Chapter 10

Exercise 10.1

Consider the database:

PRODUCTION (SerialNumber, PartType, Model, Quantity, Machine)
PICKUP (SerialNumber, Lot, Client, SalesPerson, Amount)
CLIENT (Name, City, Address)
SALESPERSON(Name, City, Address)

Assume four production centres located in Dublin, San Josè, Zurich and Taiwan and three sales points, located in San Josè, Zurich and Taiwan. Each production center is responsible for one type of part; the parts are *CPU*, *Keyboard*, *Screen* and *Cable*. Suppose also three sales points, located in San Josè, Zurich and Taiwan. Suppose that the sales are distributed by geographic location; thus, Zurich clients are served only by salespeople in Zurich (assume that the sales point in Zurich also serves Dublin). Assume that each geographic area has its own database (that is, databases are available in Dublin, San Josè, Zurich and Taiwan). Design the horizontal fragmentation of the tables **PRODUCTION**, **PICKUP**, **CLIENT** and **SALESPERSON**. Express the following queries on transparency levels of fragmentation, allocation and language:

1) Determine the available qunatity of the product 77y6878.
2) Determine the clients who have bought a lot from the retailer Wong, who has an office in Taiwan.
3) Determine the machines used for the production of the parts type *Keyboard* sold to the client Brown.
4) Modify the address of the retailer Brown, who is moving form ’27 Church St.’ in Dublin to ‘ 43 Park Hoi St.’ in Taiwan.
5) Calculate the sum of amounts of the orders received in San Josè, Zurich and Taiwan (note that the aggregate functions are also distributable).

Making any necessary assumptions about the use of the DBMS in Zurich, write a remote request, a remote transaction, a distributed transaction and a distributed request.

Sol:

Horizontal fragmentation of table PRODUCTION (4 tables)

$$PRODUCTION\_1=\sigma_{PartType=CPU}(PRODUCTION)$$
$$PRODUCTION\_2=\sigma_{PartType=Keyboard}(PRODUCTION)$$
$$PRODUCTION\_3=\sigma_{PartType=Screen}(PRODUCTION)$$
$$PRODUCTION\_4=\sigma_{PartType=Cable}(PRODUCTION)$$

Horizontal fragmentation of table PICKUP (4 tables)

$$PICKUP\_1=\Pi_{SerialNumber,\ Lot,\ Client,\ SalesPerson,\ Amount}\left(\sigma_{PartType=CPU}\left(PICKUP \underset{SN\leftarrow SerialNumber}{\underset{SerianNumber=SN}{}} (PRODUCTION)\right)\right)$$

$$PICKUP\_2=\Pi_{SerialNumber,\ Lot,\ Client,\ SalesPerson,\ Amount}\left(\sigma_{PartType=Keyboard}\left(PICKUP \underset{SN\leftarrow SerialNumber}{\underset{SerianNumber=SN}{}} (PRODUCTION)\right)\right)$$

$$PICKUP\_3=\Pi_{SerialNumber,\ Lot,\ Client,\ SalesPerson,\ Amount}\left(\sigma_{PartType=Screen}\left(PICKUP \underset{SN\leftarrow SerialNumber}{\underset{SerianNumber=SN}{}} (PRODUCTION)\right)\right)$$

$$PICKUP\_4=\Pi_{SerialNumber,\ Lot,\ Client,\ SalesPerson,\ Amount}\left(\sigma_{PartType=Cable}\left(PICKUP \underset{SN\leftarrow SerialNumber}{\underset{SerianNumber=SN}{}} (PRODUCTION)\right)\right)$$

Horizontal fragmentation of table CLIENT (3 tables)

$$CLIENT\_1=\sigma_{City=\text{"San Josè"}}(CLIENT)$$
$$CLIENT\_2=\sigma_{City=\text{"San Josè"}\ \ City=\text{"Dublin"}}(CLIENT)$$
$$CLIENT\_3=\sigma_{City=\text{"Taiwan"}}(CLIENT)$$

Horizontal fragmentation of table SALESPERSON (3 tables)

$$SALES\_1=\sigma_{City=\text{"San Josè"}}(SALESPERSON)$$
$$SALES\_2=\sigma_{City=\text{"Zurich"}}(SALESPERSON)$$
$$SALES\_3=\sigma_{City=\text{"Taiwan"}}(CLIENT)$$

The tables PRODUTION_1, PICKUP_1, CLIENT_1 and SALES_1 will be located in San Josè (Company.SanJose.com)

The tables PRODUTION_2, PICKUP_2, CLIENT_2 and SALES_2 will be located in Zurich (Company.Zurich.com)

The tables PRODUTION_3, PICKUP_3, CLIENT_3 and SALES_3 will be located in Taiwan (Company.Taiwan.com)

The tables PRODUTION_4 and PICKUP_4, will be located in Dublin (Company.Dublin.com)

1) Transparency of fragmentation:

```
Procedure Query1 ( :Quan)
   select Quantity into :Quan
   from PRODUCTION
   where SerialNumber=77y6828
end Procedure;
```

Transparency of allocation:

```
Procedure Query2 (:Quan)
   select Quantity into :Quan
   from PRODUCTION_1
   where SerialNumber=77y6828
if :empty then
   select Quantity into :Quan
   from PRODUCTION_2
   where SerialNumber=77y6828
if :empty then
   select Quantity into :Quan
   from PRODUCTION_3
   where SerialNumber=77y6828
if :empty then
   select Quantity into :Quan
   from PRODUCTION_4
   where SerialNumber=77y6828
end Procedure;
```

Transparency of language:

```
Procedure Query3 (:Quan)
   select Quantity into :Quan
   from PRODUCTION_1@company.sanjose.com
   where SerialNumber=77y6828
if :empty then
   select Quantity into :Quan
   from PRODUCTION_2@company.Zurich
   where SerialNumber=77y6828
if :empty then
   select Quantity into :Quan
   from PRODUCTION_3@Company.taiwan.com
   where SerialNumber=77y6828
if :empty then
   select Quantity into :Quan
   from PRODUCTION_4@company.dublin.com
   where SerialNumber=77y6828
end Procedure;
```

2) Transparency of fragmentation:

```
Procedure Query1 (:cl)
  select Client int :cl
  from PICKUP
  where SalesPerson=”Wong”
end Procedure;
```

Transparency of allocation:

```
Procedure Query2 (:cl)
  select Client into :cl
  from PICKUP_1
  where SalesPerson=”Wong”
if :empty then
  select Client init :cl
  from PICKUP_2
  where SalesPerson=”Wong”
if :empty then
  select Client into :cl
  from PICKUP_3
  where SalesPerson=”Wong”
if :empty then
  select Client into :cl
  from PICKUP_4
  where SalesPerson=”Wong”
end Procedure;
```

Transparency of allocation:

```
Procedure Query3 (:cl)
  select Client into :cl
  from PICKUP_1@company.sanjose.com
  where SalesPerson=”Wong”
if :empty then
  select Client into :cl
  from PICKUP_2@company.zurich.com
  where SalesPerson=”Wong”
if :empty then
  select Client into :cl
  from PICKUP_3@company.taiwan.com
  where SalesPerson=”Wong”
if :empty then
  select Client into :cl
  from PICKUP_4@company.dublin.com
  where SalesPerson=”Wong”
end Procedure;
```

3) Transparency of fragmentation:

```
Procedure Query1 (:mach)
  select machine into :mach
  from PRODUCTION join PICKUP on
        PRODUCTION.SerialNumber=PICKUP.SerialNumber
  where Client=”Brown” and PartType=”Keyboard”
end Procedure;
```

Transparency of allocation:

```
Procedure Query2 (:mach)
  select machine into :mach
  from PRODUCTION_2 join PICKUP_2 on
        PRODUCTION_2.SerialNumber=PICKUP_2.SerialNumber
  where Client=”Brown”
end Procedure;
```

Transparency of language:

```
Procedure Query3 (:mach)
  select machine into :mach
  from PRODUCTION_2@company.zurich.com
            join PICKUP_2@company.zurich.com on
        PRODUCTION_2.SerialNumber=PICKUP_2.SerialNumber
  where Client=”Brown”
end Procedure;
```

4) Transparency of fragmentation:

```
Procedure Update1
  update CLIENT
  set Address=”43 Park Hoi St.”, City=”Taiwan”
  where name=”Brown”
end Procedure;
```

Transparency of allocation:

```
Procedure Update2
  delete from CLIENT_2
  where name=”Brown”;
  insert into CLIENT_3 (Name, Address, City) values
       (“Brown”, “43 Park Hoi St.”, “Taiwan”)
end Procedure;
```

Transparency of language:

```
Procedure Update3
  delete from CLIENT_2@company.zurich.com
  where name=”Brown”;
  insert into CLIENT_3@company.taiwan.com
       (Name, Address, City) values
            (“Brown”, “43 Park Hoi St.”, “Taiwan”)
end Procedure;
```

5) Transparency of fragmentation:

```
Procedure Query1
  select City, sum(Amount)
  from PICKUP join SALESPERSON on SalesPerson=Name
  group by City
end Procedure;
```

Transparency of allocation:

```
Procedure Query2
  create view PICKUP as
       PICKUP_1 union PICKUP_2 union PICKUP_3 union PICKUP_4;
  select City, sum (Amount)
  from CLIENT_1 join PICKUP on SalesPerson=Name
       union
  select City, sum (Amount)
  from CLIENT_2 join PICKUP on SalesPerson=Name
       union
  select City, sum (Amount)
  from CLIENT_3 join PICKUP on SalesPerson=Name
       union
  select City, sum (Amount)
  from CLIENT_4 join PICKUP on SalesPerson=Name
end Procedure;
```

Transparency of language:

```
Procedure Query3
  create view PICKUP as
       PICKUP_1@company.sanjose.com union
       PICKUP_2@company.zurich.com union
       PICKUP_3@company.taiwan.com union
       PICKUP_4@company.dublin.com;
  select City, sum (Amount)
  from CLIENT_1@company.sanjose.com join PICKUP on
            SalesPerson=Name
       union


  select City, sum (Amount)
  from CLIENT_2@company.zurich.com join PICKUP on
```

```
            SalesPerson=Name
        union
    select City, sum (Amount)
    from CLIENT_3@company.taiwan.com join PICKUP on
            SalesPerson=Name
        union
    select City, sum (Amount)
    from CLIENT_4@company.dublin.com join PICKUP on
            SalesPerson=Name
end Procedure;
```

Remote Request:

```
select *
from PRODUCTION_2
where quantity >10
```

Remote Transaction:

```
update CLIENT_2
set Address=”45 Rome st.”
where Name=”Thomson”
```

Distributed Transaction:

```
delete from CLIENT_2
where Name=”Thomson”;
insert into CLIENT_3 (Name, Address, City)
    values (“Thomson”, “43 Park Hoi St.”, “Taiwan” )
```

Distributed Request:

```
select Name
from PICKUP_2 join CLIENT_3 on Client=Name
where SerialNumber=”445679”.
```

Exercise 10.2

Assign the timestamps to the events described in Figure 10.22 with the Lamport method, and indicate which events are pseudo-simultaneous (events at different nodes that cannot be ordered).

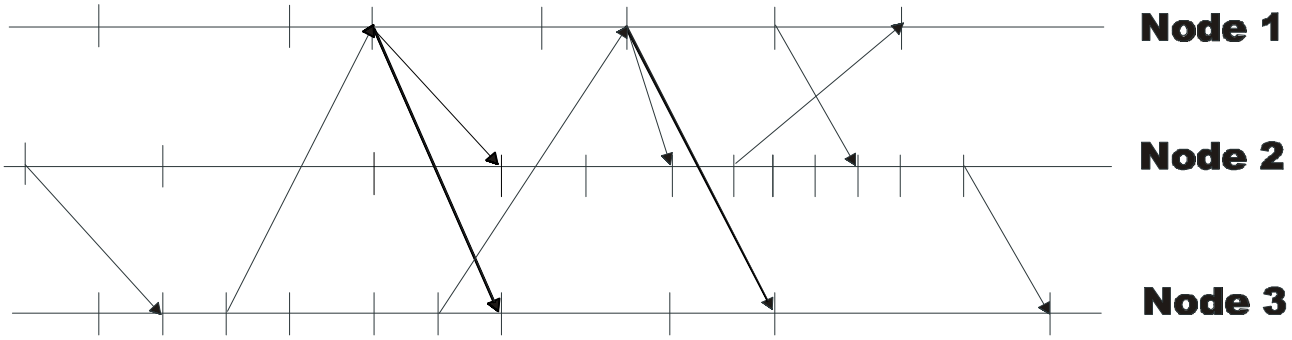
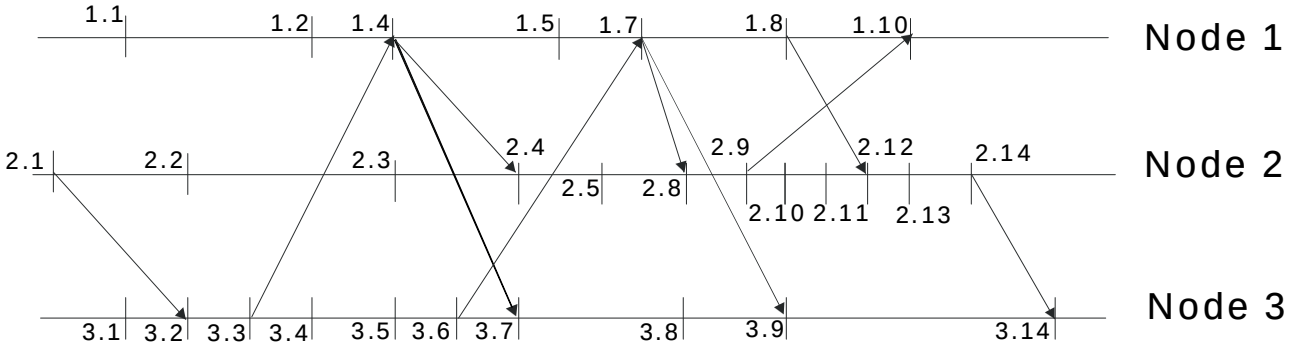


Figure 10.22

Sol:

The timestamps are indicated in the following figure:



The events which are pseudo-simultaneous are :

1.2 , 2.1 and 3.1;
1.1 and 2.2
1.2 and 2.3
1.5 and 2.5
2.3 and 3.4
2.8 and 3.8

Exercise 10.3

Given the wait conditions shown in Figure 10.23, look for deadlocks with the distributed deadlock detection algorithm; assume two different hypotheses of wait condition for node 4.

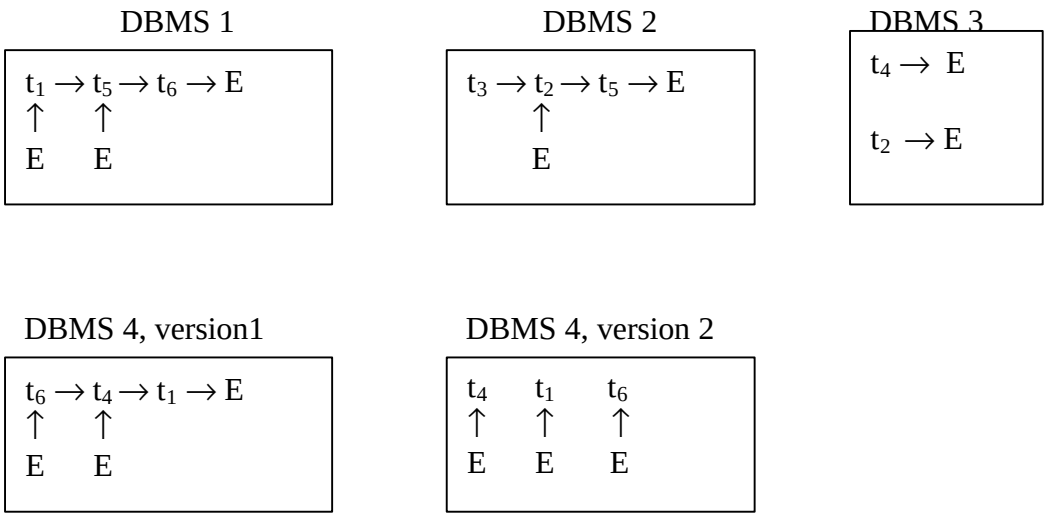


Figure 10.23

Sol:

First version

In this situation, the wait conditions for the system are:

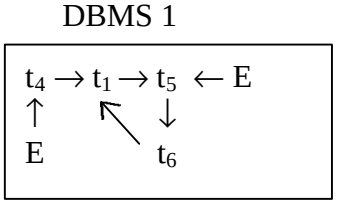
1) $E \rightarrow t_1 \rightarrow t_6 \rightarrow E$
   $E \rightarrow t_5 \rightarrow t_6 \rightarrow E$

2) $E \rightarrow t_2 \rightarrow t_5 \rightarrow E$

3) $t_4 \rightarrow E$
   $t_2 \rightarrow E$

4) $E \rightarrow t_6 \rightarrow t_1 \rightarrow E$
   $E \rightarrow t_4 \rightarrow t_1 \rightarrow E$

The DBMS 4 sends its wait conditions to DBMS 1 which discovers the deadlock:



One of the transactions t_1, t_5, t_6 will be rolled up

Second Version

In this situation the wait conditions are:

1) $E \rightarrow t_1 \rightarrow t_6 \rightarrow E$
   $E \rightarrow t_5 \rightarrow t_6 \rightarrow E$

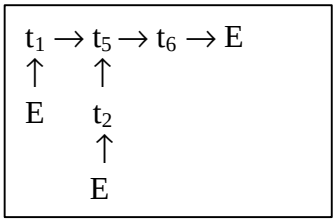
2) $E \rightarrow t_2 \rightarrow t_5 \rightarrow E$

3) $t_4 \rightarrow E$
   $t_2 \rightarrow E$

4) $E \rightarrow t_4$
   $E \rightarrow t_6$
   $t_1 \rightarrow E$

The DBMS 2 sends its wait conditions to DBMS 1:

DBMS 1

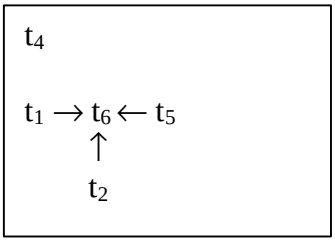


The new wait conditions are:

1) $E \rightarrow t_1 \rightarrow t_6 \rightarrow E$
   $E \rightarrow t_5 \rightarrow t_6 \rightarrow E$
   $E \rightarrow t_2 \rightarrow t_6 \rightarrow E$

The DBMS 1 and DBMS 3 send their wait condition to DBMS 4:

DBMS 1



There is no deadlock.

Exercise 10.4

Describe how the warm restart protocol (of Chapter 9) is modified, by taking into account the fact that some sub-transactions can be in a ready state.

Sol:

Warm Restart Protocol in distributed system

1) The log is traced back until the first checkpoint record is found. 3 sets are created: UNDO, REDO and READY. The sets REDO and READY are empty, while UNDO contains all the transaction indicated in the checkpoint.

2) The log is traced forward, if a Commit record is found, the corresponding transaction is put in the REDO set and removed from UNDO or READY;
   if a READY record is found, the corresponding transaction is put in the READY set and removed from UNDO;
   if an ABORT record is found and the corresponding transaction is in the READY set, it is put in UNDO set and removed from the READY set.

3) If the READY set is not empty, the system must ask to the Transaction Manager for the fate of the transactions in the set. The Transaction Manager indicates which transactions are aborted and which are committed; these transaction are transferred in the UNDO and REDO sets, and the respective records are written in the log.

4) The protocol continues as in non distributed systems.

Exercise 10.5

Apply the warm restart protocol after a failure of a node, assuming a two-phase commit protocol, having the following input (where $R(T_i)$ indicate the presence of a **ready** record):

$B(T_1)$, $B(T_2)$, $B(T_3)$, $I(T_1,O_1,A_1)$, $D(T_2,O_2,B_2)$, $B(T_4)$, $R(T_1)$, $U(T_4,O_3,B_3,A_3)$, $C(T_1)$, $CK(T_2,T_3,T_4)$, $B(T_5)$, $B(T_6)$, $U(T_5,O_5,B_5,A_5)$, $R(T_5)$, $B(T_7)$, $U(T_7,O_6,B_6,A_6)$, $B(T_8)$, $U(T_6,O_1,B_7,A_7)$, $A(T_7)$, $R(T_6)$, failure.

Sol:

1) The log is traced back until $CK(T_2,T_3,T_4)$

   UNDO={ T_2, T_3, T_4 } REDO={} READY={}

2) The log is traced forward, updating the sets:

$B(T_5)$	UNDO={ T_2, T_3, T_4, T_5 }	REDO={}	READY={}
$B(T_6)$	UNDO={ T_2, T_3, T_4, T_5, T_6 }	REDO={}	READY={}
$R(T_5)$	UNDO={ T_2, T_3, T_4, T_6 }	REDO={}	READY={T_5}
$B(T_7)$	UNDO={ T_2, T_3, T_4, T_6, T_7 }	REDO={}	READY={T_5}
$B(T_8)$	UNDO={ T_2, T_3, T_4, T_6, T_7, T_8 }	REDO={}	READY={T_5}
$A(T_7)$	UNDO={ T_2, T_3, T_4, T_6, T_7, T_8 }	REDO={}	READY={T_5}
$R(T_6)$	UNDO={ T_2, T_3, T_4, T_7, T_8 }	REDO={}	READY={T_5, T_6}

3) Let us suppose that the TM indicates for T_5 and T_6 a global commit:

   UNDO={ T_2, T_3, T_4, T_7, T_8 } REDO={T_5, T_6}

   The records $C(T_5)$, and $C(T_6)$ are written to the log.

4) The log is traced back until $D(T_2,O_2,B_2)$, executing the following undo operation:

   $O_6=B_6$
   $O_3=B_3$
   Insert $O_2=B_2$

5) Finally the log is traced forward executing the following redo-operation:

   $O_5=A_5$
   $O_1=A_7$

Exercise 10.6

Describe the warm restart protocol after the failure of a node, assuming a three-phase commit protocol, having the following input (where $PC(T_i)$ indicates the presence of a **pre-commit** record).

$B(T_1)$, $B(T_2)$, $B(T_3)$, $I(T_1,O_1,A_1)$, $D(T_2,O_2,B_2)$, $B(T_4)$, $R(T_1)$, $U(T_4,O_3,B_3,A_3)$, $PC(T_1)$, $C(T_1)$, $CK(T_2,T_3,T_4)$, $B(T_5)$, $B(T_6)$, $U(T_5,O_5,B_5,A_5)$, $R(T_5)$, $B(T_7)$, $U(T_7,O_6,B_6,A_6)$, $U(T_6,O_3,B_7,A_7)$, $B(T_8)$, $PC(T_5)$, $A(T_7)$, $R(T_6)$, failure.

Sol:

1) As in the other warm restart protocols, first of all the log is traced back until the first checkpoint record $CK(T_2,T_3,T_4)$. The 4 sets are created: REDO, UNDO, READY and PRECOMMIT. The set UNDO contains all the transactions indicated in the checkpoint record, while the other sets are ready.

   UNDO={ T_2, T_3, T_4 } REDO={} READY={} PRECOMMIT={};

2) the log is traced forward. If a commit record is found, the transaction is collocated in the REDO set; if an abort record is found, the transaction is collocated in the UNDO set; if a ready record is found, the transaction is collocated in the READY set; finally, if a pre-commit record is found, the transaction is collocated in the PRECOMMIT set.
   In this case:

   UNDO={ T_2, T_3, T_4, T_7, T_8 } REDO={} READY={T_6} PRECOMMIT={T_5};

3) The system must consult the coordinator about the fate of the transactions in READY and PRECOMMIT sets. For each transaction in PRECOMMIT set the coordinator can answer with commit, abort or still precommit: in the first two cases the transaction is collocated in the respective set, elsewhere the system must resume the 3PC protocol.
   For each transaction in the READY set the coordinator can answer again with commit, abort or precommit. In case of precommit the transaction is collocated in the PRECOMMIT set and the 3PC protocol is resumed.

4) When the fate of each transaction is known, the warm restart protocol continues with the undo and redo operations.

If the coordinator does not answer in a specified time the system starts the Coordinator Failure Protocol.

Exercise 10.8

On the same database schema of Exercise 10.1, describe an execution schema for the following queries that maximize the inter-query parallelism:

- extract the sum of the production quantities, grouped according to type and model of the parts;
- extract the average value of parts sold by salespeople, grouped according to type and model of the parts.

Sol:

In Exercise 10.1 we suggested an horizontal fragmentation of tables PRODUCTION and PICKUP based on the different values of the attribute PartType.
According to this fragmentation, the first query can be executed with 4 different queries, each applied to one of the tables PRODUCTION_1, PRODUCTION_2, PRODUCTION_3, PRODUCTION_4

```
select sum(quantity), Model, PartType
from PRODUCTION_1
group by Model, PartType

select sum(quantity), Model, PartType
from PRODUCTION_2
group by Model, PartType

select sum(quantity), Model, PartType
from PRODUCTION_3
group by Model, PartType

select sum(quantity), Model, PartType
from PRODUCTION_4
group by Model, PartType
```

Note that each table contains only one value in PartType

On the same schema the second query may be

```
select avg(amount), Model, PartType
from PICKUP_1
group by Model, PartType

select avg(amount), Model, PartType
from PICKUP_2
group by Model, PartType

select avg(amount), Model, PartType
from PICKUP_3
group by Model, PartType
```

```
select avg(amount), Model, PartType
from PICKUP_4
group by Model, PartType
```

In the situation described in Exercise 10.1, each query will be executed by a different DBMS. To maximize the inter-query parallelism, each query could be even more fragmented according to the different values of attribute Model.
In this way each of the 4 queries becomes a set of small queries without group by operations. This could be very useful if the DBMS has a multi-processor architecture.

Exercise 10.9

Describe an example of replicated database behaviour that produces a data inconsistency.

Sol:

With reference to the database schema described in Exercise 10.1, we suppose that each fragment of table PRODUCTION is allocated to all DBMSs. Each DBMS uses only one fragment and transmits all the changes on this fragment to the other DBMSs. In this way all the DBMSs have an entire copy of the Database.
In case of failure of one of the DBMSs the Database is still completely accessible by the other systems: if all the queries are correctly re-directed, the failure is transparent to a Client of the Database.
However if a network failure occurs, this may generate a network partition, for example with two independent sub-networks.
In this situation a data inconsistency is possible: suppose that table PRODUCTION contains a tuple

SerialNumber	PartType	Model	Quantity	Machine
12345	CPU	Pentium II	1000	A

if two transactions would both subtract 800 from the quantity of this part, one of the transactions (the second in temporal order) should fail; but if the two transactions are present in two DBMSs which are not connected, they will not fail and produce an inconsistency in the complete Database.

Exercise 10.10

Describe an example of symmetrical replication that produces a data inconsistency.

Sol:

In the context of symmetrical replication, each DBMS has a copy of the entire Database and accesses all of it; then the copies are “reconciled”.
This situation may produce the same effect of a single Database without concurrency control.
The two transactions described in Exercise 10.9 may produce an inconsistency also without any network failure if the changes are not communicated immediately to the other copies.