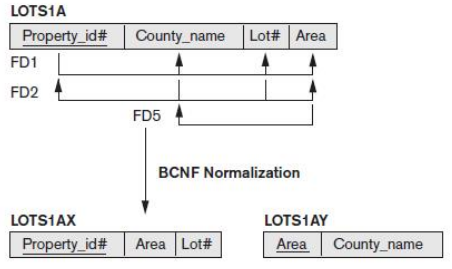
**Q1. In what normal form is the LOTS relation schema in Figure with the respect to the restrictive interpretations of normal form that take only the primary key into account? Will it be in the same normal form if the general definitions of normal form were used? 4.5M**



Answer:

**If we only take the primary key into account, the LOTS relation schema in Figure**

will be in 2NF since there are no partial dependencies on the primary key .However, it is not in 3NF, since there are the following two transitive dependencies onthe primary key:PROPERTY\_ID# ->COUNTY\_NAME ->TAX\_RATE, andPROPERTY\_ID# ->AREA ->PRICE.Now, if we take all keys into account and use the general definition of 2NF and 3NF, theLOTS relation schema will only be in 1NF because there is a partial dependencyCOUNTY\_NAME ->TAX\_RATE on the secondary key {COUNTY\_NAME, LOT#}, which violates 2NF.

Scheme

**the primary key into account? 2M**

**Will it be in the same normal form if the general definitions of normal form were used?2.5M**

**Q2. Suppose that we have the following three tuples in legal instance of a relation schema with three attributes ABC as (1,2,3) (4,2,3) and (5,3,3) . Then which of the following dependencies can you infer does not hold over schema. 4.5M**

Answer:

BC→A

2 3 →1

2 3 →4

3 3 →5

It does not hold.

Scheme

**Identifying dependencies that does not hold 2M**

**Q3. Explain the difference between each of the following:**

**1. Primary versus secondary indexes.**

**2. Dense versus sparse indexes. 3. Clustered versus unclustered indexes.**

**If you were about to create an index on a relation, what considerations would guide your choice with respect to each pair of properties listed above? 8M**

Answer:

1. The main difference between primary and secondary index is that the primary index is an index on a set of fields that includes the primary key for the field and does not contain duplicates, while the secondary index is an index that is not a primary index and which can contain duplicates.
2. Dense index: In a dense index, an index entry appears for every search-key value in the ﬁle. In a dense clustering index, the index record contains the search-key value and a pointer to the ﬁrst data record with that search-key value. The rest of the records with the same search-key value would be stored sequentially after the ﬁrst record, since, because the index is a clustering one, records are sorted on the same search key. In a dense nonclustering index, the index must store a list of pointers to all records with the same search-key value.

Sparse index: In a sparse index, an index entry appears for only some of the search-key values. Sparse indices can be used only if the relation is stored in sorted order of the search key, that is if the index is a clustering index. As is true in dense indices, each index entry contains a search-key value and a pointer to the ﬁrst data record with that search-key value. To locate a record, we ﬁnd the index entry with the largest search-key value that is less than or equal to the search-key value for which we are looking. We start at the record pointed to by that index entry, and follow the pointers in the ﬁle until we ﬁnd the desired record.

1. The difference between Clustered and Nonclustered index in a relational database is one of the most popular SQL interview questions almost as popular as the difference between truncate and delete, primary key vs unique key and correlated vs noncorrelated subqueries. Indexes are a very important concept, it makes your queries run fast and if you compare a SELECT query which uses an indexed column to one who doesn't you will see a big difference in performance. There can be two kinds of indexes in relational database Clustered and Nonclustered indexes. A clustered index determines the physical sorting order of rows in a table similar to entries on yellow pages which are sorted in alphabetical order.

Suppose you have a table Employee, which contains emp\_id as primary key than a clustered index which is created on a primary key will sort the Employee table as per emp\_id. That was a brief introduction of What is clustered index in SQL.

On another hand, the Non-Clustered index involves one extra step which points to the physical location of the record. In this SQL Interview question, we will see some more differences between clustered and nonclustered index in point format.

Scheme

1. **3M**
2. **3M**
3. **2M**

**Q4. Consider a view branch\_cust created on a bank database as follows:create view branch\_cust asselect branch name, customer name from depositor, account where depositor.account number = account.account number branch(branch name, branch city,assets)customer (customer name, customer street, cust city )loan (loan number, branch name, amount)borrower(customer name, loan number)account (account number, branch name, balance )depositor (customer name, account number)Suppose that the view is materialized; that is, the view is computed and stored. Write triggers to maintain the view, that is, to keep it up-to-date on insertions to and deletions from depositor or account. Do not bother about updates. 8M**

Answer: For inserting into the materialized view branch-cust we must set a database trigger on an insert into depositor and account. We assume that the database system uses immediate binding for rule execution. Further, assume that the current version of a relation is denoted by the relation name itself, while the set of newly inserted tuples is denoted by qualifying the relation name with the prefix–inserted. The active rules for this insertion are given below–define trigger insert into branch-cust via depositor after insert on depositor referencing new table as inserted for each statement insert into branch-cust select branch-name, customer-name from inserted, account where inserted .account-number=account.account-numberdefine trigger insert into branch-custvia account after insert on account referencing new table asinserted for each statement insert into branch-cust select branch-name, customer-namefromdepositor,insertedwheredepositor.account-number=inserted.account-number.

Note that if the execution binding was deferred (instead of immediate), then the result of the join of the set of new tuples of account with the set of new tuples of have been inserted by both active rules, leading to duplication of the corresponding tuples in branch-cust. deletion of a tuple from branch-cust is similar to insertion, except that a deletion from either depositoror account will cause the natural join of these relations to have a lesser number of tuples. We denote the newly deleted set of tuples by qualifying the relation name with the keyword deleted. define trigger delete from branch-cust via depositor after delete on depositor referencing old table as deleted for each statement delete from branch-cust select branch-name, customer-namefromdeleted,accountwheredeleted.account-number=account.account-numberdefine trigger delete from branch-cust viaa ccount after delete on account referencing old table as deleted for each statement delete from branch-cust select branch-name, customer-namefromdepositor,deletedwheredepositor.account-number=deleted.account-number

Scheme

**Trigger 8M**

**Q5. Consider the following relation:R (Doctor#, Patient#, Date, Diagnosis, Treat\_code, Charge)In this relation, a tuple describes a visit of a patient to a doctor along with a treatment code and daily charge. Assume that diagnosis is determined (uniquely) for each patient by a doctor. Assume that each treatment code has a fixed charge (regardless of patient). Is this relation in 2NF? Justify your answer and decompose if necessary. Then argue whether further normalization to 3NF is necessary, and if so, perform it. 12.5M**

Answer: From the question’s text, we can infer the following functional dependencies:

Doctor#, Patient#, Date -> Diagnosis, Treat\_code, Charge

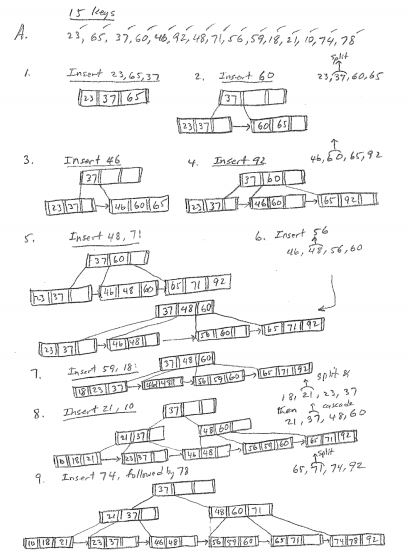
Scheme

**Is this relation in 2NF 6.5M**

**Justify 6M**

**Q6. A PARTS file with Part# as key field includes records with the following Part# values: 23, 65, 37,60, 46, 92, 48, 71, 56, 59, 18, 21, 10, 74, 78, 15, 16, 20, 24, 28, 39, 43, 47, 50, 69, 75, 8, 49, 33, 38.Suppose the search field values are inserted in the given order in a B + -tree of order p=4 and p leaf=3; show how the tree will expand and what the final tree looks** like. **12.5M**

**Answer:**



Scheme

Construction of Tree **125M**

**Q7. Predict whether the given schedules is (conflict) serializable and determine its equivalent serial schedules. r1(X); r3(X);w3(X); w1(X); r2(X); 4.5M**

**Answer:** The above schedule is conflict serializable as two conflicting instructions i.e. r3(X)

and w1 (X) is appearing one after the other . The equivalent serial schedule is as

follows:

The conflicting instructions can be swapped to make it serializable. After swapping

the conflicting instructions we get:

r1 (X); r3 (X); r2 (X); w1 (X); w3 (X);

Scheme

**Prediction** **2.5M**

**Equivalent Serial Schedule 2M**

**Q8. Describe the MapReduce join procedures for Sort-Merge join, Partition Join, N-way Map-side join, and Simple N-way join**

**What is a Join? 4.5M**

Answer:

The join operation is used to combine two or more database tables based on foreign keys. In general, companies maintain separate tables for the customer and the transaction records in their database. And, many times these companies need to generate analytic reports using the data present in such separate tables. Therefore, they perform a join operation on these separate tables using a common column (foreign key), like customer id, etc., to generate a combined table. Then, they analyze this combined table to get the desired analytic reports.

**Joins in MapReduce**

Just like SQL join, we can also perform join operations in MapReduce on different data sets. There are two types of join operations in MapReduce:

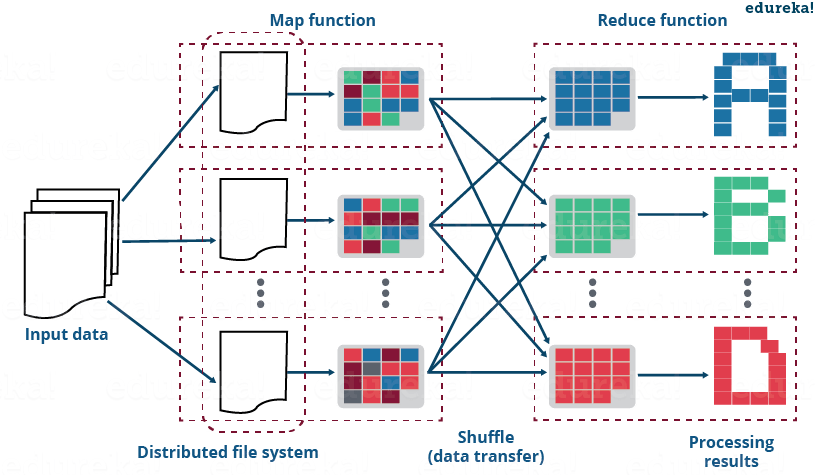
* **Map Side Join:** As the name implies, the join operation is performed in the map phase itself. Therefore, in the map side join, the mapper performs the join and it is mandatory that the input to each map is partitioned and sorted according to the keys.

The map side join has been covered in a separate blog with an example.

* **Reduce Side Join:**As the name suggests, in the reduce side join, the reducer is responsible for performing the join operation. It is comparatively simple and easier to implement than the map side join as the sorting and shuffling phase sends the values having identical keys to the same reducer and therefore, by default, the data is organized for us.

Now, let us understand the reduce side join in detail.

**What is Reduce Side Join?**



As discussed earlier, the reduce side join is a process where the join operation is performed in the reducer phase. Basically, the reduce side join takes place in the following manner:

* Mapper reads the input data which are to be combined based on common column or join key.
* The mapper processes the input and adds a tag to the input to distinguish the input belonging from different sources or data sets or databases.
* The mapper outputs the intermediate key-value pair where the key is nothing but the join key.
* After the sorting and shuffling phase, a key and the list of values is generated for the reducer.
* Now, the reducer joins the values present in the list with the key to give the final aggregated output.

*Meanwhile, you may go through this MapReduce Tutorial video where various MapReduce Use Cases has been clearly explained and practically demonstrated:*

Now, let us take a MapReduce example to understand the above steps in the reduce side join.

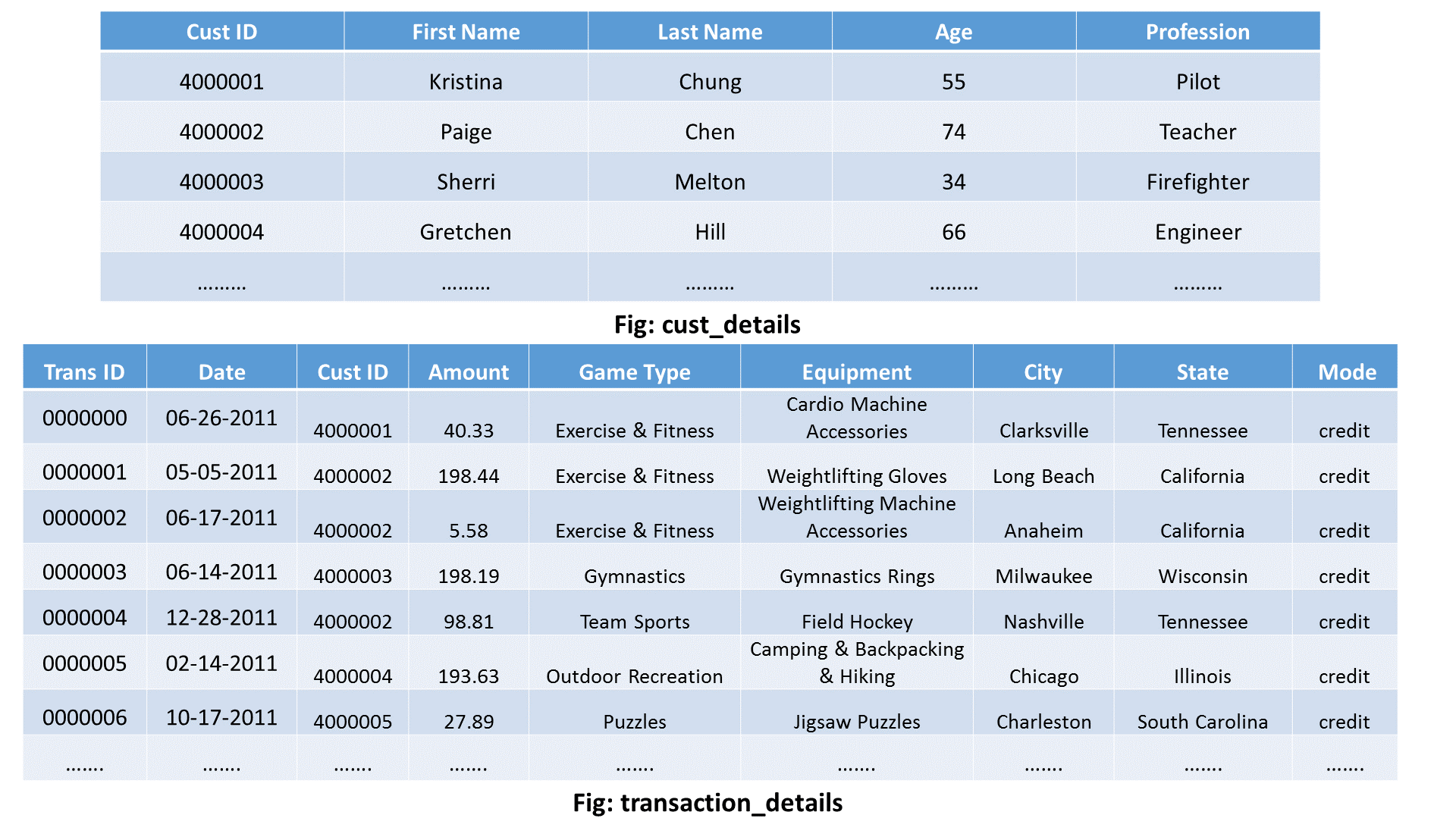
**MapReduce Example of Reduce Side Join**

Suppose that I have two separate datasets of a sports complex:

* **cust\_details:** It contains the details of the customer.
* **transaction\_details:** It contains the transaction record of the customer.

Using these two datasets, I want to know the lifetime value of each customer. In doing so, I will be needing the following things:

* The person’s name along with the frequency of the visits by that person.
* The total amount spent by him/her for purchasing the equipment.



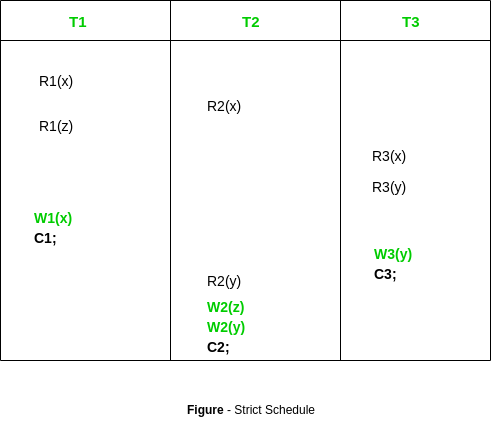
Scheme

**Join** **2.5M**

**Reduce Side Join 2M**

**Q9. Describe the MapReduce join proConsider schedule below. Determine whether the below mentioned schedule is strict, cascadeless, recoverable, or nonrecoverable. Determine the strictest recoverability condition that the schedule satisfies. S3: r1 (X); r2 (Z); r1 (Z); r3 (X);r3 (Y); w1 (X); c1; w3 (Y); c3; r2 (Y);w2 (Z); w2 (Y); c2;cedures for Sort-Merge join, Partition Join, N-way Map-side join, and Simple N-way join 8M**

Answer: In this schedule no read-write or write-write conflict arises before commit hence its strict schedule:



Scheme

**Schedule 8M**

**Q10. Describe the MapReduce join procedures for Sort-Merge join, Partition Join, N-way Map-side join, and Simple N-way join. 8M**

Answer: **What is a Join?**

The join operation is used to combine two or more database tables based on foreign keys. In general, companies maintain separate tables for the customer and the transaction records in their database. And, many times these companies need to generate analytic reports using the data present in such separate tables. Therefore, they perform a join operation on these separate tables using a common column (foreign key), like customer id, etc., to generate a combined table. Then, they analyze this combined table to get the desired analytic reports.

**Joins in MapReduce**

Just like SQL join, we can also perform join operations in MapReduce on different data sets. There are two types of join operations in MapReduce:

* **Map Side Join:** As the name implies, the join operation is performed in the map phase itself. Therefore, in the map side join, the mapper performs the join and it is mandatory that the input to each map is partitioned and sorted according to the keys.

The map side join has been covered in a separate blog with an example.

* **Reduce Side Join:**As the name suggests, in the reduce side join, the reducer is responsible for performing the join operation. It is comparatively simple and easier to implement than the map side join as the sorting and shuffling phase sends the values having identical keys to the same reducer and therefore, by default, the data is organized for us.

Scheme

**Join** **4.5M**

**Reduce Side Join 4M**

**Q11. Consider the three transactions T1, T2, and T3, and the schedules S1 and S2 given below. Draw the serializability (precedence) graphs for S1 and S2, and state whether each schedule is serializable or not. If a schedule is serializable, write down the equivalent serial schedule(s). 12.5M**

**T1: r1 (X); r1 (Z); w1 (X);**

**T2: r2 (Z); r2 (Y); w2 (Z); w2 (Y);**

**T3: r3 (X); r3 (Y); w3 (Y);**

**S1: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); w3 (Y); r2 (Y); w2 (Z); w2 (Y);**

**S2: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); w2 (Z); w3 (Y); w2 (Y);**

**Answer:**

T1, T2, T3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| T1 | T2 | T3

T | | |

I | r1(X) | r2(Z) | r3(X)

M | r1(Z) | r2(Y) | r3(Y)

E | w1(X) | w2(Z) | w3(Y)

| | w2(Y) |

Schedule: S1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| T1 | T2 | T3

| | |

| r1(X) | |

T | | r2(Z) |

I | r1(Z) | |

M | | | r3(X)

E | | | r3(Y)

| w1(X) | |

| | | w3(Y)

| | r2(Y) |

| | w2(Z) |

| | w2(Y) |

Summary: Possible conflicts occur when T1 writes to X when T3 is

still reading X. However T3 does not write to X so this is ok.

T3 Then reads and writes to Y before T2 reads and writes to Y so

this is ok as well. Since T2 reads and writes to Z, it is also ok

that T1 reads Z but does not write. This schedule is serializable

because there are no cycles.

Schedule: S2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| T1 | T2 | T3

| | |

| r1(X) | |

| | r2(Z) |

T | | | r3(X)

I | r1(Z) | |

M | | r2(Y) |

E | | | r3(Y)

| w1(X) | |

| | w2(Z) |

| | w3(Y) |

| | w2(Y) |

Summary: This schedule is non-serializable and contains a major

conflict. Both T2 and T3 are accessing 'Y' when T3 writes to it.

Therefore when T2 writes to 'Y', the transaction for T3 is lost

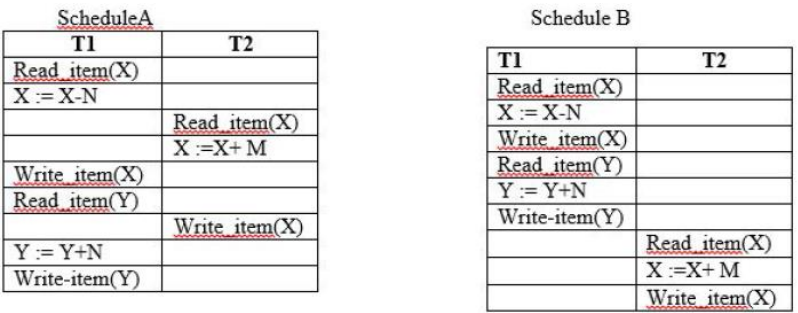
and overridden.

Scheme

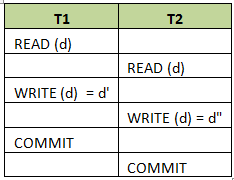
**Precedence Graph 6.5M**

**Equivalent serial schedule 6M**

**Q12. Consider the execution of two transactions T1 and T2 assume that if the initial values of X, Y, M and N are 100, 800, 10, 45 respectively.i. Write the final values of X and Y as per schedule A. Is this a serializable schedule? ii. Write the final values of X and Y for all possible serial schedules as per schedule B.**



Answer: Suppose we have two concurrent transactions T1 and T2, where both are updating data d. Suppose T1 started first and read d for update.  As soon as T1 read d, T2 started and read d for its update. As soon as T2 reads d, T1 updates d to d’. Once T1 is complete, T2 updates d to d”. Here T2 is unaware of T1’s update as it has read the data before T1 has updated it. Similarly, T1 is unaware of T2’s updates. What happens to final result and T1’s update here? Which value of d will be final here – d’ or d” ?



Since T2 is unaware of T1’s update and is processed at the last, the updates done by T1 is lost. The updates done by T2 will only be retained. T1’s update is totally lost and nowhere its symptom of update is kept. This type of update is known as lost update.

But T1’s transaction is valid one and cannot be ignored. Its update is also as important as T2’s. Probably if T1’s update might have changed the result of T2’s update (cases like update is dependent on the value of the column that we are updating – d=d\*10). Hence we cannot lose the data that are being updated by any transactions. This type of lost update can be prevented if these transactions are grouped and executed serially. Suppose T1 is allowed to read and write d, once it completes write then T2 is allowed to read d, then we will have updates done by T1 as well as T2. The first update will however changed by T2, the update of T1 will be stored in undo log or rollback segment. Hence we will know at least there is some value in between transaction begin (here transaction means group of T1 and T2 together) and end of it (end of T2). Such a grouping of transactions and defining the order of execution is known as scheduling or serialization. This type of execution guarantees isolation of transaction. It will not have any dirty reads, non-repeatable reads, deadlocks or lost update issues.

Scheme

**Write the final values of X and Y as per schedule A. 3M**

**Is this a serializable schedule? ii. 6.5M**

**Write the final values of X and Y for all possible serial schedules as per schedule B. 3M**