Unit – 8

Distributed Transaction Management

&

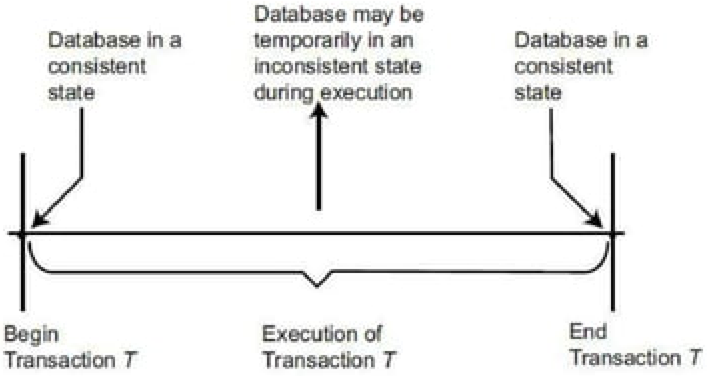
Concurrency Control

Outlines..

- Transaction concept
- ACID property
- Objectives of Distributed Concurrency Control
- Concurrency Control anomalies
- Methods of concurrency control
- Serializability and recoverability
- Multiple granularity
- Multi version schemes

Transaction concept

- A transaction is considered to be made up of a sequence of read and write operation on the database, together with computation.


Database in a consistent state
Database may be temporarily in an inconsistent state during execution
Database in a consistent state
Begin Transaction *T*
Execution of Transaction *T*
End Transaction *T*

Example of Transactions

- Updating a record
- Locate the record on disk
- Bring record into buffer
- Update data in the buffer
- Writing data back to disk

Syntax of Transaction

- Consider the following SQL query for increasing by 10% the budget of the CAD/CAM project.

➢
```
UPDATE PROJ
SET BUDGET = BUDGET*1.1
WHERE PNAME= "CAD/CAM"
```

➢
```
Begin transaction BUDGET UPDATE
begin
    EXEC SQL UPDATE PROJ
    SET BUDGET = BUDGET*1.1
    WHERE PNAME= "CAD/CAM"
end.
```

Directed acyclic graph (DAC)

$$\begin{tikzcd}
R(x) \arrow[dr] & & \\
 & W(x) \arrow[r] & C \\
R(y) \arrow[ur] & &
\end{tikzcd}$$

Properties of Transactions

- The consistency and reliability aspects of transactions are due to four properties

1. Atomicity
2. Consistency
3. Isolation
4. Durability

- Together, these are commonly referred to as the ACID properties of transactions.

Atomicity

- Atomicity refers to the fact that a transaction is treated as a unit of operation. Therefore, either all the transaction’s actions are completed, or none of them are. This is also known as the “all-or-nothing property.”

Consistency

- The consistency of a transaction is simply its correctness.
- In other words, a transaction is a correct program that maps one consistent database state to another.

Isolation

- Isolation is the property of transactions that requires each transaction to see a consistent database at all times.
- In other words, an executing transaction cannot reveal its results to other concurrent transactions before its commitment.

Durability

- Durability refers to that property of transactions which ensures that once a transaction commits, its results are permanent and cannot be erased from the database.

Objectives of transaction management

- CPU and main memory utilization
- Control message and their response time
- Availability

Types of transactions

- Transactions have been classified according to a number of criteria. One criterion is the duration of transactions. Accordingly, transactions may be classified as **online** or **batch.**
- These two classes are also called short-life and long-life transactions, respectively.

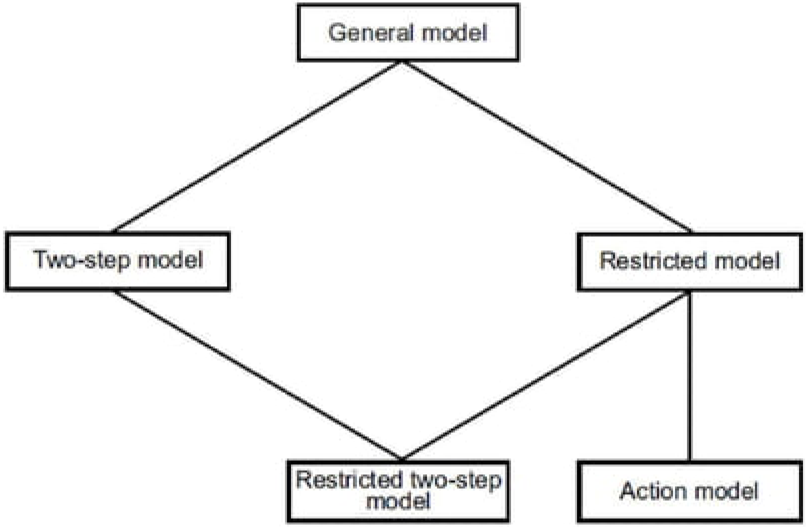
Online Transaction

- Online transactions are characterized by very short execution/response times (typically, on the order of a couple of seconds) and by access to a relatively small portion of the database.
- This class of transactions probably covers a large majority of current transaction applications. Examples include banking transactions and airline reservation transactions.

Batch Transactions

- Batch transactions, on the other hand, take longer to execute (response time being measured in minutes, hours, or even days) and access a larger portion of the database.
- Typical applications that might require batch transactions are design databases, statistical applications, report generation, complex queries, and image processing.

Various Transaction Models



Conti..

- Another classification that has been proposed is with respect to the organization of the read and write actions.

1. Read and write actions without any specific ordering. We call this type of **transactions general**.
2. If the transactions are restricted so that all the read actions are performed before any write action, the transaction is called a **two-step transaction.**
3. if the transaction is restricted so that a data item has to be read before it can be updated (written), the corresponding class is called **restricted (or read-before-write)**
4. If a transaction is both two-step and restricted, it is called a **restricted two-step transaction**.
5. **action model of transactions**, which consists of the restricted class with the further restriction that each <read>,<write> pair be executed atomically.

Flat Transactions

- Flat transactions have a single start point (Begin transaction) and a single termination point (End transaction).

Nested Transactions

- An alternative transaction model is to permit a transaction to include other transactions with their own begin and commit points. Such transactions are called nested transactions.

```
Begin transaction Reservation
begin
   Begin transaction Airline
   : : :
   end. {Airline}
   Begin transaction Hotel
   : : :
   end. {Hotel}
end.
```

Workflows

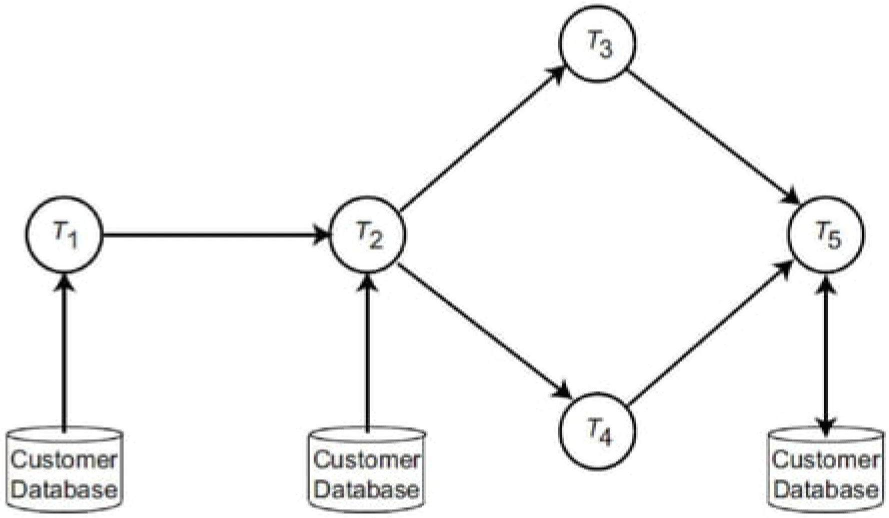
- **A working definition is that a workflow is "a collection of tasks organized to accomplish some business process."**

1. **Human-oriented workflows**, which involve humans in performing the tasks. The system support is provided to facilitate collaboration and coordination among humans, but it is the humans themselves who are ultimately responsible for the consistency of the actions.
2. **System-oriented workflows** are those that consist of computation-intensive and specialized tasks that can be executed by a computer. The system support in this case is substantial and involves concurrency control and recovery, automatic task execution, notification, etc.

Workflows

3. **Transactional workflows** range in between human-oriented and system oriented workflows and borrow characteristics from both. They involve "coordinated execution of multiple tasks that (a) may involve humans, (b) require access to HAD [heterogeneous, autonomous, and/or distributed] systems, and (c) support selective use of transactional properties [i.e., ACID properties] for individual tasks or entire workflows."

Example Workflow



Objectives of Distributed Concurrency Control

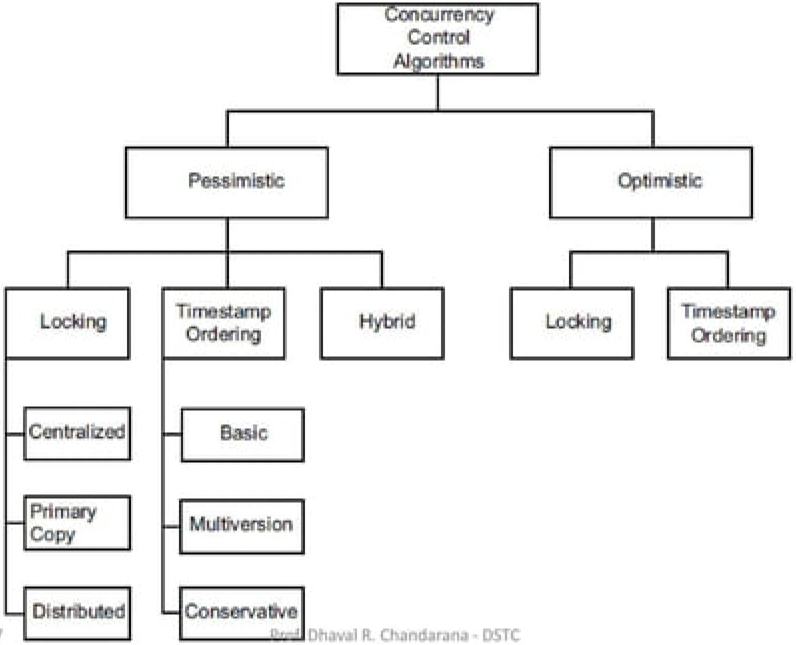
- In distributed database system, database is typically used by many users. These system usually allow multiple transaction to run concurrently at the same time.
- It must support parallel execution of transaction.
- Communication delay is less.
- It must be recovery from site and communication failure.

Concurrency Control anomalies

- Lack of Concurrency Control can create data integrity and consistency problem:

1. Lost updates
2. Uncommitted data
3. Inconsistent retrievals

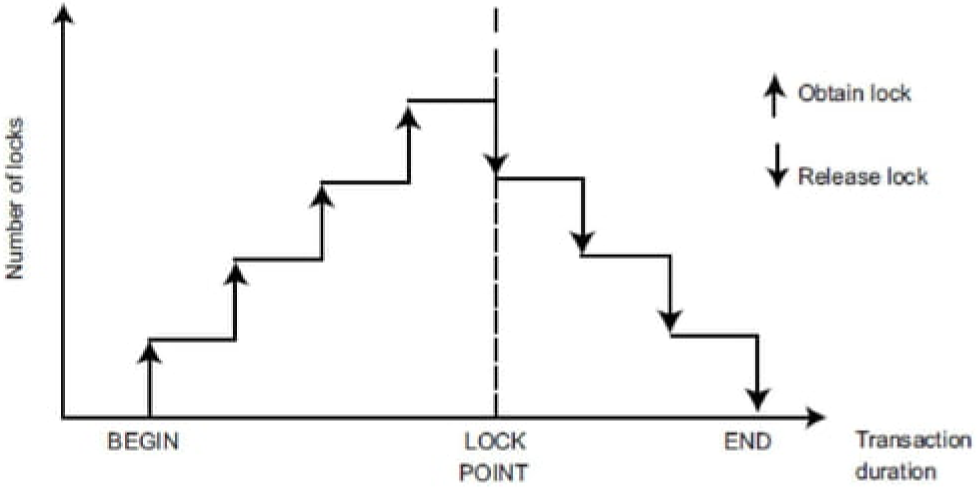
Methods of concurrency control



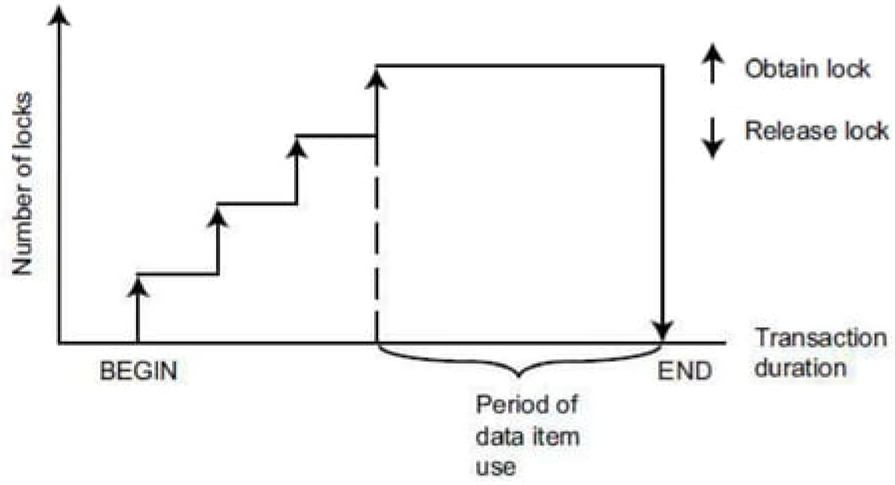
Locking-Based Concurrency Control

- The main idea of locking-based concurrency control is to ensure that a data item that is shared by conflicting operations is accessed by one operation at a time.
- This lock is set by a transaction before it is accessed and is reset at the end of its use.
- There are two types of locks read lock (rl) and write lock (wl)

Locking-Based Concurrency Control Algorithms



2PL Lock Graph



Timestamp-Based Concurrency Control Algorithms

- To establish this ordering, the transaction manager assigns each transaction Ti a unique timestamp, ts(Ti), at its initiation.
- A timestamp is a simple identifier that serves to identify each transaction uniquely and is used for ordering.
- **Uniqueness** is only one of the properties of timestamp generation.
- The second property is **monotonicity**.
- There are a number of ways that timestamps can be assigned. One method is to use a global (system-wide) monotonically increasing counter.
- However, the maintenance of global counters is a problem in distributed systems. Therefore, it is preferable that each site autonomously assigns timestamps based on its local counter.

{local counter value, site identifier}

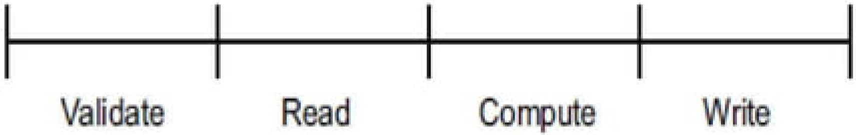
Basic timestamp ordering Rule

- A transaction's request to write an object is valid only if that object was last read and written by earlier transaction.

- A transaction's request to read an object is valid only if that object was last written by earlier transaction.

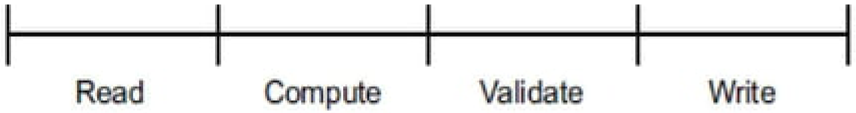
Optimistic Concurrency Control Algorithms

- the conflicts between transactions are quite frequent and do not permit a transaction to access a data item if there is a conflicting transaction that accesses that data item.
- Thus the execution of any operation of a transaction follows the sequence of phases: validation (V), read (R),computation (C), write (W)



Optimistic Concurrency Control Algorithms

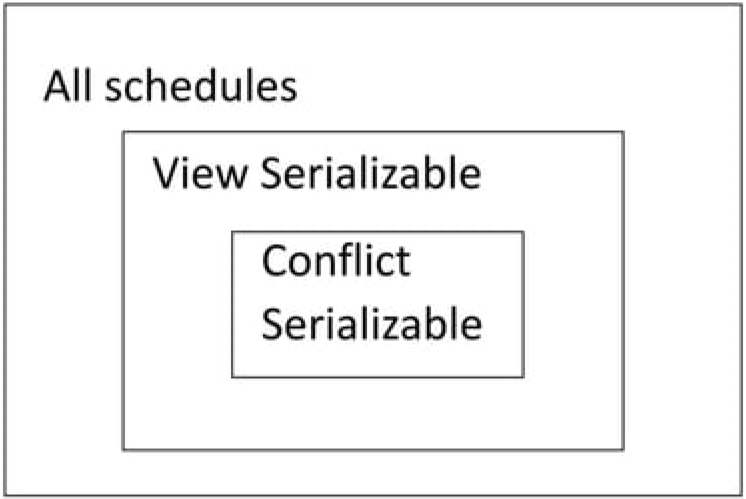
- Optimistic algorithms, on the other hand, delay the validation phase until just before the write phase.
- The read, compute, and write operations of each transaction are processed freely without updating the actual database.
- Each transaction initially makes its updates on local copies of data items. The validation phase consists of checking if these updates would maintain the consistency of the database. If the answer is affirmative, the changes are made global otherwise, the transaction is aborted and has to restart.



Serializability

- Transaction are considered serialisable if the effect of running them in an interleaved fashion is equivalent to running them serially in some order.

Diagram



Serial Schedule

			A	B	C
	Read (A, t)		500	500	500
	t = t - 100				
T1	Write (A, t)				
	Read (B, t)				
	t = t + 100				
	Write (B, t)		400	600	500
		Read (A, s)			
		s = s - 100			
		Write (A, s)			
T2		Read (C, s)			
		s = s + 100			
		Write (C, s)	300	600	600

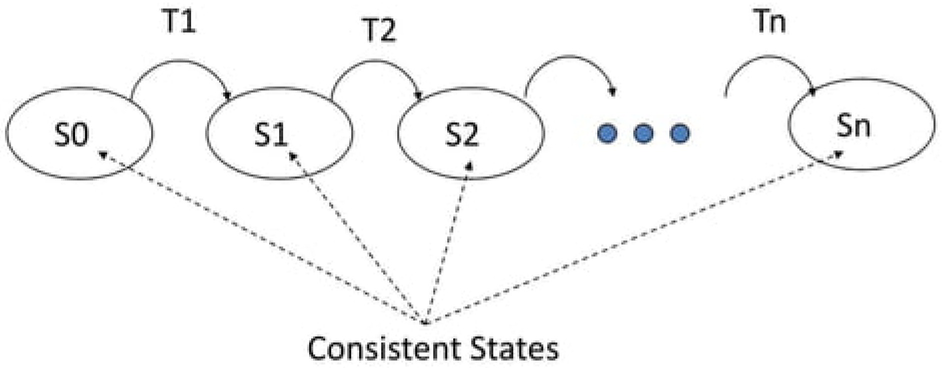
$\boxed{300 + 600 + 600 = 1500}$

Serial Schedule

			A	B	C
		Read (A, s)	500	500	500
		s = s - 100			
		Write (A, s)			
T2		Read (C, s)			
		s = s + 100			
		Write (C, s)	400	500	600
	Read (A, t)				
	t = t - 100				
	Write (A, t)				
T1	Read (B, t)				
	t = t + 100				
	Write (B, t)		300	600	600

300 + 600 + 600 = 1500

Serial Schedule



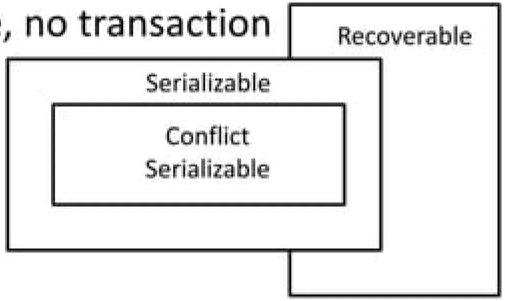
Conflict Serializability

	T2: Read (A)	T2: Write (A)
T1: Read (A)	OK	Read/Write Conflict
T1: Write (A)	Write/Read Conflict	Write/Write Conflict

1. **Read/Write Conflict:** conflict because value read depend on whether write has occurred.
2. **Write/Write Conflict:** conflict because value left in db depend on which write occurred last.
3. **Read/Read**: **no conflict.**

Recoverability

- If transaction fails, users undo the transaction effect because of atomicity property. The durability property states that once a transaction commits, its change cannot be undone.
- In recoverable schedule, no transaction need to be roll back.

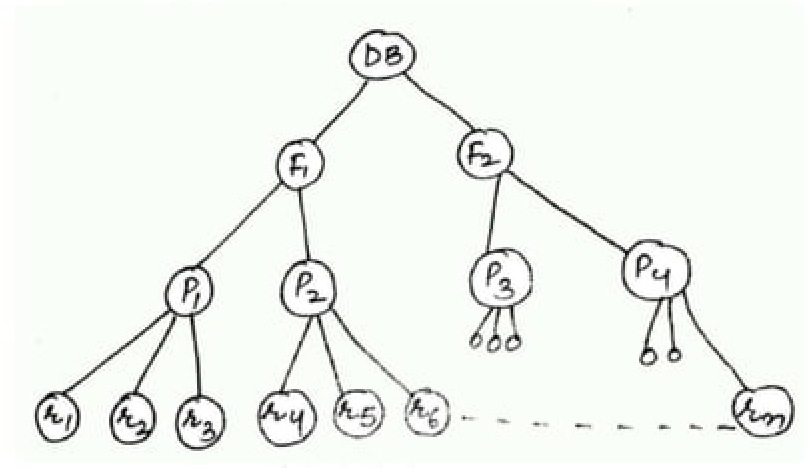


Multiple granularity

- **Granularity** is the size of data item allowed to lock.
- **Multiple Granularity** is the hierarchically breaking up the database into portions which are lockable and maintaining the track of what to be lock and how much to be lock so that it can be decided very quickly either to lock a data item or to unlock a data item.

Example of Multiple Granularity

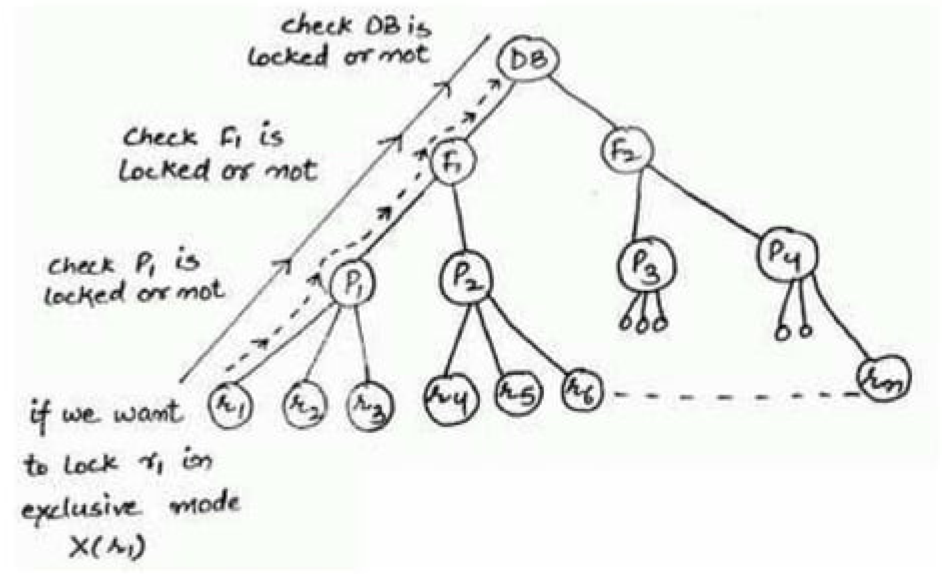
- Suppose a database is divided into files; files are divided into pages; pages are divided into records.



Example of Multiple Granularity

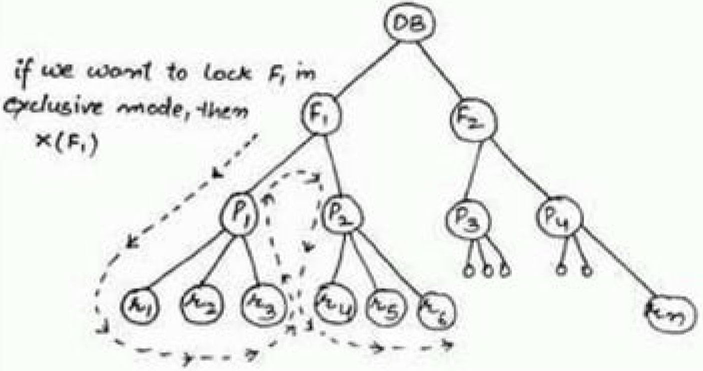
- if there is a need to lock a record, then a transaction can easily lock it. But if there is a need to lock a file, the transaction have to lock firstly all the records one after another, then pages in that file and finally the file. So, there is a need to provide a mechanism for locking the files also which is provided by multiple granularity.

For a low-level request



check DB is locked or not
check F1 is locked or not
check P1 is locked or not
if we want to lock r1 in exclusive mode X(r1)
DB
F1
F2
P1
P2
P3
P4
r1
r2
r3
r4
r5
r6
rn

For a high-level request



∴ Checking Orders:

$(F_1) \Rightarrow (P_1) \Rightarrow (r_1) \Rightarrow (r_2) \Rightarrow (r_3) \Rightarrow (P_2) \Rightarrow (r_4) \Rightarrow (r_5) \Rightarrow (r_6)$

Multi version schemes

- Multiversion schemes keep old version of data item to increase concurrency. Each successful write result in the creation of a new version of the data item written.
- When a read (Q) operation is issued, select an appropriate version of Q base on the timestamp of the transaction and return the value of the selected version.

Multi version timestamp ordering

- Each data item Q has sequence of versions < Q1,Q2,….,Qm>. Each version Qk contain three data fields:

a. **Content**: the value of version Qk.
b. **W-timestamp (Qk)**: timestamp of the transaction that created (wrote) version Qk.
c. **R-timestamp (Qk)**: largest timestamp of the transaction that successfully read version Qk.