

INFS3200 REVISION TUTORIAL

Question 1

(a) List all correctness criteria for all types of relational data fragmentation.

Explain the role of each of the listed conditions.

- **Completeness**

Horizontal – all tuples from the original relation are accounted for; as long as the selection predicates are complete, the resulting fragmentation will be complete.

Vertical – each attribute of the original relation **MUST** be assigned to one of the fragments; this is guaranteed by the PARTITION algorithm.

- **Reconstruction**

Horizontal – Reconstruction is performed by the union operator. By applying union to all horizontal fragments resulting from fragmentation of the original relation r , we **MUST** be able to get the original relation as the result.

Vertical - Reconstruction is performed by the join operator. By joining all vertical fragments resulting from fragmentation of the original relation r , we **MUST** be able to get the original relation as the result.

- **Disjointness**

Horizontal - a tuple t from one fragment of relation r may **NOT** appear in any other fragment of relation r ; disjointness is guaranteed as long as the minterm predicates are disjoint.

Vertical – The key attributes are replicated in each fragment in the case of vertical fragmentation, so the fragments are never ‘completely’ disjoint. However, they are considered to be disjoint if non-key attributes from the original relation r appear in only one fragment.

(b) What is the goal of data allocation process when designing a distributed database?

Discuss different criteria for “optimal” data allocation.

- Reliability – more copies available in case of site crashes
- Increased performance for read-only queries
- Optimality may be defined in two ways: minimal cost and maximized performance.
 - Minimal cost – cost of storing and querying a fragment at a site
 - cost of updating a fragment at all sites
 - cost of data communication
 - Maximized performance – minimize query response time
 - maximize system throughput

Question 2

Is there any relationship between semijoin computation and derived horizontal fragmentation? Justify your answer.

Yes. Derived horizontal fragmentation is defined on the member relation of a link according to the selection operation defined on its owner, i.e. produces a fragment of the member relation which contains only tuples which are guaranteed to join with tuples in the owner relation (and satisfy some selection condition).

In a semijoin, say $r \bowtie s$, the joining column of s (owner) is sent over to r (member) to produce the *horizontal fragment* of r that is guaranteed to join with s .

Question 3

(a) Explain the main difference between 2 Phase Commit and 2 Phase Commit with Presumed Abort. Make your statement short and clear.

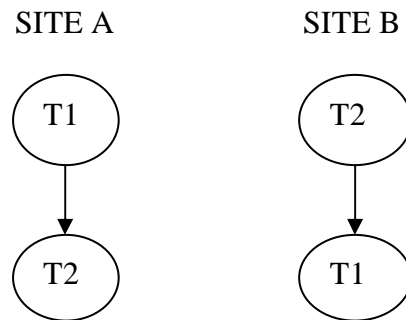
A transaction aborted by the coordinator can be immediately undone and removed from the transaction table. Any subsequent enquiry about the transaction from the subordinates may be answered with a default abort message. If a subordinate receives an abort message, it **need not**

return an ack message, since the coordinator is not waiting for it. **Abort records need not be force-written in the log**; subsequent to a crash, lack of a commit record leads the recovery module to abort the transaction. The basis for each of the above optimizations is that **lack of information (in terms of a message or a log record) implies the transaction needs to be aborted**. Hence the name 2PC with Presumed Abort.

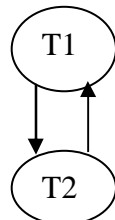
(b) Give a scenario of a transactions schedule demonstrating a distributed deadlock situation. Comment on how deadlock detection can be handled in a distributed system.

T1 @ Site A	T2 @ Site B
Read O1	
	Read O2
Write O2	
	Write O1

Local 'wait for' graphs:



Global 'wait for' graph:



There is a cycle in the global waits-for graph thus there is a deadlock.

There are three solutions to distributed deadlock detection:

- **Centralised** – one site is designated as the deadlock detector, all other sites send their local wait for graphs to the designated site which then constructs the global wait for graph
- **Hierarchical** – the sites are organised into a hierarchy of deadlock detectors, local wait for graphs are sent to a ‘parent’ node in the hierarchy which construct the global wait for graph (and in turn may send that graph to its parent node if it has one)
- **Time-out** – there is no ‘detection’ as such thus there is no overhead. Any transaction that has not completed within some specified time is assumed to be deadlocked and is subsequently aborted.

Question 4

Assume that there are 4 CPUs available for computation of a selection query on r (FIND ALL EMPLOYEES WITH LAST NAME STARTING WITH LETTER “L”) in a Shared Nothing architecture. Assume also, that the following table (with the standard meaning of the schema) provides a representative sample of the data for the selection calculation.

r(empID, Fname, Lname, Position, ProjectID)

23	John	Koo	S1	P160
244	Joe	Lee	S3	P160
334	John	Li	S1	P12
335	Mark	Jay	S2	P8
238	Steve	Loo	S15	P8
139	John	Len	S3	P20
236	Andy	Lek	S1	P24
233	John	Kay	S1	P24
137	Xue	Lam	S2	P4
34	Xum	Jin	S2	P32
583	John	Lek	S2	P28

- (a) Design TWO different ways to approach such a distribution of a selection operation.**

To answer this question you need to give two methods for partitioning the data. They do not have to be the BEST methods. The question simply asks for two reasonable methods and then a discussion of the results.

So, one possible way is:

Assign each letter of the alphabet a number $A = 1$, $B = 2$, and so on.

Hash on the first letter of the surname: e.g. $Lee - 12 \bmod 4 = 0$. Therefore all surnames starting with L go into bucket 0, all surnames starting with K go into bucket 3, all surnames starting with J go into bucket 2. Bucket 1 is empty.

Another possible method is to apply range partitioning, for example all tuples with surname values between A and G go to bucket 0, from H to K -> bucket 1 etc.

Yet another method is hashing by empID for example.

We could also hash on the empID, eg. $23 - 23 \bmod 4 = 3$. Many other functions are possible.

(b) Comment on the suitability of your invented methods.

The first one gives a skewed result for this dataset, however we know that for this particular dataset all tuples in bucket 0 have a surname starting with L. This means that the other buckets need not be considered when we compute the selection query given in the question. If more tuples are inserted into the table, the skewness may even out, all tuples with the surname starting with L will still be in bucket 0 however there may be tuples with other surnames in bucket 0 as well.

Range partitioning in this example (and for the chosen ranges) results in the data being distributed over two buckets with two buckets remaining empty. However, this means that we can identify the two buckets/processors which will be relevant to the selection query.

Hashing on the empID gives a less skewed result. However all processors are involved in the query execution as there is no way of identifying buckets which contain tuples with the surname starting with L.

Question 5

Assume that we need to build a new integrated system from two pre-existing databases; one managed and owned by Company A and another one by Company B, both operating in three Australian states only – QLD, ACT and NT.

Both companies store and maintain semantically compatible data although design in a different ways. The relational data structure for Company A and B is presented below. The data is a snapshot of tables for a given year (period of time)

Company A database – all sale values are in thousands (for example ‘23’ stands for ‘23000’);

Sale (ProductID, Sale-QLD , Sale-ACT, Sale-NT)

123	23	45	126
378	12	12	67
562	10	34	50
234	24	26	27
612	56	11	32
134	67	34	32

Company B database:

Sale (ProdID, State, Sale)

612	QLD	56000
234	QLD	31000
897	QLD	67000
612	NT	32000
234	NT	10000
897	NT	78000
612	ACT	34000
234	ACT	9000
897	ACT	78000

(a) Show a mapping of the data from Company B to the data format of Company A using SQL view definitions.

One possible way:

Create view saleQ(ProductID, Sale-QLD) as
(select ProdID, Sale/1000 from Sale@B where state = 'QLD')

Create view saleN(ProductID, Sale-NT) as
(select ProdID, Sale/1000 from Sale@B where state = 'NT')

Create view saleA(ProductID, Sale-ACT) as
(select ProdID, Sale/1000 from Sale@B where state = 'ACT')

Create view sale(ProductID, Sale-QLD , Sale-ACT, Sale-NT) as

Select *

From saleQ, saleA, saleN

Where saleQ.ProductID = saleN.ProductID and saleN.ProductID = saleA.ProductID

(b) When the physical data is combined (from Company A database and Company B database) will the schema Sale (ProductID, Sale-QLD, Sale-ACT, Sale-NT) be adequate for the integrated data store? If YES say why, if NO show further data modification required.

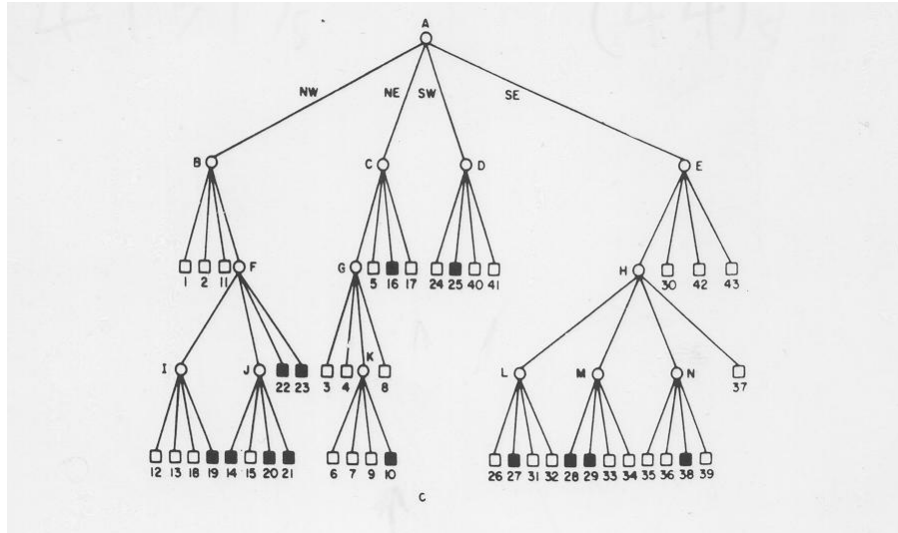
Generally NO. The key ProdID would be violated – multiple tuples with the same ProdID.

One way to deal with this problem is to extend the schema of Company A by adding a new attribute/column 'Company name' and making this attribute a part of the primary key of this table.

Another way to approach the problem is to sum up the sale values from Company A and B without modifying the structure of the schema or its key but changing the table name to reflect the data aggregation from the two companies.

Question 6

Consider the following quadtree:



(a) Find the location code for nodes: 10 and 32.

Node 10 location code:

Base 5: 2134

Base 10: 294

Node 32 location code:

Base 5: 4114

Base 10: 532

(b) Find the node with location code 294.

Converting 294 to base 5:

$$96 = 5 * 19 + 1$$

$$19 = 5 * 3 + 4$$

$$3 = 5 * 0 + 3$$

$$2 = 5 * 0 + 2$$

Hence, 294 (base 10) = 2134 (base 5). Which is node 10. (see part a)