Database Administration and Tuning
Mini-Project 2 - Report

Henrique Rocha - 68621
Ludijor Barros - 68626
Fábio Martins - 71073

May 28, 2015

1 Transaction Isolation Levels

Consider a relational database for information related to cheese products, with the following tables:

CHEESE(cheeseID, type, producer, calories, proteins)
REGION(regionID, name, country)
PRODUCTION (productionID, cheeseID, season, amount)
PROVENANCE (productionID, regionID)

Assume that the following three stored procedures can run concurrently in a given application that is supported by the relational database:

1. **insert_cheese**: creates new records in the CHEESE table, for new cheeses that are produced. This procedure uses only the CHEESE table.

2. **update_production**: inserts new records, or modifies existing PRODUCTION and PROVENANCE records. This procedure writes to the PRODUCTION and PROVENANCE tables, updates the tuples in the PROVENANCE table, and may be reading from the REGION and CHEESE tables.

3. **delete_region**: deletes a given region from the REGION table.

1.1

Give a scenario that leads to a possible dirty read in the concurrent execution of operations from this group of stored procedures, or explain why a dirty read cannot happen in this group of stored procedures.

Consider two transactions both using **update_production** stored procedure over the same PRODUCTION tuples.

T1	T2
BEGIN	
R(REGION)	
R(CHEESE)	
R(PRODUCTION_X)	
W(PRODUCTION_X)	
	BEGIN
	R(REGION)
	R(CHEESE)
	R(PRODUCTION_X)
	W(PRODUCTION_X)
	R(PROVENANCE)
	W(PROVENANCE)
	COMMIT
R(PROVENANCE)	
ABORT	

Figure 1: An example of dirty read

T2 commits and T1 aborts after, rolling back its changes, the PRODUCTION tuples that T2 read from T1 are dirty reads because they reflect changes of an aborted transaction (T1).

1.2

Give a scenario that leads to a possible non-repeatable read in the concurrent execution of operations from this group of stored procedures, or explain why a non-repeatable read cannot happen in this group of stored procedures.

Consider that transaction one executes two **update_production** stored procedures over the same REGION tuple, the second transaction deletes this REGION tuple.

T1	T2
BEGIN	
R(REGION_X)	
R(CHEESE)	
R(PRODUCTION)	
W(PRODUCTION)	
R(PROVENANCE)	
W(PROVENANCE)	
	BEGIN
	DELETE(REGION_X)
	COMMIT
R(REGION_X)	
R(CHEESE)	
R(PRODUCTION)	
W(PRODUCTION)	
R(PROVENANCE)	
W(PROVENANCE)	
COMMIT	

Figure 2: An example a non-repeatable read

The second time the REGION tuple is read, it doesn't exists, leading to a non-repeatable read.

1.3

Give an example of a possible overwriting of uncommitted data in the concurrent execution of operations from this group of stored procedures, or explain why a phantom read cannot happen in this group of stored procedures.

Consider two transactions both using **update_production** stored procedure over the same PRODUCTION tuples.

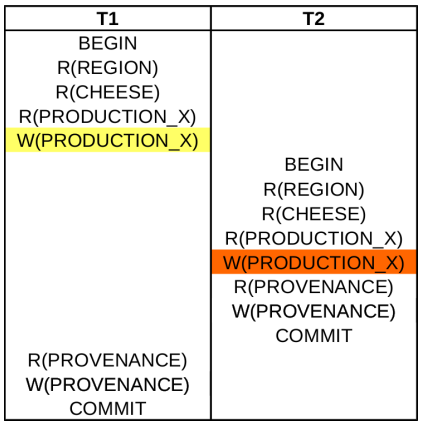


Figure 3: An example overwriting of uncommitted data

T2 commits and T1 PRODUCTION tuple value is lost.

1.4

Indicate what transaction isolation level would you use for executing each of the three procedures above, and why? For each procedure you should use the least restricted transaction isolation level that ensures correctness.

1. **insert_cheese**: read uncommited, because one either serializable or not, if two transactions insert the same cheese, one of them must fail with a violation of the primary key constraint.

2. **update_production**: repeatable read, if a read over the same PRODUCTION or PROVENANCE tuples occurs, the second transaction should wait for the first to commit.

3. **delete_region**: read commited, because if an update_production is happening the delete should wait for it to commit (update_production has read a region)

2 Concurrency Control

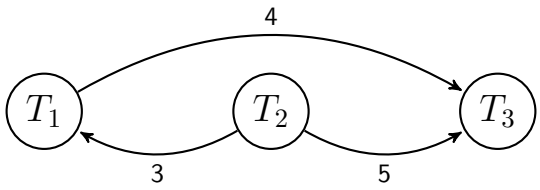
2.1

Consider the following schedule for three concurrent transactions:

	T1	T2	T3
1	write(A)		
2			write(B)
3		write(A)	
4	write(B)		
5		write(B)	

2.1.1

Is the schedule allowed in Strict 2-Phase Locking? Justify.



Yes, strict 2-Phase Locking allows only schedules whose precedence graph is acyclic, in this case the graph is acyclic.

2.1.2

Is the schedule allowed by the timestamp-based protocol? Justify.
TS(T_1)=1
TS(T_2)=3
TS(T_3)=2

	A		B	
t	Rts(A)	Wts(A)	Rts(B)	Wts(B)
1	0	1	0	0
2	0	1	0	2
3	0	3	0	2
4				
5				

No, at $t = 4$ transaction T_1 issues **write**(B), $[TS(T_1) = 1] < [Wts(B) = 2]$, which means that T_1 is attempting to write an obsolete value of B, hence this write operation is rejected, T_1 is rolled back.

2.2

Consider the following schedule for two concurrent transactions:

	T1	T2
1	lock-S(A)	
2	read(A)	
3	unlock(A)	
4		lock-X(B)
5		write(B)
6		unlock(B)
7	lock-S(B)	
8	read(B)	
9	unlock(B)	

2.2.1

Is the schedule allowed in Strict 2-Phase Locking? Justify.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

2.2.2

Is the schedule allowed by the timestamp-based protocol? Justify.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

3 Recovery System

Consider the following simplified representation for the log records that correspond to a given execution, and suppose the ARIES algorithm is followed by the recovery system:

LSN	Type	Transaction	Page
10	Update	T1	P1
20	Update		P2
30	Begin_checkpoint	-	-
40	End_checkpoint	-	-
50	Commit	T1	
60	Update	T3	P3
70	Commit	T2	-
80	Update	T3	P2
90	Update	T3	P5
	CRASH!!!		

Show the contents of the log, the dirty page table, and the active transactions table after (a) the analysis phase, (b) the redo phase, and (c) the undo phase.

3.1 Analysis Phase

LSN	Type	Transaction	Page
50	Commit	T1	
60	Update	T3	P3
70	Commit	T2	-
80	Update	T3	P2
90	Update	T3	P5

Table 1: Log

Transaction ID	Last LSN
T1	50
T2	70
T3	90

Table 2: Active Transactions Table

Page ID	Recovery LSN
P3	60
P5	90

Table 3: Dirty Pages Table

3.2 Redo Phase

LSN	Type	Transaction	Page
50	Commit	T1	
60	Update	T3	P3

Table 4: Log

Transaction ID	Last LSN
T3	60

Table 5: Active Transactions Table

Page ID	Recovery LSN
P3	60

Table 6: Dirty Pages Table

3.3 Undo Phase

LSN	Type	Transaction	Page
50	Commit	T1	
60	Update	T3	P3

Table 7: Log

Transaction ID	Last LSN
T3	60

Table 8: Active Transactions Table

Page ID	Recovery LSN
P3	60

Table 9: Dirty Pages Table

4 Schema and Index Tuning

Consider the relation CHEESE from Question 1. For each cheese, the relation lists an id, a name, a number of calories, and a number of proteins:

CHEESE(cheeseID, Type, Producer, Calories, Proteins)

The following two queries are extremely important and frequent:

Q1) Finding the average number of calories by cheese type, for all cheeses.

Q2) Listing the cheese ids of the cheeses with most proteins.

4.1

Write the two queries in SQL.

4.1.1

```
SELECT Type, AVG(Calories)
FROM CHEESE
GROUP BY Type;
```

4.1.2

```
SELECT cheeseID, Proteins
FROM CHEESE
ORDER BY Proteins
```

4.2

Suppose you have a heap file to store the records. What indexes would you create to speed up the two queries above? Justify.

4.2.1

For the first query we would create a non-clustered index over "Type" with Calories as additional attribute, because retrieving the grouping columns directly from an index is the most efficient way to process a "GROUP BY" (Loose Index Scan), the reason for adding Calories as additional attribute is to prevent finding the CHEESE tuple itself, obtaining the result directly from the index.

4.2.2

For the second query we would create a non-clustered index over "Proteins" because if the index is B+ tree then we don't need to perform the sort operation.

4.3

Suppose your customers complain that performance is not satisfactory. If you consider a schema redesign as a tuning option, explain how you would try to obtain better performance by describing the schema for the relation(s) that you would use and your choice of file organizations and indexes on these relations.

4.3.1

We would create a special table, also known as materialized view, with pre-computed sum of Calories and eventual triggers to update this new special table.

4.3.2

We would create another materialized view, without the extra Type, Producer, Calories attributes. The second query would process less amount of data.

5 Query Tuning

For each of the following queries, identify one possible reason why an optimizer might not find a good execution plan. Rewrite the query so that a good plan is likely to be found. Any available indexes, or known constraints, are listed before each query.

5.1

An index is available on the calories attribute:

```
SELECT type
FROM CHEESE
WHERE calories * 100 < 800
```

The optimizer might not find a good execution plan because the index over calories times 100 might not exist, a good solution is to divide 800 by 100.

```
SELECT type
FROM CHEESE
WHERE calories < 8
```

It is important that the index is a b+ tree so the search for cheeses with calories lower then 8 is sequential after the first tree traversal.

5.2

An index is available on the calories attribute:

```
SELECT type
FROM CHEESE
WHERE calories<90 AND calories>40
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX[1]

5.3

A B+ Tree index is available on the calories attribute:

```
SELECT type
FROM CHEESE
WHERE calories=90 OR calories=40
```

The optimizer will not consider the index on age as it is misled by the OR predicate. To make it consider the index we can use UNION instead or an OR.

```
SELECT type    FROM CHEESE    WHERE calories=90
UNION
SELECT type    FROM CHEESE    WHERE calories=40
```

5.4

No index is available:

```
SELECT DISTINCT CheeseId, type
FROM CHEESE
```

Tuples containing the primary key will be by default unique, so the distinct operation is needless.

```
SELECT CheeseId, type
FROM CHEESE
```

5.5

No index is available:

```
SELECT AVG (proteins)
FROM CHEESE
GROUP BY producer
HAVING type = "Alverca"
```

[1] BETWEEN

By grouping cheeses by producer and filtering by type only after, most of groups with type "Alverca" will be discarded, to improve the performance of this query, the filter condition by type should be first in order to the group operation process less tuples, in this case the filter condition is applied to tuples instead of groups so the HAVING condition is swapped by WHERE condition.

```
SELECT AVG (proteins)
FROM CHEESE
WHERE type = "Alverca"
GROUP BY producer
```

6 Database Tuning

Consider the following two queries:

```
Q1:    SELECT type, producer
       FROM CHEESE NATURAL JOIN PRODUCTION NATURAL JOIN PROVENANCE NATURAL JOIN
           REGION
       WHERE country = 'Portugal' AND amount > 10,000 AND season = 'Winter'
```

```
Q2:    SELECT type, COUNT(*)
       FROM CHEESE
       GROUP BY type;
```

6.1

Which of these queries corresponds to a typical access pattern of OLTP applications? And which one corresponds to a typical access pattern of OLAP queries? Justify your answer.

OLAP applications typically involves agreggations and complex queries which is the case of Query 1.

OLTP applications typically involves simple queries that returns relatively few records, which is the case of Query 2.

6.2

In which of the queries would it make sense to use a low isolation level (e.g. READ UNCOMMITED)? And a higher isolation level (e.g. SERIALIZABLE)? Justify your answer.

The first query does natural joins between four tables which is an indicator that this query should use an high isolation level such as SERIALIZABLE.

On the other hand, the second query does a simple group by "Type" and changes to this table would produce just different counts, this type of query could require a lower isolation level.

6.3

Consider the different tuning levels considered in the lectures (schema, index, queries, etc). For the first query, indicate two possible optimizations at two different levels.

Queries formadas a partir de varios natural joins, criam tabelas com um tamanho enorme o que torna as pesquisas mais caras e lentas. Uma forma de otimizar este problema é criar diferentes tabelas temporários com os dados necessários para responder às condições desejadas, por exemplo, a criação de uma query temporária sobre a tabela que responda à condição "WHERE country = 'Portugal'", outra em relação à tabela que responde à condição "WHERE amount $>$ 1000" e por ultimo uma sobre a tabela que responde à condição "WHERE season = 'Winter'".

Após esta otimização é ainda possível criar um indice sobre os atributos desejados, "country", "amount" e "season" de forma a acelerar as pesquisas.