

Tutorial 3: Distributed Transactions

INFS3200/7907
Advanced Database Systems

Overview

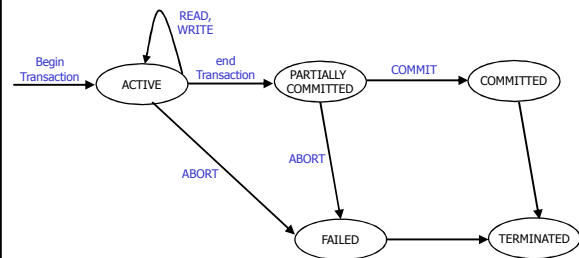
- Transaction
 - What?
 - A transaction is an executing program that forms a logical unit (read or write data terms) of database processing.

Overview - Transaction

- ACID
 - Atomicity
 - A transaction is either completed or abort entirely
 - Consistency
 - The maintenance of legal state in databases when the transaction begins and when it ends
 - Isolation
 - Each transaction is independent.
 - Durability
 - As a transaction is committed, the transaction will persist, and not be undone.
- Why?
 - The maintenance of data integrities.

Overview

How a transaction processes



Overview

- What is the difference between centralised and distributed transactions?
 - *Centralised* Transaction
 - Is a single process running on a **single processor**.
 - *Distributed* Transaction
 - Involves multiple **sub-transaction** processes on **tables, their replicas or fragments**, running on **multiple processors**.

Question 1 (Centralised Database Systems)

- When a race is run, a number of tables need to be updated
- Database Schemas
 - Results (EventID, CompID, Position)
 - Medals (EventID, Medal, CompID) where attribute Medal is "Gold", "Silver", "Bronze".
 - MedalTally (Country, Medal, Number)
 - Competitors (CompID, Country)
 - Medalists (CompID, NMedals) where NMedals is total number of medals for that competitor.

Question 1 Continues

Assumption

- When an event is completed, a file called *EventIdent* is generated at the venue giving *CompIdent* and *Pos*.

EventIdent file: 100MMBSFinal
(100M Men Butterfly Stroke Final)

CompIdent	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

EventIdent file: 100MBSF
(100M Men Butterfly Stroke Final)

EventIdent		Result		
CompIdent	Pos	EventID	CompID	Position
Hoogenband	4	100MBSF	Hoogenband	4
Iles	3	100MBSF	Iles	3
Schoeman	2	100MBSF	Schoeman	2
Thorpe	1	100MBSF	Thorpe	1

Medalists		Medals		
CompID	NMedals	EventID	Medal	CompID
Thorpe	2	100MBSF	Gold	Thorpe
Schoeman	2	100MBSF	Silver	Schoeman
Iles	2	100MBSF	Bronze	Iles

MedalTally		
Country	Medal	Number
Australia	Gold	10
S. Africa	Silver	3
Algeria	Bronze	5

Q1a

Write a program to perform the updates, using SQL *INSERT INTO* and *UPDATE* commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by *BEGIN TRANSACTION* and *COMMIT/END TRANSACTION* statements rather than using the SQL transaction management commands.

How are the event's results updated in a **centralised** database system?

- The system
 - Reads *EventIdent*, *CompIdent* and *Position* from file *EventIdent* and inserts the above parameters into table *Results*.
 - Declares interim variable *MedalAwarded* to specify which type of medal the athlete is awarded according to his/her position, i.e., position one, two and three are for medal Gold, Silver and Bronze respectively. And then inserts *EventIdent*, *CompIdent* and *MedalAwarded* into table *Medals*.
 - Regarding table *Medalists*, the athlete's medal number is incremented by one if his/her position in the event is in one of the top three.
 - The number of the awarded medals in table *MedalTally* achieved by the country the athlete belongs to is incremented by one as well.

Q1a

Write a program to perform the updates, using SQL *INSERT INTO* and *UPDATE* commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by *BEGIN TRANSACTION* and *COMMIT/END TRANSACTION* statements rather than using the SQL transaction management commands.

- The system reads *EventIdent*, *CompIdent* and *Position* from file *EventIdent*.

EventIdent file: 100MMBSFinal
(100M Men Butterfly Stroke Final)

CompIdent	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1a

Write a program to perform the updates, using SQL *INSERT INTO* and *UPDATE* commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by *BEGIN TRANSACTION* and *COMMIT/END TRANSACTION* statements rather than using the SQL transaction management commands.

- The system inserts the above parameters into table *Results*.

Results (*EventID*, *CompID*, *Position*)

INSERT INTO Results (EventID, CompID, Position)

(*EventIdent*, *CompIdent*, *Pos*)

EventIdent file: 100MMBSFinal

CompIdent	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1a

Write a program to perform the updates, using SQL *INSERT INTO* and *UPDATE* commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by *BEGIN TRANSACTION* and *COMMIT/END TRANSACTION* statements rather than using the SQL transaction management commands.

- Declares interim variable *MedalAwarded* to specify which type of medal the athlete is awarded according to his/her position, i.e., position one, two and three are for medal Gold, Silver and Bronze respectively.

Declare String MedalAwarded;

if Pos = 1

MedalAwarded = 'Gold';

else if Pos = 2

MedalAwarded = 'Silver';

else if Pos = 3

MedalAwarded = 'Bronze';

EventIdent file: 100MMBSFinal

CompIdent	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1a

Write a program to perform the updates, using SQL INSERT INTO and UPDATE commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by BEGIN TRANSACTION and COMMIT/END TRANSACTION statements rather than using the SQL transaction management commands.

2. Inserts Event, CompID and MedalAwarded into table Medals.

```
Medals (EventID, Medal, CompID)
INSERT Medals (EventID, Medal, CompID)
(EventID, MedalAwarded, CompID);
```

EventID file: 100MMBSFinal

CompID	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1a

Write a program to perform the updates, using SQL INSERT INTO and UPDATE commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by BEGIN TRANSACTION and COMMIT/END TRANSACTION statements rather than using the SQL transaction management commands.

3. Regarding to table Medalists, the athlete's medal number is incremented by one if his/her position in the event is in one of the top three.

```
Medalists (CompID, NMedals)
```

if Pos < 4

```
UPDATE Medalists
SET NMedals = NMedals + 1
Where CompID = CompID
```

EventID file: 100MMBSFinal

CompID	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1a

Write a program to perform the updates, using SQL INSERT INTO and UPDATE commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by BEGIN TRANSACTION and COMMIT/END TRANSACTION statements rather than using the SQL transaction management commands.

4. The number of the awarded medals in table MedalTally achieved by the country the athlete belongs to is incremented by one as well.

```
MedalTally (Country, Medal, Number)
Competitors (CompID, Country)
```

```
UPDATE MedalTally
SET Number = Number + 1
WHERE MedalTally.Country =
```

```
(SELECT Country
FROM Competitors
Where CompID = CompID) AND
Medal = MedalAwarded
```

EventID file: 100MMBSFinal

CompID	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1a Continues

Write a program to perform the updates, using SQL INSERT INTO and UPDATE commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by BEGIN TRANSACTION and COMMIT/END TRANSACTION statements rather than using the SQL transaction management commands.

```
BEGIN TRANSACTION
read inputfile EventID
while inputfile not empty
read inputfile CompID, Pos,
INSERT INTO Results(EventID, CompID, Position)
(EventID, CompID, Pos)
```

Q1a Continues

Write a program to perform the updates, using SQL INSERT INTO and UPDATE commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by BEGIN TRANSACTION and COMMIT/END TRANSACTION statements rather than using the SQL transaction management commands

```
if Pos < 4 then
  if Pos = 1 then
    MedalAwarded = "Gold"
  else if Pos = 2 then
    MedalAwarded = "Silver"
  else
    MedalAwarded = "Bronze"
INSERT INTO Medals (EventID, Medal, CompID)
(EventID, MedalAwarded, CompID)
UPDATE Medalists
SET NMedals = NMedals + 1
WHERE CompID = CompID
UPDATE MedalTally
SET Number = Number + 1
WHERE MedalTally.Country =
(SELECT Country FROM Competitors
WHERE CompID = CompID) AND
Medal = MedalAwarded
```

```
COMMIT
END TRANSACTION
```

Q1b

For each table, argue whether or not interference with another transaction is possible. In each case where it is possible, give an example of an interfering transaction.

Assumption

- All events are run under a centralised authority, e.g. Beijing Olympic Committee.
- Interference in this context means more than two transactions try to update same tables at the same time.
- with each other trying to update MedalTally at the same time.

Q1b

For each table, argue whether or not **interference** with another transaction is possible. In each case where it is possible, give an example of an interfering transaction.

- Database Schemas
 - Competitors (CompID, Country)
 - This table is relatively static because the roster is rarely changed during the Olympic Game.
 - Results (EventID, CompID, Position)
 - Both tables have EventID as part of primary keys, Normally, each event will be updated once and almost only once as the event completes. The likelihood of interference is quite low.
 - Medals (EventID, Medal, CompID)
 - Both tables have EventID as part of primary keys, Normally, each event will be updated once and almost only once as the event completes. The likelihood of interference is quite low.
 - Medalists (CompID, NMedals)
 - Similarly, each competitor will be updated once and almost only because they cannot compete two independent events at the same time.
 - MedalTally (Country, Medal, Number)
 - it is likely that two different events can interfere

Q1c

Annotate your program with **read-lock**, **write-lock** and **unlock** statements in some suitable notation (not SQL). The notation should make clear the **level of granularity of the locking**. In each case indicate whether the granularity of locking is more than strictly necessary. Justify your decision based on the characteristics of the application.

```

BEGIN TRANSACTION
WRITE LOCK TABLE Results
WRITE LOCK TABLE Medals
WRITE LOCK TABLE MedalTally
WRITE LOCK TABLE Medalists
READ LOCK TABLE Competitors
  perform body of transaction
UNLOCK TABLE Results
UNLOCK TABLE Medals
UNLOCK TABLE MedalTally
UNLOCK TABLE Medalists
UNLOCK TABLE Competitors
COMMIT
END TRANSACTION

```

Q1a

Write a program to perform the **updates**, using SQL **INSERT INTO** and **UPDATE** commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by **BEGIN TRANSACTION** and **COMMIT/END TRANSACTION** statements rather than using the SQL transaction management commands.

- The system inserts the above **parameters** into table **Results**.

Results (EventID, CompID, Position)

```

INSERT INTO Results (EventID, CompID, Position)
(EventID, CompID, Pos)

```

EventID file: 100MMBSFinal

CompID	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1a

Write a program to perform the **updates**, using SQL **INSERT INTO** and **UPDATE** commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by **BEGIN TRANSACTION** and **COMMIT/END TRANSACTION** statements rather than using the SQL transaction management commands.

- Inserts **Event**, **CompID** and **MedalAwarded** into table **Medals**.

Medals (EventID, Medal, CompID)

```

INSERT Medals (EventID, Medal, CompID)
(EventID, MedalAwarded, CompID);

```

EventID file: 100MMBSFinal

CompID	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1c Continues

Annotate your program with **read-lock**, **write-lock** and **unlock** statements in some suitable notation (not SQL). The notation should make clear the **level of granularity of the locking**. In each case indicate whether the granularity of locking is more than strictly necessary. Justify your decision based on the characteristics of the application.

- Table **Results** and **Medals**
 - It is not strictly necessary to lock Results or Medals at all
 - In this particular application only one transaction can **insert** tuples for any given event.
 - A conflict of write-item can not happen.

Q1a

Write a program to perform the **updates**, using SQL **INSERT INTO** and **UPDATE** commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by **BEGIN TRANSACTION** and **COMMIT/END TRANSACTION** statements rather than using the SQL transaction management commands.

- The number of the awarded medals in table **MedalTally** achieved by the country the athlete belongs to is incremented by one as well.

MedalTally (Country, Medal, Number)

Competitors (CompID, Country)

```

UPDATE MedalTally
SET Number = Number + 1
WHERE MedalTally.Country =
(SELECT Country
FROM Competitors
Where CompID = CompID) AND
Medal = MedalAwarded

```

EventID file: 100MMBSFinal

CompID	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1c Continues

Annotate your program with `read-lock`, `write-lock` and `unlock` statements in some suitable notation (not SQL). The notation should make clear the level of granularity of the locking. In each case indicate whether the granularity of locking is more than strictly necessary. Justify your decision based on the characteristics of the application.

Table Competitors

- The read lock is not strictly necessary because the table is quite static throughout the Games.

Q1a

Write a program to perform the updates, using SQL `INSERT INTO` and `UPDATE` commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by `BEGIN TRANSACTION` and `COMMIT/END TRANSACTION` statements rather than using the SQL transaction management commands.

- Regarding table `Medalists`, the athlete's medal number is incremented by one if his/her position in the event is in one of the top three.

Medalists (`CompID`, `NMedals`)

if `Pos < 4`

```
UPDATE Medalists
SET NMedals= NMedals+1
Where CompID=CompIdent
```

EventIdent file: 100MMBSFinal

CompIdent	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1a

Write a program to perform the updates, using SQL `INSERT INTO` and `UPDATE` commands, as shown in Section 8.6 of Elmasri and Navathe 4th Edition. Write the program as a single transaction bounded by `BEGIN TRANSACTION` and `COMMIT/END TRANSACTION` statements rather than using the SQL transaction management commands.

- The number of the awarded medals in table `MedalTally` achieved by the country the athlete belongs to is incremented by one as well.

MedalTally (`Country`, `Medal`, `Number`)

Competitors (`CompID`, `Country`)

```
UPDATE MedalTally
SET Number = Number + 1
WHERE MealTally.Country =
(SELECT Country
FROM Competitors
Where CompID = CompIdent) AND
Medal = MedalAwarded
```

EventIdent file: 100MMBSFinal

CompIdent	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

Q1c Continues

Annotate your program with `read-lock`, `write-lock` and `unlock` statements in some suitable notation (not SQL). The notation should make clear the level of granularity of the locking. In each case indicate whether the granularity of locking is more than strictly necessary. Justify your decision based on the characteristics of the application.

- Table `MedalTally` and `Medalist`
 - The writes locks are necessary, but coarse. We could lock specific tuples rather than the whole tables.
 - Further, the locks are outside the (if) loop, so persist for the entire process of updating the results of an event.

Q1c Continues

Annotate your program with `read-lock`, `write-lock` and `unlock` statements in some suitable notation (not SQL). The notation should make clear the level of granularity of the locking. In each case indicate whether the granularity of locking is more than strictly necessary. Justify your decision based on the characteristics of the application.

- The coarse granularity – the locks are imposed at table level instead of tuple level.
 - Pros
 - Overheads is much less because such locking is much simpler and performed much less often (outside the loop).
 - Cons
 - The competing transaction could have to wait longer.

Q1d

Show that your lock/unlock annotation constitute two-phase locking.

Two-Phase Locking (2PL) for Concurrency Control

- 2PL – A transaction is said to follow the 2PL if all locking operations (`read_lock`, `write_lock`) precede the first unlock operation in the transaction.
- Serializability guaranteed by 2PL
- Phases:
 - First: an expanding or growing phase
 - For example, adding `read_lock` or `write_lock`, or upgrading from `read_lock` to `write_lock`.
 - Second: shrinking phase
 - For example, `unlock` or downgrading from `write_lock` to `read_lock`.

Q1d Continues

Show that your lock/unlock annotation constitute *two-phase locking*.

- According to the definition, this is two-phase locking.

Q1e

Is there any circumstance in which the program would have to *abort* the transaction?

- Assuming that all the programs are working correctly, there is no semantic reason why an abort would ever be needed.
- A *system failure* or a *program error* resulting in an invalid database state would generate an abort as part of its error handling.

Question 2 (Distributed Database Systems)

- Assumption:
 - Each table is stored in a different site.
 - Site 0 Venue
 - Site 1 Results
 - Site 2 Medalists
 - Site 3 Medals
 - Site 4 MedalTally
 - Table Medals, Medalists and MedalTally are all replicated at several sites.
 - Your programming language supports a command `EXECUTE TransactionName AT siteID WITH a parameter list`, having the syntax of SQL `INSERT INTO` and `UPDATE` statements.

Q2a

Rewrite your program of Q1a into a series of transactions, one for each table. For replicated tables, shown two copies at distinct sites.

- Assumptions:
 - The interim variable `MedalAwarded` declared in Q1a *no longer exists* because this question is currently under distributed database systems.
 - Instead, there exists a table `MedalPos` (Position, Medal) whose rows are 1, "Gold"; 2, "Silver"; and 3, "Bronze".

MedalPos	
Position	Medal
1	Gold
2	Silver
3	Bronze

How a distributed transaction works



What happens at Site 0

EventID file: 100MBSF
(100M Men Butterfly Stroke Final)

A interim/ temporary table

EventIndent	
CompID	Pos
Hoogenband	4
Iles	3
Schoeman	2
Thorpe	1

TempResultsN		
EventID	CompID	Position
100MBSF	Hoogenband	4
100MBSF	Iles	3
100MBSF	Schoeman	2
100MBSF	Thorpe	1

Creates a temporary table `TempResultsN` (EventID, CompID, Position) to cache the parameters from file `EventIndent` generated by the venue, and then reads `EventID`, `CompID` and `Position` from file `EventIndent` and inserts to table `TempResultsN`.

What happens at Site 1

TempResultsN		
EventID	CompID	Position
100MBSF	Hoogenband	4
100MBSF	Iles	3
100MBSF	Schoeman	2
100MBSF	Thorpe	1

⇒

Result		
EventID	CompID	Position
100MBSF	Hoogenband	4
100MBSF	Iles	3
100MBSF	Schoeman	2
100MBSF	Thorpe	1

The tuples in table TempResultsN are retrieved as parameters to be inserted into table Result.

What happens at Site 2

TempResultsN		
EventID	CompID	Position
100MBSF	Hoogenband	4
100MBSF	Iles	3
100MBSF	Schoeman	2
100MBSF	Thorpe	1

⇒

Medalists	
CompID	NMedals
Thorpe	2
Schoeman	3
Iles	5

The athlete's medal number is incremented by one if his/her position in the event is in one of the top three.

What happens at Site 3

TempResultsN		
EventID	CompID	Position
100MBSF	Hoogenband	4
100MBSF	Iles	3
100MBSF	Schoeman	2
100MBSF	Thorpe	1

⇒

Medals		
EventID	Medal	CompID
100MBSF	Gold	Thorpe
100MBSF	Silver	Schoeman
100MBSF	Bronze	Iles

⇒

MedalPos	
Position	Medal
1	Gold
2	Silver
3	Bronze

Selects EventID, CompID and Medal from table TempResultsN and MedalPos, and inserts into table Medals and their replicas.

What happens at Site 4

TempResultsN		
EventID	CompID	Position
100MBSF	Hoogenband	4
100MBSF	Iles	3
100MBSF	Schoeman	2
100MBSF	Thorpe	1

⇒

MedalTally		
Country	Medal	Number
Australia	Gold	10
S. Africa	Silver	3
Algeria	Bronze	5

The number of the awarded medals in table MedalTally and its replicas, achieved by the countries the athletes belong to is incremented by one as well.

Q2a Continues

Rewrite your program of Q1a into a series of transactions, one for each table. For replicated tables, shown *two copies at distinct sites*.

- How are the event's results updated in a **distributed** database system?
- The system
 1. Creates a temporary table TempResultsN (EventID, CompID, Position) to cache the parameters from file *EventIndent* generated by the venue at *site 0*, and then reads *EventID*, *CompID* and *Position* from file *EventIndent* and inserts to table TempResultsN at *site 0*.
 2. Inserts the above parameters into table Results at *site 1*.
 3. Regarding table Medalists, the athlete's medal number is incremented by one if his/her position in the event is in one of the top three at *site 2*.
 4. Selects EventID, CompID and Medal from table TempResultsN and MedalPos, and inserts into table Medals at *site 3*.
 5. The number of the awarded medals in table MedalTally achieved by the country the athlete belongs to is incremented by one as well at *site 4*.

Q2a Continues

Rewrite your program of Q1a into a series of transactions, one for each table. For replicated tables, shown *two copies at distinct sites*.

1. Creates a temporary table TempResultsN (EventID, CompID, Position) to cache the parameters from file *EventIndent* generated by the venue at *site 0*.
2. Reads *EventID*, *CompID* and *Position* from file *EventIndent* and inserts to table TempResultsN at *site 0*.


```
BEGIN TRANSACTION UpdateEvent AT Site0
CREATE TABLE TempResultsN (EventID, CompID, Position)
//where this name is unique to this procedure
read inputfile EventIndent
while inputfile not empty
read inputfile CompIdent, Pos
INSERT INTO TempResultsN (EventID, CompID, Position)
(EventID, CompIdent, Pos)
...
COMMIT
END TRANSACTION
```

Q2a Continues

Rewrite your program of Q1a into a series of transactions, one for each table. For replicated tables, shown **two copies at distinct sites**.

- Inserts the above **parameters** into table **Results** at **site 1**.

```
...
EXECUTE UpdateResults AT Site1 (EventID,
CompID, Position)
(SELECT * FROM TempResultsN)
...
```

Q2a Continues

Rewrite your program of Q1a into a series of transactions, one for each table. For replicated tables, shown **two copies at distinct sites**.

- Regarding to table **Medalists**, the athlete's medal number is incremented by one if his/her position in the event is in one of the top three at **site 2**.

```
...
EXECUTE UpdateMedalists AT Site2 (CompID, Medal)
(SELECT T.CompID, M.Medal FROM TempResultsN T,
MedalPos M
WHERE T.Position < 4 AND T.Position = M.Position)
...
```

Q2a Continues

Rewrite your program of Q1a into a series of transactions, one for each table. For replicated tables, shown **two copies at distinct sites**.

- Selects **EventID**, **CompID** and **Medal** from table **TempResultsN** and **MedalPos**, and inserts into table **Medals** at **site 3**.

```
...
EXECUTE UpdateMedals AT Site3 (EventID, Medal,
CompID)
(SELECT T.EventID, M.Medal, T.CompID FROM
TempResultsN T, MedalPos M
WHERE T.Position < 4 AND T.Position = M.Position)
...
```

Q2a Continues

Rewrite your program of Q1a into a series of transactions, one for each table. For replicated tables, shown **two copies at distinct sites**.

- The number of the awarded medals in table **MedalTally** achieved by the country the athlete belongs to is incremented by one as well at **site 4**.

```
...
EXECUTE UpdateMedalTally AT Site4 (CompID, Medal)
(SELECT T.CompID, M.Medal FROM TempResultsN T,
MedalPos M
WHERE T.Position < 4 AND T.Position = M.Position)
...
```

Q2a Continues

Rewrite your program of Q1a into a series of transactions, one for each table. For replicated tables, shown **two copies at distinct sites**.

```
BEGIN TRANSACTION UpdateEvent AT Site0
CREATE TABLE TempResultsN (EventID, CompID, Position)
//where this name is unique to this procedure
read infile EventIdent
while infile not empty
    read infile CompID, Pos
    INSERT INTO TempResultsN (EventID, CompID, Position)
    (EventIdent, CompIdent, Pos)
EXECUTE UpdateResults AT Site1 (EventID, CompID, Position)
(SELECT * FROM TempResultsN)
EXECUTE UpdateMedalists AT Site2 (CompID, Medal)
(SELECT T.CompID, M.Medal FROM TempResultsN T, MedalPos M
WHERE T.Position < 4 AND T.Position = M.Position)
EXECUTE UpdateMedals AT Site3 (EventID, Medal, CompID)
(SELECT T.EventID, M.Medal, T.CompID FROM TempResultsN T, MedalPos M
WHERE T.Position < 4 AND T.Position = M.Position)
EXECUTE UpdateMedalTally AT Site4 (CompID, Medal)
(SELECT T.CompID, M.Medal FROM TempResultsN T, MedalPos M
WHERE T.Position < 4 AND T.Position = M.Position)
COMMIT
END TRANSACTION
```

BEGIN TRANSACTION UpdateEvent AT Site0

```
CREATE TABLE TempResultsN (EventID, CompID, Position)
//where this name is unique to this procedure
read infile EventIdent
while infile not empty
```

```
EXECUTE UpdateMedalists AT Site2 (CompID, Medal)
(SELECT T.CompID, M.Medal
FROM TempResultsN T, MedalPos M
WHERE T.Position < 4 AND
T.Position = M.Position)
```

```
EXECUTE UpdateMedals AT Site3 (EventID, Medal, CompID)
(SELECT T.EventID, M.Medal, T.CompID FROM TempResultsN T, MedalPos M
WHERE T.Position < 4 AND T.Position = M.Position)
COMMIT
END TRANSACTION
```

Q2a Continues

Rewrite your program of Q1a into a series of transactions, one for each table. For replicated tables, shown **two copies at distinct sites**.

```
BEGIN SUBTRANSACTION UpdateMedalists (Site2)
EXECUTE UpdateMedalistsCopy AT Site2.1
(Select * from Parameter)
EXECUTE UpdateMedalistsCopy AT Site2.2
(Select * from Parameter)
END SUBTRANSACTION
```

```
BEGIN SUBTRANSACTION UpdateMedalistsCopy (Site2.1, Site2.2)
```

```
UPDATE Medalists
SET NMedals = NMedals + 1
WHERE CompID IN
(SELECT CompID
FROM Parameter)
END SUBTRANSACTION
```

Q2b

Annotate your programs of Q2a with read-lock/write-lock/unlock statements conforming to primary copy two-phase locking, and read one, write all locking for replicated data.

- No lock in the main transaction

```
BEGIN SUBTRANSACTION UpdateMedalists (Site2)
WRITE LOCK Medalists AT Site2.1, Site2.2
EXECUTE UpdateMedalistsCopy AT Site2.1
(Select * from Parameter)
EXECUTE UpdateMedalistsCopy AT Site2.2
(Select * from Parameter)
UNLOCK Medalists AT Site2.1, Site2.2
END SUBTRANSACTION
```

Q2c

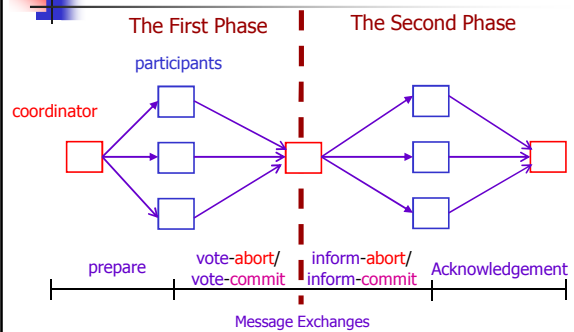
Show that it is not possible for two sub-transactions to interfere with each other, that is to say that the distributed transaction locks are equivalent to a centralised two-phase locking.

- None of the sub-transactions use data from any other sub-transactions.

Question 3

- Two-Phase Commit Protocol
 - What?
 - The Two-Phase Commit Protocol is a distributed algorithm which lets all nodes (participating sub-transactions) in a distributed system agree to commit a transaction.
 - How?
 - Global abort: if one participant vote to abort
 - Global commit: if all participants vote to commit.

Two-Phase Commit (2PC) Protocol



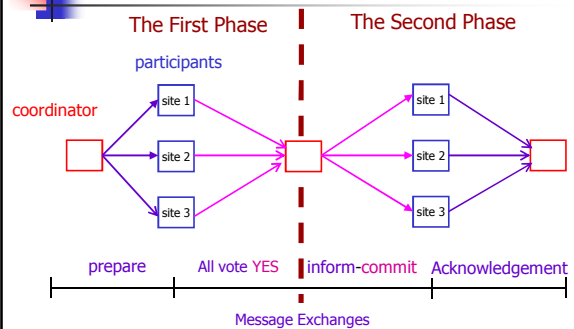
Tutorial 2 - Question 3a

Which site is the coordinator?

- Site 0, where the main transaction is executed.

Q3b

Show the exchange of messages among sub-transactions implementing two-phase commit in the case that all sub-transactions succeed.



Q3C

Suppose one of the MedallTally replica sites (site 3) fails during the transaction. Show the exchange of messages among sub-transactions resulting in the two-phase commit issuing an abort.

