ADMT - November 2018 - Paper Solution

1-a) Explain brute force nested loop join algorithm. [5M]

Brute-Force Nested Loop Join algorithm [1M]
This is a simple algorithm to compute the theta join between two relations R and S. This algorithm is called the nested-loop join algorithm, since it basically consists of a pair of nested for loops. Relation R is called the outer relation and relation S the inner relation of the join, since the loop for R encloses the loop for S. The algorithm uses the notation $t_r \cdot t_s$, where t_r and t_s are tuples; $t_r \cdot t_s$ denotes the tuple constructed by concatenating the attribute values of tuples t_r and t_s .

Algorithm [2M]
```
for each tuple t_r in R do begin
  for each tuple t_s in S do begin
       test pair (t_r, t_s) to see if they satisfy the join condition theta
       if they do then add t_r · t_s to the result;
  end
end
```

Cost (Seeks, Black transfers) [2M]
b_r -> number of blocks of relation R
b_s -> number of blocks of relation S
n_r -> number of tuples of relation R
n_s -> number of tuples of relation S

Seeks
We need only one seek for each scan on the inner relation S since it is read sequentially, and a total of br seeks to read r and therefore, Number of Seeks = $b_r + n_r$

Block transfers
For each record in R, we have to perform a complete scan on S. In the worst case, the buffer can hold only one block of each relation and therefore, Number of block transfers = $b_r + n_r * b_s$

1-b) What is deadlock, explain wait and die scheme used for deadlock prevention. [5M]

Deadlock explained [1M]
Deadlock occurs when each transaction T in a set of two or more transactions are waiting for some item that is locked by some other transaction T′ in the set. Hence, each transaction in the set is in a waiting queue, waiting for one of the other transactions in the set to release the lock on an item. But because the other transaction is also waiting, it will never release the lock.

Wait and die scheme [3M]
It's a deadlock prevention scheme where each transaction is given a unique timestamp (TS). Suppose that transaction T_i tries to lock an item X but is not able to lock it because X is locked by some other transaction T_j with a conflicting lock. The rule followed by this scheme is:

```
If TS(T_i) < TS(T_j), then ... T_i older than T_j
   Ti is allowed to wait;
otherwise ... T_i younger than T_j
  abort T_i (T_i dies) and restart it later with the same timestamp.
```

Example [1M]

For example if there are three transactions T_5, T_{10}, and T_{15} (subscript represents their timestamps).
If T_5 requests a data item held by T_{10}, then T_5 will wait since 5 < 10.
If T_{15} requests a data item held by T_{10}, then T_{15} will be aborted since 15 > 10.

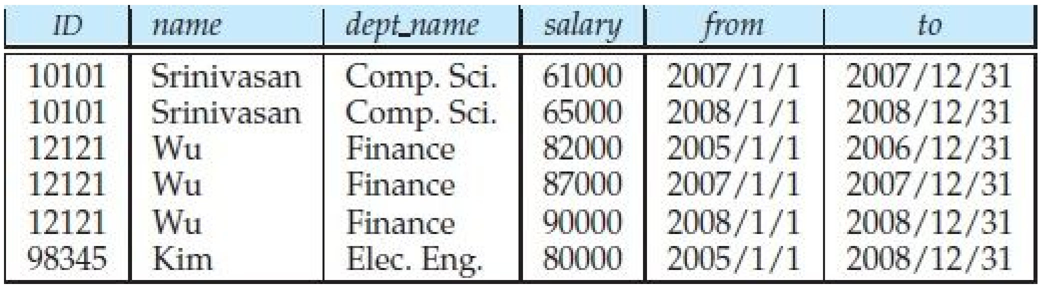
1-c) What is temporal database? What are its characteristics? [5M]

Temporal database [1M]
Databases that store information about states of the real world across time are called temporal databases. In many applications, it is important to store and retrieve information about past states. For example, a patient database must store information about the medical history of a patient. A factory monitoring system may store information about current and past readings of sensors in the factory, for analysis.

Characteristics [3M]

- The valid time for a fact is the set of time intervals during which the fact is true in the real world.
- The transaction time for a fact is the time interval during which the fact is current within the database system.
- A temporal relation is one where each tuple has an associated time when it is true; the time may be either valid time or transaction time.
- If both valid time and transaction time are stored, then the relation is called a bitemporal relation.
- A temporal selection is a selection that involves the time attributes.
- A temporal projection is a projection where the tuples in the projection inherit their times from the tuples in the original relation
- A temporal join is a join, with the time of a tuple in the result being the intersection of the times of the tuples from which it is derived. If the times do not intersect, the tuple is removed from the result.
- The predicates precedes, overlaps, and contains can be applied on intervals.

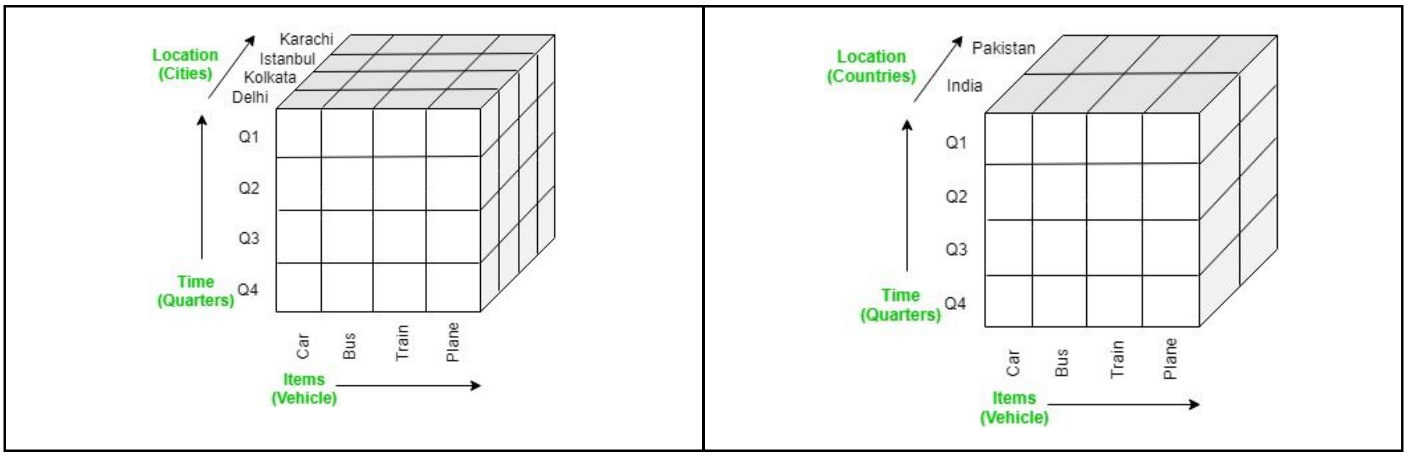
Example temporal relation - **instructor** (includes validation time with attributes **from** and **to**)



ID	*name*	*dept_name*	*salary*	*from*	*to*
10101	Srinivasan	Comp. Sci.	61000	2007/1/1	2007/12/31
10101	Srinivasan	Comp. Sci.	65000	2008/1/1	2008/12/31
12121	Wu	Finance	82000	2005/1/1	2006/12/31
12121	Wu	Finance	87000	2007/1/1	2007/12/31
12121	Wu	Finance	90000	2008/1/1	2008/12/31
98345	Kim	Elec. Eng.	80000	2005/1/1	2008/12/31

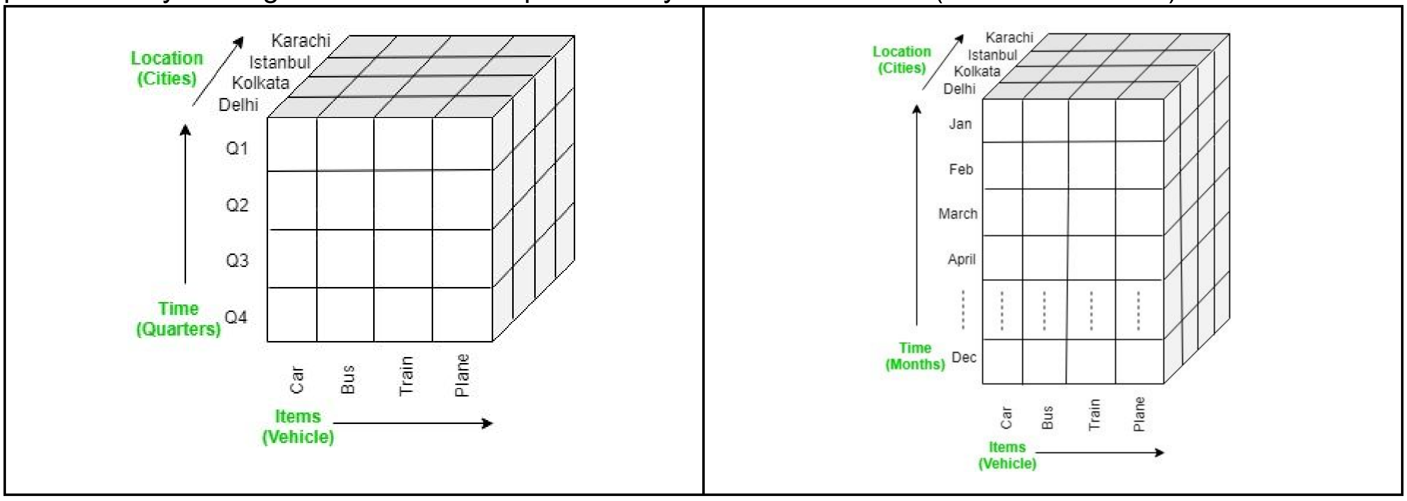
1-d) Explain roll-up, drill down, slice, dice operations in OLAP. [5M]

OLAP Roll-up operation [1M]
It performs aggregation on the OLAP cube. It can be done by climbing up in the concept hierarchy or reducing the dimensions. Example roll-up operation is performed by climbing up in the concept hierarchy of Location dimension (City -> Country).



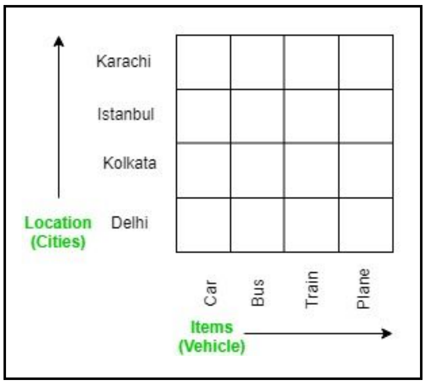
OLAP Drill-down operation [1M]

In drill-down operation, the less detailed data is converted into highly detailed data. It can be done by Moving down in the concept hierarchy or Adding a new dimension. Example drill down operation is performed by moving down in the concept hierarchy of Time dimension (Quarter -> Month).



OLAP Slice operation [1M]

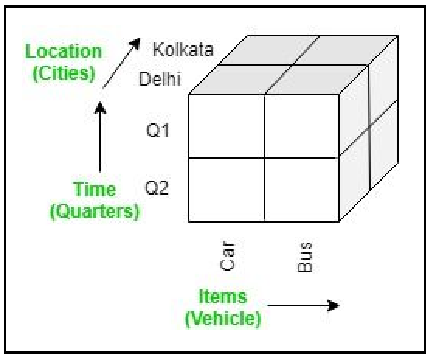
The slice operation performs a selection on one dimension of the given cube. Example Slice is performed on the dimension Time = "Q1".



OLAP Dice operation [1M]

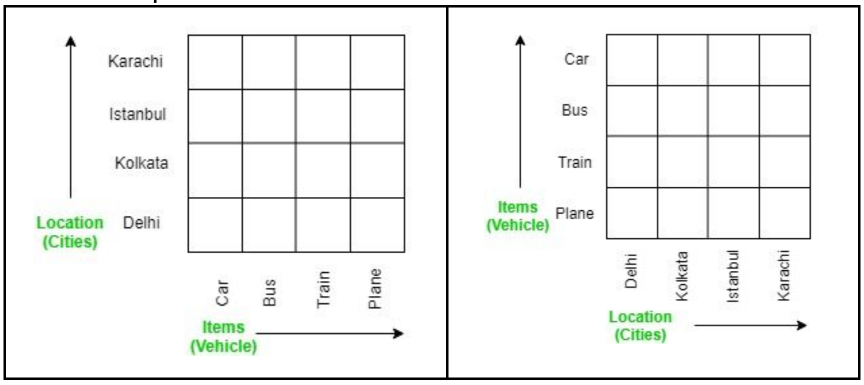
Dice operation defines a subcube by performing a selection on two or more dimensions. Example a sub-cube is selected by selecting following dimensions with criteria:

- Location = “Delhi” or “Kolkata”
- Time = “Q1” or “Q2”
- Item = “Car” or “Bus”



All images [1M]

OLAP Pivot operation
It is a visualization operation that rotates axes in view in order to provide an alternative presentation of the data. Example -



2-a) Explain basic timestamp ordering protocol and compare it with 2 phase locking protocol in terms of deadlock and rollbacks. [10M]

Basic Timestamp Ordering Protocol [6M]
The idea for this scheme is to enforce the equivalent serial order on the transactions based on their timestamps. To do this each database item X is associated two timestamp (TS) values:

1. **read_TS(X) :** The read timestamp of item X is the largest timestamp among all the timestamps of transactions that have successfully read item X—that is, read_TS(X) = TS(T), where T is the youngest transaction that has read X successfully.
2. **write_TS(X) :** The write timestamp of item X is the largest of all the timestamps of transactions that have successfully written item X—that is, write_TS(X) = TS(T), where T is the youngest transaction that has written X successfully.

Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with read_TS(X) and write_TS(X) to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp.

Case 1: Write request
Whenever a transaction T issues a write_item(X) operation, the following check is performed:

- If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with a timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering.

- If write_TS(X) ≤ TS(T) and read_TS(X) ≤ TS(T), then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

Case 2: Read request
Whenever a transaction T issues a read_item(X) operation, the following check is performed:
- If write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of item X before T had a chance to read X.
- If write_TS(X) ≤ TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Example [2M]
Schedule: $r_3(X)$, $r_2(X)$, $w_3(X)$, $r_1(X)$, $w_1(X)$;

	TS(T)	read_TS(X)	write_TS(X)	allowed
		0	0	
$r_3(X)$	3	3	0	As write_TS(X) ≤ TS(T) i.e. (0≤3) allow read and change read_TS(X) = max[3,0] = 3
$r_2(X)$	2	3	0	As write_TS(X) ≤ TS(T) i.e. (0≤2) allow read and change read_TS(X) = max[3,2] = 3
$w_3(X)$	3	3	3	As write_TS(X) ≤ TS(T) and read_TS(X) ≤ TS(T) i.e. 0≤3 and 3≤3 allow write and change write_TS(X) = 3
$r_1(X)$	1			As write_TS(X) > TS(T) i.e. (3>1) roll back T_1 and reject the operation

The given schedule does not follow timestamp ordering protocol.

Comparison [2M]

basic timestamp ordering protocol	2 phase locking protocol
Basic timestamp protocol ensures freedom from deadlock as no transaction ever waits.	As 2PL uses locks and if requested lock is not compatible then transaction must wait which may lead to deadlock.
Cascading rollback is possible.	Cascading Rollback is possible.

2-b) Explain Mandatory Access Control and Discretionary Access Control also explain access control list and access control entry w.r.t. the same. [10M]

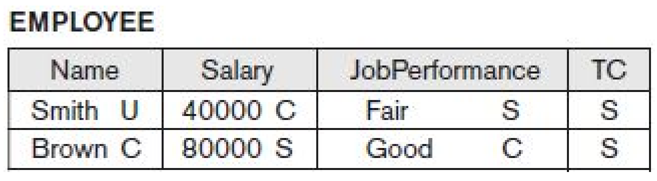
Mandatory Access Control [5M]
In many applications, an additional security policy is needed that classifies data and users based on security classes. This approach, known as mandatory access control (MAC). Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest.

The commonly used model for multilevel security, known as the Bell-LaPadula model classifies each subject (user, account, program) and object (relation, tuple, column, view, operation) into one of the security classifications TS, S, C, or U. Two restrictions are enforced on data access based on the subject/object classifications:

- A subject S is not allowed read access to an object O unless class(S) ≥ class(O). This is known as the simple security property.
- A subject S is not allowed to write an object O unless class(S) ≤ class(O). This is known as the star property (or *-property).

Hence, each attribute A is associated with a classification attribute C in the schema, and each attribute value in a tuple is associated with a corresponding security classification. In addition, in some models, a tuple classification attribute TC is added to the relation attributes to provide a classification for each tuple as a whole.

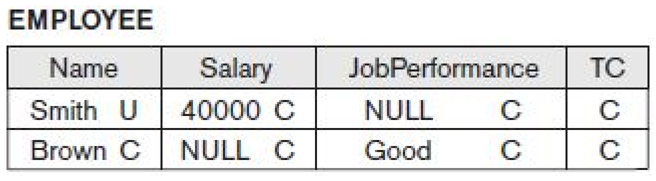
Example
A user with security clearance S would see the same relation shown in Figure (a), since all tuple classifications are less than or equal to S. However, a user with security clearance C would not be allowed to see the values for Salary of 'Brown' and Job_performance of 'Smith', since they have higher classification. The tuples would be filtered to appear as shown in Figure (b), with Salary and Job_performance appearing as null.



EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	Fair S	S
Brown C	80000 S	Good C	S

a) The original EMPLOYEE tuples.



EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	NULL C	C
Brown C	NULL C	Good C	C

b) Appearance of EMPLOYEE after filtering for classification C users.

Access-control list
In the above example the object is an attribute of relation EMPLOYEE and the subject will be the user with corresponding classification level.

Access control entry
Each tuple with the corresponding classification level for each attribute and the entire tuple itself.

Discretionary Access Control [5M]
The typical method of enforcing discretionary access control in a database system is based on the granting and revoking of privileges. There are two levels for assigning privileges to use the database system:

1. The account level: At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
2. The relation (or table) level: At this level, the DBA can control the privilege to access each individual relation or view in the database.

The owner of a relation is given all privileges on that relation. The owner account holder can pass privileges on any of the owned relations to other users by granting privileges to their accounts. In SQL, the following types of privileges can be granted on each individual relation R:

- SELECT (retrieval or read) privilege on R. Gives the account retrieval privilege. In SQL, this gives the account the privilege to use the SELECT statement to retrieve tuples from R.
- Modification privileges on R: This gives the account the capability to modify the tuples of R. In SQL, this includes three privileges: UPDATE, DELETE, and INSERT.

- References privilege on R: This gives the account the capability to reference (or refer to) a relation R when specifying integrity constraints. This privilege can also be restricted to specific attributes of R.

GRANT Command
GRANT command is used to provide access or privileges on the database object to the users.

Syntax
```
GRANT ALL | SELECT | DELETE | UPDATE
ON table | view
TO user_name | PUBLIC | role_name
[WITH GRANT OPTION];
```

Example
```
GRANT SELECT, DELETE
ON employee
TO xyz;
```

REVOKE Command
The REVOKE command removes user access rights or privileges to the database objects.

Syntax
```
REVOKE ALL | SELECT | DELETE | UPDATE
ON table | view
FROM user_name | PUBLIC | role_name;
```

Example
```
REVOKE DELETE
ON employee
FROM xyz;
```

3-a) Why fragmentation is required in distributed databases. Explain vertical fragmentation with example, comment on completeness, reconstruction, disjointness aspect of it. [10M]

Reasons for Fragmentation [2M]
- The decomposition of a relation into fragments, each being treated as a unit, permits a number of transactions to execute concurrently.
- The fragmentation of relations typically results in the parallel execution of a single query by dividing it into a set of subqueries that operate on fragments. Thus fragmentation typically increases the level of concurrency and therefore the system throughput.

Vertical Fragmentation [1M]
Vertical fragmentation divides a relation “vertically” by columns. A vertical fragment of a relation keeps only certain attributes of the relation. It is necessary to include the primary key or unique key attribute in every vertical fragment so that the full relation can be reconstructed from the fragments.

Example [2M]
Figure shows the PROJ relation partitioned vertically into two subrelations, $PROJ_1$ and $PROJ_2$. $PROJ_1$ contains only the information about project budgets, whereas $PROJ_2$ contains project names and locations. It is important to notice that the primary key of the relation (PNO) is included in both fragments.

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

$PROJ_1$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris

Figure: Example of Vertical fragmentation

Comment on correctness rules of fragmentation [5M]
Correctness rules (completeness, reconstruction, disjointness) ensure that the database does not undergo semantic change during fragmentation.

Completeness
If a relation instance R is decomposed into fragments then each data item that can be found in R can also be found in one or more of fragments. Note that in the case of horizontal fragmentation, the “item” typically refers to a tuple, while in the case of vertical fragmentation, it refers to an attribute.

For the vertical fragmentation example given above each attribute found in PROJ can also be found in $PROJ_1$ or $PROJ_2$.

Reconstruction
If a relation R is decomposed into fragments then it should be possible to define a relational operator such that the entire relation R can be reconstructed from it.

Reconstruction of a global relation PROJ from its fragments $PROJ_1$ and $PROJ_2$ is performed by the join operator. $PROJ = PROJ_1 \bowtie PROJ_2$

Disjointness
This rule ensures that no record/attribute will become a part of two or more different fragments during the fragmentation process. This criterion ensures that the horizontal fragments are disjoint. If relation R is vertically decomposed, its primary key attributes are typically repeated in all its fragments (for reconstruction). Therefore, in the case of vertical partitioning, disjointness is defined only on the non-primary key attributes of a relation.

For the vertical fragmentation example given above non-primary key attributes of $PROJ_1$ and $PROJ_2$ are disjoint.

3-b) Explain 2 Phase commit protocol with proper flow diagram. [10M]

Overview [1M]
Two-phase commit (2PC) is a very simple and elegant protocol that ensures the atomic commitment of distributed transactions. It extends the effects of local atomic commit actions to distributed transactions by insisting that all sites involved in the execution of a distributed transaction agree to commit the transaction before its effects are made permanent.

Flow diagram [3M]

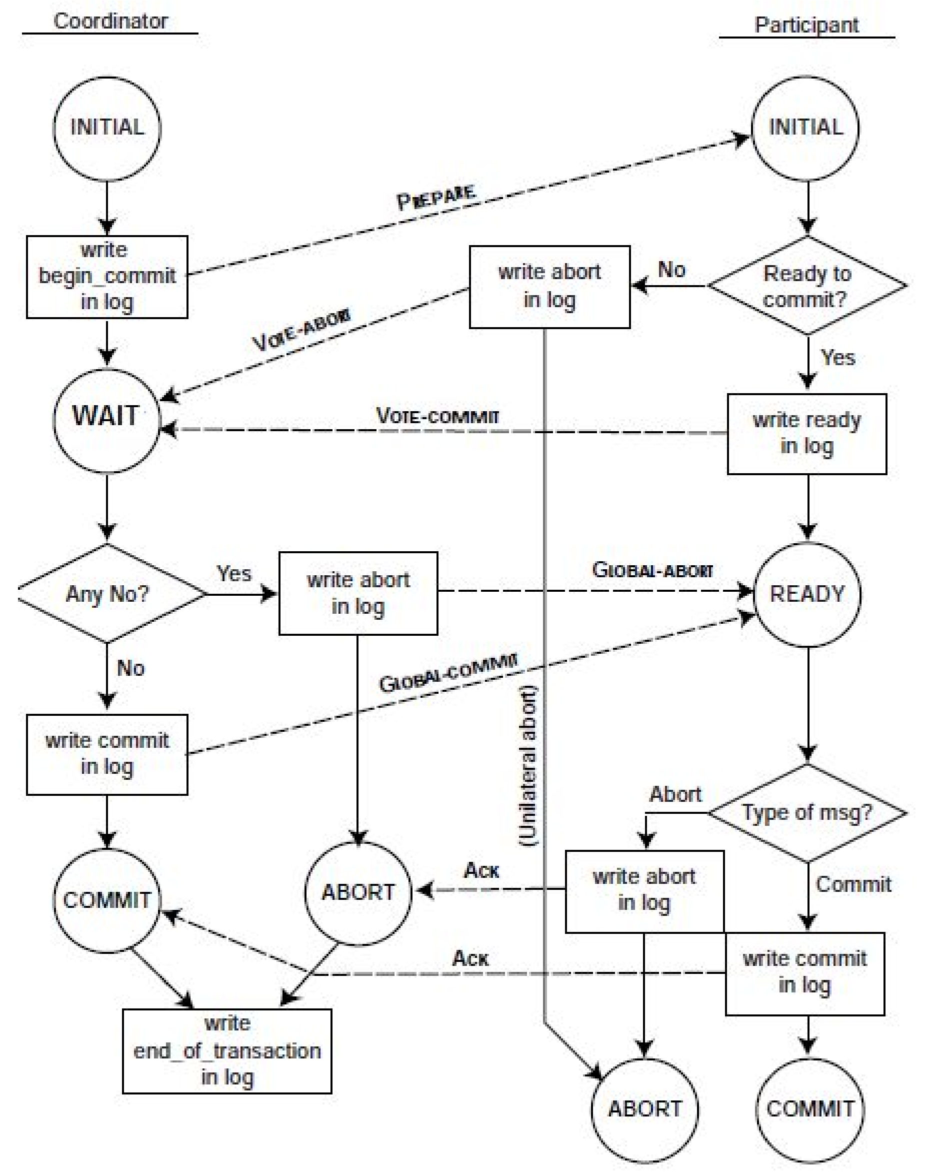


Figure: Two phase commit protocol flow diagram
Circles => states
Dashed lines => messages between the coordinator and the participants

Protocol description [5M]

1. Initially, the coordinator writes a begin commit record in its log, sends a “prepare” message to all participant sites, and enters the WAIT state.
2. When a participant receives a “prepare” message, it checks if it could commit the transaction. If so, the participant writes a ready record in the log, sends a “vote-commit” message to the coordinator, and enters READY state; otherwise, the participant writes an abort record and sends a “vote-abort” message to the coordinator.
3. If the decision of the site is to abort, it can forget about that transaction, since an abort decision serves as unilateral abort.
4. After the coordinator has received a reply from every participant, it decides whether to commit or abort the transaction. If even one participant has registered a negative vote, the coordinator has to abort the transaction globally. So it writes an abort record, sends a “global-abort” message to all participant sites, and enters the ABORT state; otherwise, it writes a commit record, sends a “global-commit” message to all participants, and enters the COMMIT state.
5. The participants either commit or abort the transaction according to the coordinator’s instructions and send back an acknowledgment, at which point the coordinator terminates the transaction by

writing an end of transaction record in the log.

Global commit rule [1M]

- If even one participant votes to abort the transaction, the coordinator has to reach a global abort decision.
- If all the participants vote to commit the transaction, the coordinator has to reach a global commit decision.

4-a) Explain MOLAP, ROLAP and HOLAP models. [10M]

These are the variations relating to the way data is stored for OLAP (OnLine Analytical Processing).

MOLAP [4M]

Multidimensional OLAP tools utilize a pre-calculated dataset (data cube) or optimized multidimensional array storage that contains summarized data. In MLAP data is stored in a multidimensional cube. The storage is not in the relational database, but in proprietary formats. MOLAP tools feature very fast response and the ability to quickly write back data into the database.

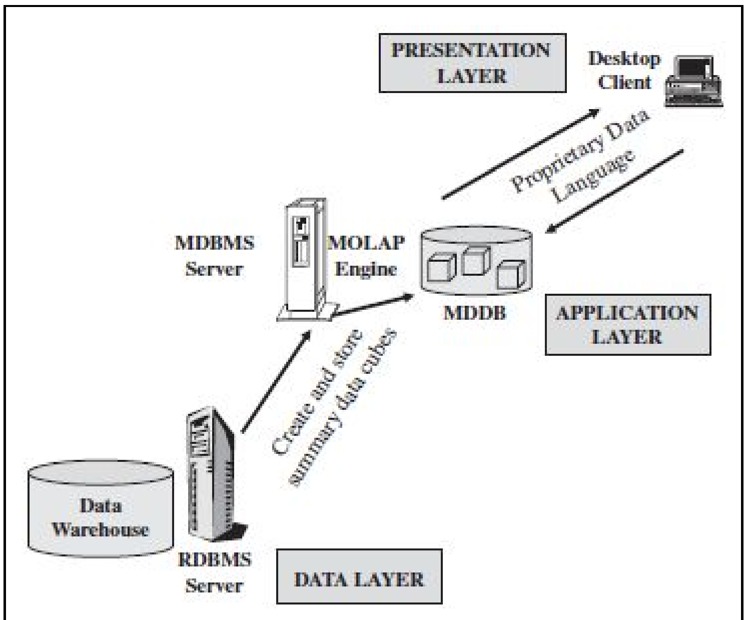


Figure: MOLAP model

Primary downside of MOLAP tools are:

- Limited scalability as the cubes get very big when you start to add dimensions and more detailed data.
- Inability to contain detailed data as you are forced to use summary data unless your dataset is very small and high load time of the cubes.

ROLAP [4M]

Relational Online Analytical Processing (ROLAP) tools do not use precalculated data cubes. Instead, they intercept the query and the question to the standard relational database and bring back the data required to answer the question.

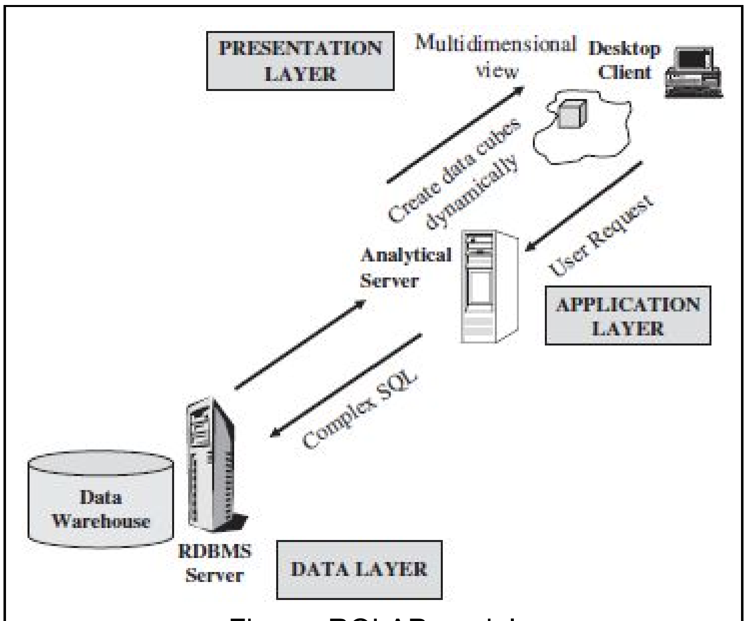


Figure: ROLAP model

ROLAP tools feature the ability to ask any question (you are not limited to the contents of a cube) and the ability to drill down to the lowest level of detail in the database. This methodology relies on manipulating the data stored in the relational database to give the appearance of traditional OLAP's slicing and dicing functionality. In essence, each action of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement. ROLAP tools are best suited for users who deal with "unbounded" problems

With ROLAP, it is possible to create additional database files - summary tables or aggregations which summarize the data at any desired combination of dimensions. Primary downsides of ROLAP tools tools are slow response and some limitations on scalability.

HOLAP [2M]
Hybrid OLAP (HOLAP) addresses the shortcomings of ROLAP and MOLAP by combining the capabilities of both approaches. HOLAP tools can utilize both pre-calculated cubes and relational data sources. Thus, in a way, HOLAP technologies combine the advantages of MOLAP and ROLAP.

When summary information is required, HOLAP leverages cube technology for faster performance and when detailed information is needed, HOLAP can "drill through" from the cube into the underlying relational data to satisfy the user's need for information. The OLAP server has the task of figuring out where to access the data. In the case of aggregated data requests, the information is sought from the MDDB cube while granular queries are redirected to the relational database.

DOLAP (Desktop OLAP)
In this technology, multidimensional cubes are formed and downloaded on the user's desktop. Here, datasets are limited to the boundaries defined by user with no access to granular data. Depending on the tool selected for the data warehouse users, either ROLAP, MOLAP or a combination of both can be implemented. However, it is always best to start with ROLAP architecture for large voluminous data and then later on move in to MOLAP cubes when aggregate data is needed.

With Web-based access on the increase to encompass as many users as possible, Desktop OLAP (DOLAP) is becoming increasingly popular. DOLAP server just performs the task of sending a cube of the requested data to the client like that of a Java applet which resides on the client machine and enables viewing, analysis and computation of this data. Such client applets download MOLAP cubes on the user's desktop so that reports can be generated. Linking cubes to related cubes or sharing them with users across the network can be easily achieved with DOLAP servers. In addition to this, Web-based architectures can be further enhanced to include OLAP caching servers to improve query performance in a data warehousing environment. Then, parallel processing can also be implemented to improve query performance even to a greater extent.

4-b) What is the significance of serializability, explain conflict serializability and view serializability with the help of example. [10M]

Let operations of T1 = {O11, O12, O13} and T2 = {O21, O22}

Some terms defined:

- Schedule: A chronological order (sequence) in which operations of the transactions are executed.
- Serial schedule: A schedule in which all the operations of one transaction are executed completely before starting another transaction. Example, S: O11, O12, O13, O21, O22;
- Non-serial schedule: A schedule where the operations of the transactions are interleaved. Example, S’: O11, O21, O12, O22, O13;
- Serializable schedule: Non-serial schedule equivalent to some serial schedule.

Significance of serializability [2M]
When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state. Serializability is a concept that helps us to check which schedules are serializable. A serializable schedule is the one that always leaves the database in a consistent state.

Conflict serializability [4M]

Conflicting operations
Two operations are said to be conflicting if all of the following conditions are satisfied:

- They belong to different transactions
- They operate on the same data item
- At least one of them is a write operation

If two operations conflict then their order must be preserved in the schedule. It implies that if two operations do not conflict then their order can be swapped.

Conflict equivalent
We say that schedule S is conflict equivalent to schedule S’ if conflicting operations order is preserved in both the schedules. Example, Let S1: r1(X), w1(X), r2(X), w2(X), r1(Y), w1(Y);

In the schedule S1 highlighted operations are non-conflicting and hence can be swapped, creating new schedule S2: r1(X), w1(X), r2(X), r1(Y), w2(X), w1(Y); Here we say S1 and S2 are conflict equivalent. We can go on swapping non-conflicting operations as shown below:
S3: r1(X), w1(X), r1(Y), r2(X), w2(X), w1(Y);
S4: r1(X), w1(X), r1(Y), r2(X), w1(Y), w2(X);
S5: r1(X), w1(X), r1(Y), w1(Y), r2(X), w2(X); (Serial schedule)

Note that S5 is a serial schedule T1 → T2. It implies S1 is conflict equivalent to S5 which is a serial schedule. We say that S1 is conflict serializable.

Conflict serializable

A schedule is conflict serializable if it is conflict equivalent to some serial schedule.

To check if the given schedule is conflict serializable or not is called conflict serializability. Which in turn ensures that the schedule will always give consistent (correct) state of the database.

View serializability [4M]
Consider two schedules S and S', where the same set of transactions participates in both schedules. The schedules S and S' are said to be view equivalent if three conditions are met:

1. **Same initial read:** For each data item Q, if transaction Ti reads the initial value of Q in schedule S, then transaction Ti must, in schedule S', also read the initial value of Q.
2. **Same read after write:** For each data item Q, if transaction Ti executes read(Q) in schedule S, and if that value was produced by a write(Q) operation executed by transaction Tj, then the read(Q) operation of transaction Ti must, in schedule S', also read the value of Q that was produced by the same write(Q) operation of transaction Tj.
3. **Same final write:** For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S'.

The concept of view equivalence leads to the concept of view serializability. We say that a schedule S is view serializable if it is view equivalent to a serial schedule. Example

T1	T2	T3
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

Figure: Schedule S4

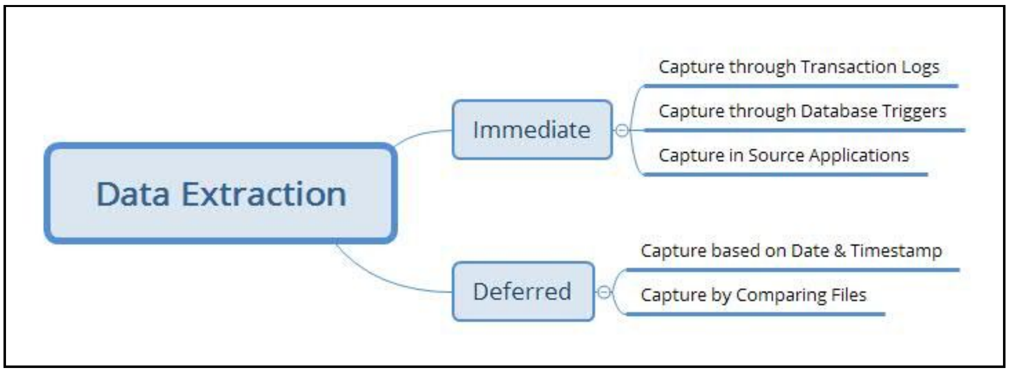
Above schedule S4 is view equivalent to the serial schedule T1, T2, T3; since the one read(Q) instruction reads the initial value of Q in both schedules and T3 performs the final write of Q in both schedules.

Note: Every conflict-serializable schedule is also view serializable, but there are view-serializable schedules that are not conflict serializable. Schedule S4 is not conflict serializable, since every pair of consecutive instructions conflicts, and, thus, no swapping of instructions is possible.

5-a) Explain types of data extraction methods in ETL process. [10]

Data Extraction methods can be broadly classified into two categories:

1. Immediate data extraction
2. Deferred data extraction


Data Extraction
Immediate
Capture through Transaction Logs
Capture through Database Triggers
Capture in Source Applications
Deferred
Capture based on Date & Timestamp
Capture by Comparing Files

Immediate Data Extraction [2+2+2 = 6M]
It is real-time and occurs as the transactions happen at the source databases and files. Options:
1. Capture through Transaction Logs
2. Capture through Database Triggers
3. Capture in Source Applications

Capture through Transaction Logs
This option uses the transaction logs of the DBMSs maintained for recovery from possible failures. As each transaction adds, updates, or deletes a row from a database table, the DBMS immediately writes entries in the log file. This data extraction technique reads the transaction log and selects all the committed transactions.

Capture through Database Triggers
Triggers are special stored procedures (programs) that are stored on the database and fired when certain predefined events occur. We can create trigger programs for all events for which we need data to be captured. The output of the trigger programs is written to a separate file that will be used to extract data for the data warehouse.

Capture in Source Applications
The source application is made to assist in the data capture for the data warehouse. We have to modify the relevant application programs that write to the source files and databases. We revise the programs to write all adds, updates, and deletes to the source files and database tables. Then other extract programs can use the separate file containing the changes to the source data.

Deferred Data Extraction [2+2 = 4M]
Deferred data extraction do not capture the changes in real time. The capture happens later. Options:
1. Capture Based on Date and Time Stamp
2. Capture by Comparing Files

Capture Based on Date and Time Stamp
Every time a source record is created or updated it may be marked with a stamp showing the date and time. The time stamp provides the basis for selecting records for data extraction. Here the data capture occurs at a later time, not while each source record is created or updated. We run the data extraction program at the desired date and time.

Capture by Comparing Files
This technique is also called the snapshot differential technique because it compares two snapshots of the source data. Here we compare two snapshots of a file and then capture any changes between the two copies. This technique necessitates the keeping of prior copies of all the relevant source data.

5-b) What is basic difference between pessimistic and optimistic concurrency control algorithm. Explain distributed 2PL algorithm [10M].

Difference [2M]

Pessimistic concurrency control algorithm	Optimistic concurrency control algorithm
Based on assumption that possibility of conflicts when updating a record is more.	Based on the assumption that the possibility of conflicts is very less.
Protects system from concurrency conflict so it will not happen.	Allow concurrency conflict happens and if it happens then resolve it.

Locks records so that another user cannot update it at the same time.	Doesn’t lock records.
Best solution when conflicts are high.	Best solution when conflicts are low.
More complex in designing and managing.	Simple in designing and programming

Distributed 2PL Algorithm Overview [1M]
Distributed 2PL requires the availability of lock managers at each site. The communication between cooperating sites that execute a transaction according to the distributed 2PL protocol is depicted in Figure.

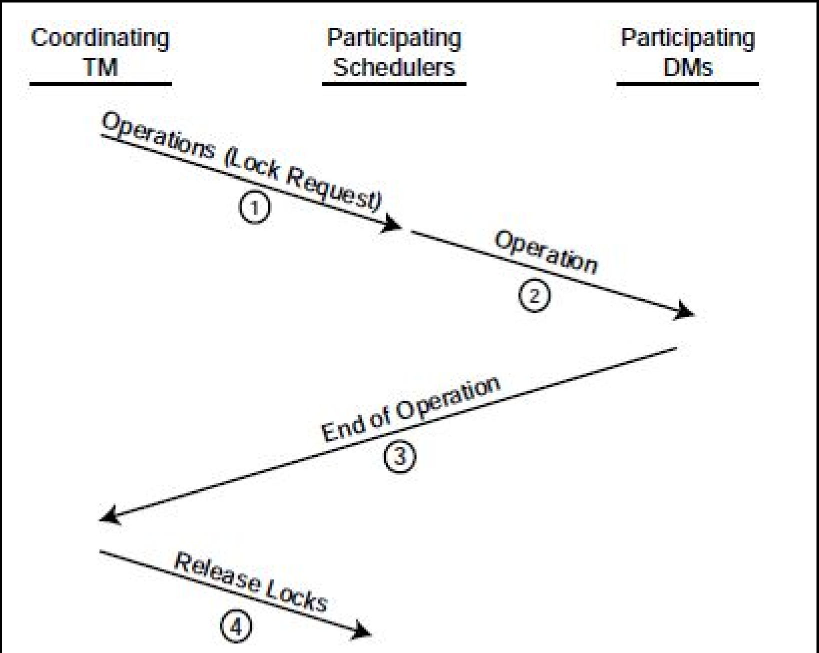


Figure. Communication Structure of Distributed 2PL
TM - Transaction Manager; DM/DP - data processor

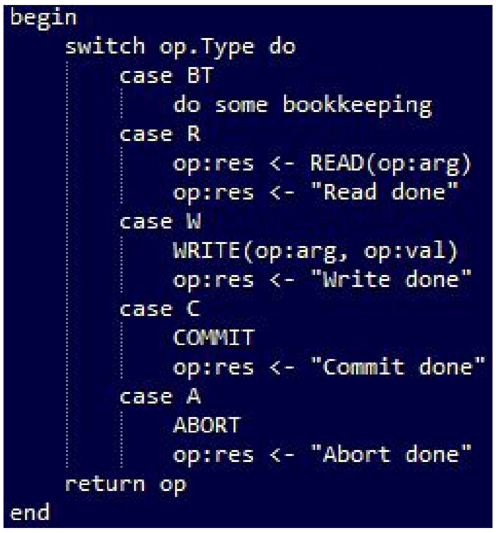
1. The messages are sent to the lock managers at all participating sites in D2PL-TM.
2. The operations are passed to the data processors by the participating lock managers. The coordinating transaction manager does not wait for a “lock request granted” message.
3. The participating data processors send the “end of operation” messages to the coordinating TM.
4. Coordinating TM release locks.

Terms used in the algorithm [1M]
The operation that is defined as a 5-tuple, Op :
1. Type = {BT, R, W, A, C};
   where BT = Begin transaction, R = Read, W = Write, A = Abort, and C = Commit
2. arg : Data item that the operation accesses
   (reads or writes; for other operations this field is null)
3. val : Value; also used in case of Read and Write operations to specify the value that has been read or the value to be written for data item arg (otherwise it is null)
4. tid : Transaction identifier of the transaction that this operation belongs to
5. res : Result; indicates the completion code of operations requested of DP (data processor)

The transaction manager (D2PL-TM) algorithm is written as a process that runs forever and waits until a message arrives from either an application (with a transaction operation) or from a lock manager, or from a data processor. The lock manager (D2PL-LM) and data processor (DP) algorithms are written as procedures that are called when needed.

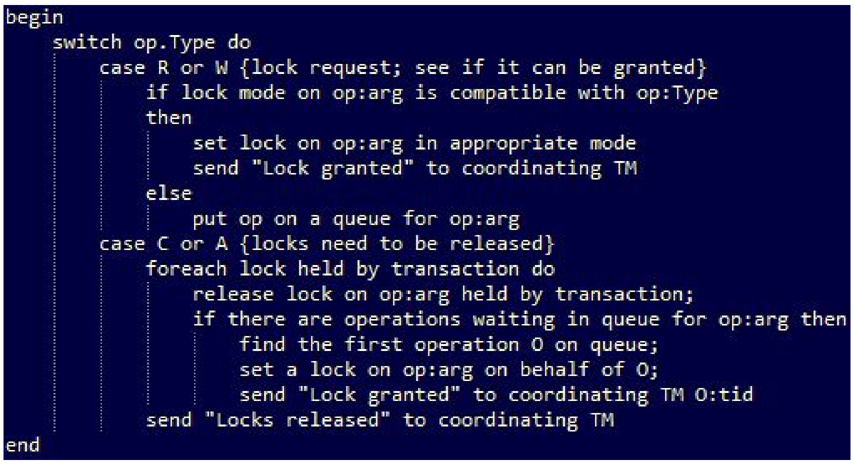
Algorithms [6M]

Data Processor (DP) Algorithm
Input: op : Op



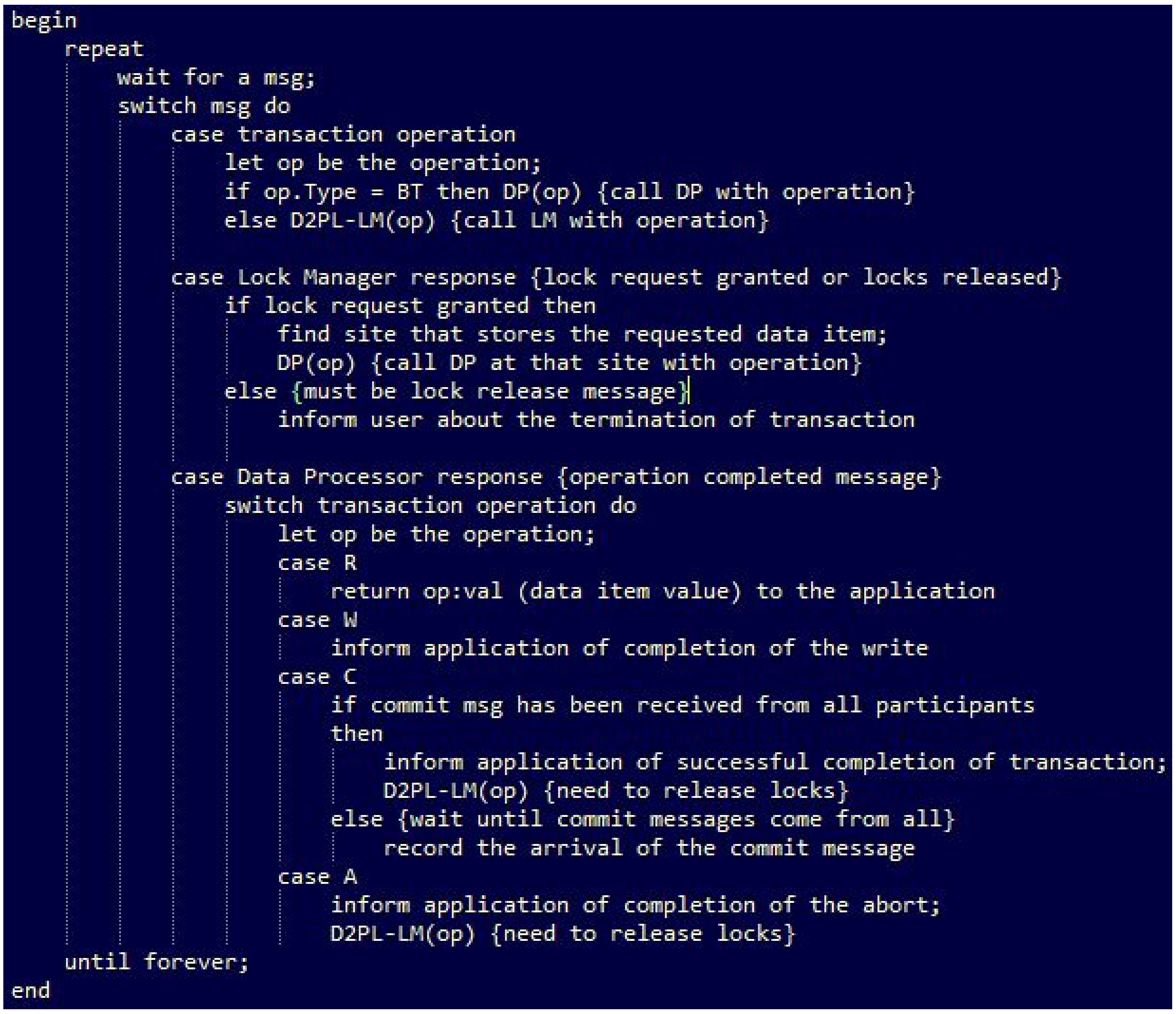
```
begin
    switch op.Type do
        case BT
            do some bookkeeping
        case R
            op:res <- READ(op:arg)
            op:res <- "Read done"
        case W
            WRITE(op:arg, op:val)
            op:res <- "Write done"
        case C
            COMMIT
            op:res <- "Commit done"
        case A
            ABORT
            op:res <- "Abort done"
    return op
end
```

Distributed 2PL Lock Manager (D2PL-LM) Algorithm
Input: op : Op



```
begin
    switch op.Type do
        case R or W {lock request; see if it can be granted}
            if lock mode on op:arg is compatible with op:Type
            then
                set lock on op:arg in appropriate mode
                send "Lock granted" to coordinating TM
            else
                put op on a queue for op:arg
        case C or A {locks need to be released}
            foreach lock held by transaction do
                release lock on op:arg held by transaction;
                if there are operations waiting in queue for op:arg then
                    find the first operation O on queue;
                    set a lock on op:arg on behalf of O;
                    send "Lock granted" to coordinating TM O:tid
            send "Locks released" to coordinating TM
end
```

Distributed 2PL Transaction Manager (D2PL-TM) Algorithm
Input: msg : a message



```
begin
    repeat
        wait for a msg;
        switch msg do
            case transaction operation
                let op be the operation;
                if op.Type = BT then DP(op) {call DP with operation}
                else D2PL-LM(op) {call LM with operation}

            case Lock Manager response {lock request granted or locks released}
                if lock request granted then
                    find site that stores the requested data item;
                    DP(op) {call DP at that site with operation}
                else {must be lock release message}
                    inform user about the termination of transaction

            case Data Processor response {operation completed message}
                switch transaction operation do
                    let op be the operation;
                    case R
                        return op:val (data item value) to the application
                    case W
                        inform application of completion of the write
                    case C
                        if commit msg has been received from all participants
                        then
                            inform application of successful completion of transaction;
                            D2PL-LM(op) {need to release locks}
                        else {wait until commit messages come from all}
                            record the arrival of the commit message
                    case A
                        inform application of completion of the abort;
                        D2PL-LM(op) {need to release locks}
    until forever;
end
```

Reference: [3] - pg. 374-377

6-a) Role Based Access Control [10M]

Explanation [1M]
Role-based access control (RBAC) is a proven technology for managing and enforcing security in large-scale enterprise-wide systems. Its basic notion is that privileges and other permissions are associated with organizational roles rather than with individual users. Individual users are then assigned to appropriate roles.

Commands [2M]
To create a role:
```
CREATE ROLE role_name;
```

To destroy a role:
```
DESTROY ROLE role_name;
```

The GRANT and REVOKE commands can then be used to assign and revoke privileges from roles, as well as for individual users when needed.

RBAC combined with DAC and MAC [1M]
RBAC can be used with traditional discretionary and mandatory access controls; it ensures that only authorized users in their specified roles are given access to certain data or resources.

Separation of duties - mutual exclusion of roles [2M]
Separation of duties is needed to prevent one user from doing work that requires the involvement of two or more people, thus preventing collision. One method in which separation of duties can be successfully implemented is with mutual exclusion of roles.

Two roles are said to be mutually exclusive if both the roles cannot be used simultaneously by the user. Mutual exclusion of roles can be categorized into two types:
1. Authorization time exclusion (static) - two roles that have been specified as mutually exclusive cannot be part of a user's authorization at the same time.
2. Runtime exclusion (dynamic) - here both these roles can be authorized to one user but cannot be activated by the user at the same time.

Role hierarchy [2M]
The role hierarchy in RBAC is a natural way to organize roles to reflect the organization's lines of authority and responsibility. By convention, junior roles at the bottom are connected to progressively senior roles as one moves up the hierarchy. If a user has one role, the user automatically has roles lower in the hierarchy. Role hierarchy can be implemented in the following manner:
```
GRANT ROLE full_time TO employee_type1
GRANT ROLE intern TO employee_type2
```

Temporal constraints [1M]
Temporal constraints such as the time and duration of role activations and the timed triggering of a role by an activation of another role is possible. Roles can be assigned to workflow tasks so that a user with any of the roles related to a task may be authorized to execute it and may play a certain role only for a certain duration.

RBAC Advantages [1M]
- Flexibility, policy neutrality, better support for security management and administration, and a natural

enforcement of the hierarchical organization structure within organizations.
- Can be used for developing secure Web-based applications.
- Provides mechanisms for addressing the security issues related to the execution of tasks and workflows, and for specifying user-defined and organization-specific policies.

6-b) Query Optimization [10M]

Overview [2M]
Given a query, there are generally a variety of methods for computing the answer. It is the responsibility of the system to transform the query as entered by the user into an equivalent query that can be computed more efficiently. The process of finding a good strategy for processing a query is called query optimization. One aspect of optimization occurs at the relational-algebra level, where the system attempts to find an expression that is equivalent to the given expression, but more efficient to execute. Another aspect is selecting a detailed strategy for processing the query, such as choosing the algorithm to use for executing an operation, choosing the specific indices to use, and so on.

Transformation of Relational Expressions [2M]
The given query is transformed to relational-algebra expression. There are a number of equivalence rules that we can use to transform an expression into an equivalent one. We use these rules to generate systematically all expressions equivalent to the given query.

Each relational-algebra expression represents a particular sequence of operations. The first step in selecting a query-processing strategy is to find a relational-algebra expression that is equivalent to the given expression and is estimated to cost less to execute.

Estimating Statistics of Expression Results [2M]
The strategy that the database system chooses for evaluating an operation depends on the size of each relation and on the distribution of values within columns. So that they can base the strategy choice on reliable information, database systems may store statistics for each relation R. These statistics include:

a. The number of tuples in the relation R.
b. The size of a record (tuple) of relation R in bytes.
c. The number of distinct values that appear in the relation R for a particular attribute.

Most database systems use histograms to store the number of values for an attribute within each of several ranges of values. Histograms are often computed using sampling.

Choice of Evaluation Plans [4M]
Generation of expressions is only part of the query-optimization process, since each operation in the expression can be implemented with different algorithms. An evaluation plan defines exactly what algorithm should be used for each operation, and how the execution of the operations should be coordinated. A cost-based optimizer explores the space of all query-evaluation plans that are equivalent to the given query, and chooses the one with the least estimated cost. We have the following choice of evaluation plans

1. **Cost-Based Join Order Selection** - For a complex join query, the number of different query plans that are equivalent to the query can be large. We can develop a dynamic-programming algorithm for finding optimal join orders.
2. **Cost-Based Optimization with Equivalence Rules** - The benefit of using equivalence rules is that it is easy to extend the optimizer with new rules to handle different query constructs. For example, nested queries can be represented using extended relational-algebra constructs, and transformations of nested queries can be expressed as equivalence rules.
3. **Heuristics** - We use heuristics to reduce the number of plans considered, and thereby to reduce the cost of optimization. Heuristic rules for transforming relational algebra queries include "Perform selection operations as early as possible," "Perform projections early," and "Avoid Cartesian

products."

6-c) Data Warehouse architecture [10M].

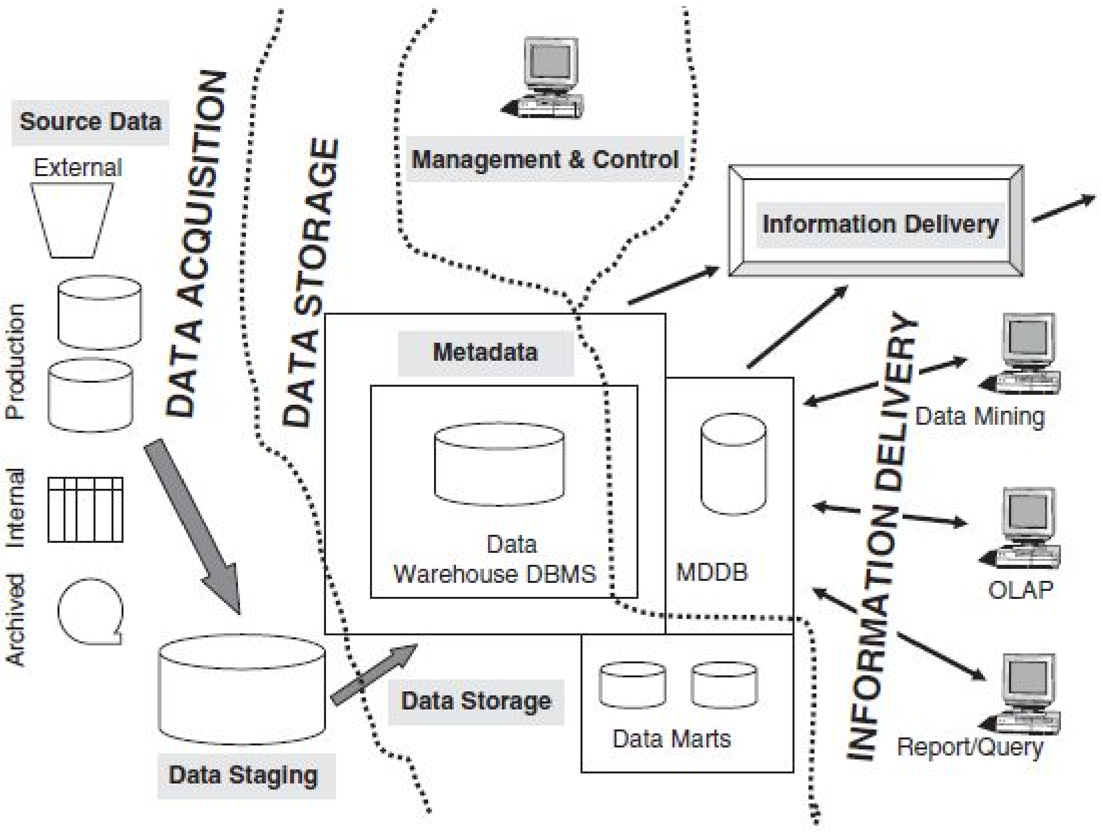


Fig. data warehouse architecture [1M]

Source Data Component [3M]
Source data coming into the data warehouse may be grouped into four broad categories:

1. Production Data - This category of data comes from the various operational systems of the enterprise. These normally include financial systems, manufacturing systems, systems along the supply chain, and customer relationship management systems.
2. Internal Data - In every organization, users keep their "private" spreadsheets, documents, customer profiles, and sometimes even departmental databases. This is the internal data, parts of which could be useful in a data warehouse.
3. Archived Data - Operational systems are primarily intended to run the current business. In every operational system, you periodically take the old data and store it in archived files. The circumstances in your organization dictate how often and which portions of the operational databases are archived for storage. Some data is archived after a year. Sometimes data is left in the operational system databases for as long as five years. Much of the archived data comes from old legacy systems that are nearing the end of their useful lives in organizations.
4. External Data - Most executives depend on data from external sources for a high percentage of the information they use. They use statistics relating to their industry produced by external agencies and national statistical offices. They use market share data of competitors. They use standard values of financial indicators for their business to check on their performance.

Data Staging Component [1M]
After you have extracted data from various operational systems and from external sources, you have to

prepare the data for storing in the data warehouse. The extracted data coming from several disparate sources needs to be changed, converted, and made ready in a format that is suitable to be stored for querying and analysis. Three major functions need to be performed for getting the data ready. You have to extract the data, transform the data, and then load the data into the data warehouse storage. These three major functions of extraction, transformation, and preparation for loading take place in a staging area.

Data Storage Component [1M]
The data storage for the data warehouse is a separate repository. The operational systems of your enterprise support the day-to-day operations. The data repository for a data warehouse, you need to keep large volumes of historical data for analysis. Further, you have to keep the data in the data warehouse in structures suitable for analysis, and not for quick retrieval of individual pieces of information. Therefore, the data storage for the data warehouse is kept separate from the data storage for operational systems.

Most of the data warehouses employ relational database management systems. Many data warehouses also employ multidimensional database management systems. Data extracted from the data warehouse storage is aggregated in many ways and the summary data is kept in the multidimensional databases (MDDBs). Such multidimensional database systems are usually proprietary products.

Information Delivery Component [2M]
In order to provide information to the wide community of data warehouse users, the information delivery component includes different methods of information delivery. Ad hoc reports are predefined reports primarily meant for novice and casual users. Provision for complex queries, multidimensional (MD) analysis, and statistical analysis cater to the needs of the business analysts and power users. Information fed into executive information systems (EIS) is meant for senior executives and high-level managers. Some data warehouses also provide data to data-mining applications. Data-mining applications are knowledge discovery systems where the mining algorithms help you discover trends and patterns from the usage of your data.

Metadata Component [1M]
Metadata in a data warehouse is similar to the data dictionary or the data catalog in a database management system. In the data dictionary, you keep the information about the logical data structures, the information about the files and addresses, the information about the indexes, and so on. The data dictionary contains data about the data in the database.

Management and Control Component [1M]
This component of the data warehouse architecture sits on top of all the other components. The management and control component coordinates the services and activities within the data warehouse. This component controls the data transformation and the data transfer into the data warehouse storage. On the other hand, it moderates the information delivery to the users. It works with the database management systems and enables data to be properly stored in the repositories. It monitors the movement of data into the staging area and from there into the data warehouse storage itself.

The management and control component interacts with the metadata component to perform the management and control functions. As the metadata component contains information about the data warehouse itself, the metadata is the source of information for the management module.

Reference: [1] - Pg. 35-41.

6-d) Challenges in ETL functions [10M]

ETL (Extract, Transform, Load) functions reshape the relevant data from the source systems into useful information to be stored in the data warehouse. Without these functions, there would be no strategic information in the data warehouse. If the source data is not extracted correctly, cleansed, and integrated in the proper formats, query processing and delivery of business intelligence, the backbone of the data warehouse, could not happen. ETL functions are challenging primarily because of the nature of the source systems. Most of the challenges in ETL arise from the disparities among the source operational systems. These challenges are:

1. Source systems are very diverse and disparate.
2. There is usually a need to deal with source systems on multiple platforms and different operating systems.
3. Many source systems are older legacy applications running on obsolete database technologies.
4. Generally, historical data on changes in values are not preserved in source operational systems. Historical information is critical in a data warehouse.
5. Quality of data is dubious in many old source systems that have evolved over time.
6. Source system structures keep changing over time because of new business conditions. ETL functions must also be modified accordingly.
7. Gross lack of consistency among source systems is prevalent. Same data is likely to be represented differently in the various source systems. For example, data on salary may be represented as monthly salary, weekly salary, and bimonthly salary in different source payroll systems.
8. Even when inconsistent data is detected among disparate source systems, lack of a means for resolving mismatches escalates the problem of inconsistency.
9. Most source systems do not represent data in types or formats that are meaningful to the users. Many representations are cryptic and ambiguous.
10. Designing ETL functions is time consuming and arduous.

Reference: [1] - Pg. 283.

Reference Books:

1. Data warehousing fundamentals for IT professionals by Paulraj Ponniah
2. Data warehousing, Reema Thareja
3. Principles of Distributed Database Systems, M. Tamer Özsu, Patrick Valduriez