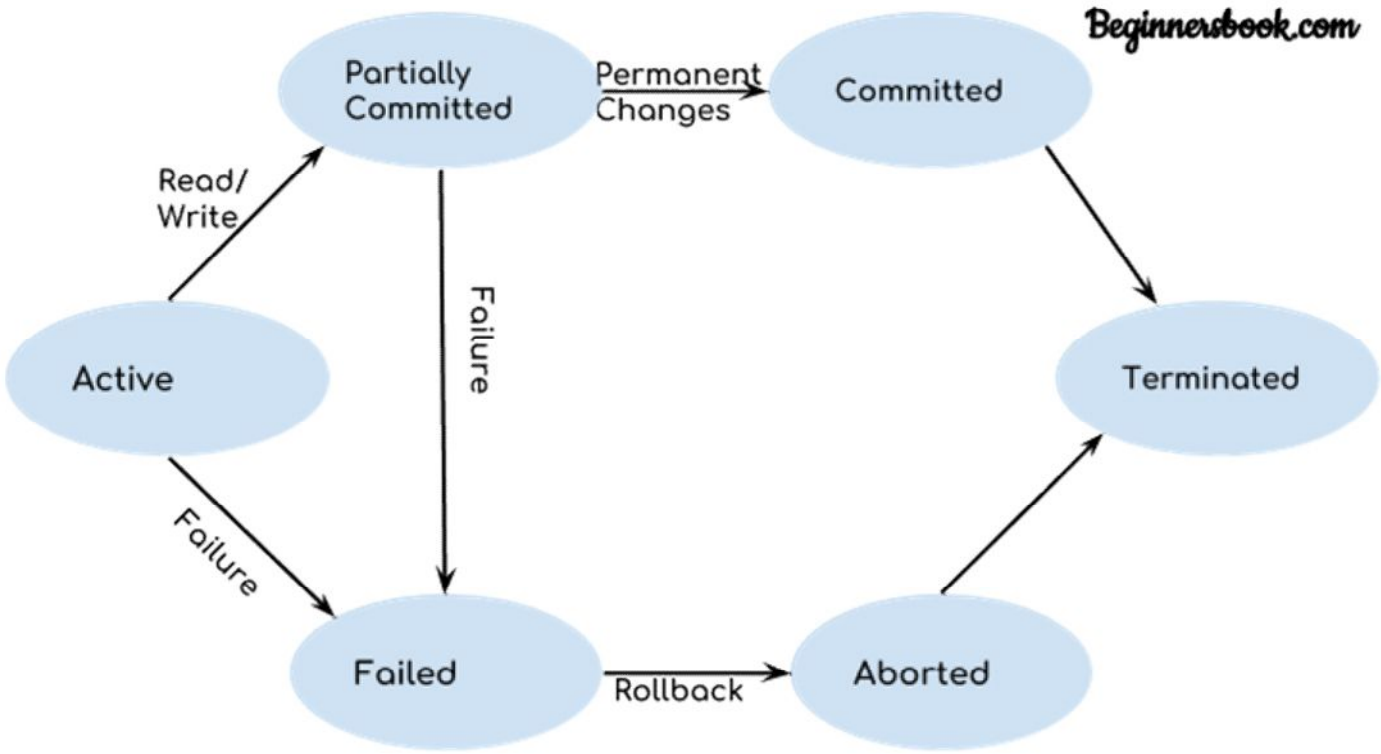
A **transaction** is a unit of program execution that accesses and possibly updates various data items.To preserve the integrity of data the database system must ensure:

- **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are.
- **Consistency.** Execution of a transaction in isolation preserves the consistency of the database.
- **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
  - That is, for every pair of transactions T_i and T_j, it appears to T_i that either T_j, finished execution before T_i started, or T_j started execution after T_i finished.
- **Durability.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Transaction State

- **Active** – the initial state; the transaction stays in this state while it is executing
- **Partially committed** – after the final statement has been executed.
- **Failed** -- after the discovery that normal execution can no longer proceed.
- **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - restart the transaction
    - can be done only if no internal logical error
  - kill the transaction
- **Committed** – after successful completion.

Transaction State (Cont.)



Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
  - **increased processor and disk utilization**, leading to better transaction *throughput*
    - E.g. one transaction can be using the CPU while another is reading from or writing to the disk
  - **reduced average response time** for transactions: short transactions need not wait behind long ones.
- **Concurrency control schemes** – mechanisms to achieve isolation
  - that is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - a schedule for a set of transactions must consist of all instructions of those transactions
  - must preserve the order in which the instructions appear in each individual transaction.
- Serial schedule – instruction sequences from one by one transactions
- A transaction that successfully completes its execution will have a commit instructions as the last statement
  - by default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an abort instruction as the last statement

Schedule 1

- Let T_1 transfer \$50 from *A* to *B*, and T_2 transfer 10% of the balance from *A* to *B.*
- A serial schedule in which T_1 is followed by T_2 :

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 2

- A serial schedule where T_2 is followed by T_1

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

Schedule 3

- Let T_1 and T_2 be the transactions defined previously. The following schedule is not a serial schedule, but it is *equivalent* to Schedule 1. We call it a serializable schedule

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

In Schedules 1, 2 and 3, the sum A + B is preserved.

Schedule 4

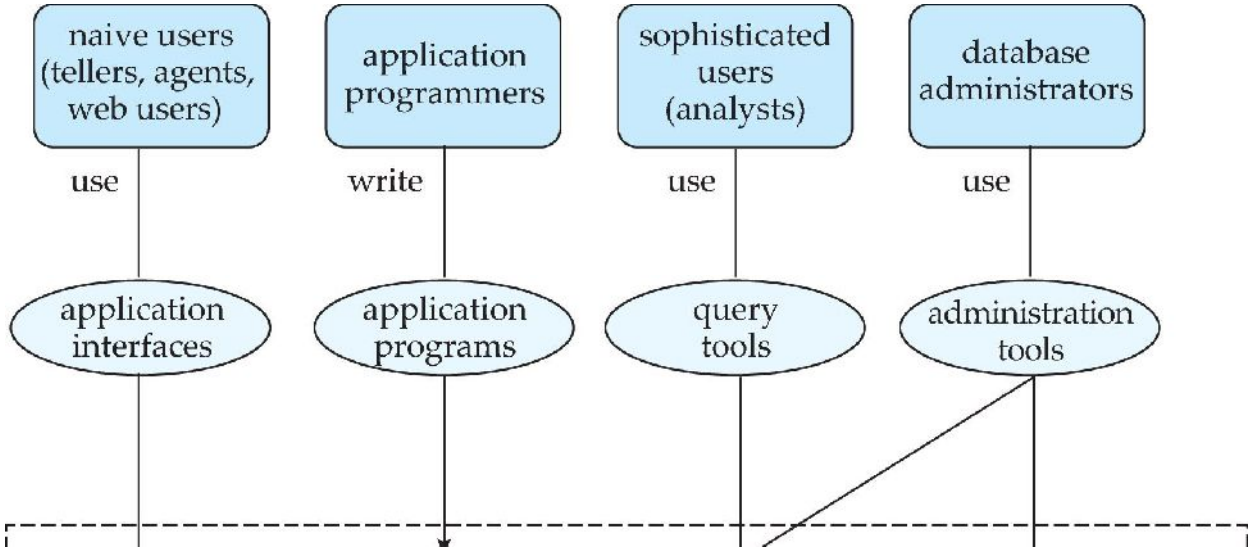
- The following concurrent schedule does not preserve the value of (*A* + *B*). The following schedule is not serializable.

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

Serializability

- **Basic Assumption** – Each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
  1. **conflict serializability**
  2. **view serializability**

1.12 Database Users and Administrators



Database Users

Users are differentiated by the way they expect to interact with the system

- **Application programmers** – interact with system through DML calls
- **Sophisticated users** – form requests in a database query language
- **Specialized users** – write specialized database applications that do not fit into the traditional data processing framework
- **Naïve users** – invoke one of the permanent application programs that have been written previously
  - Examples, people accessing database over the web, bank tellers, clerical staff

Database Administrator

- Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements

End of Chapter 2