

Professor: Dennis Shasha      Homework 2 - Due: Tuesday, November 22, 2005

Each question is worth 10 points. You may work with one partner and sign both of your names to your paper.

1. The Undo, No Redo algorithm has the following description: Transactions First transfer before-images of each page to the audit trail and then put the after-images in the database.

Suppose we changed it so that we transfer the after-image to the database before putting the before-image to the audit trail. Will this be correct? Prove your answer.

2. Suppose that a transaction manager has done the following steps in the two phase commit protocol for a transaction T:
  - (a) It has asked all servers whether they are willing to commit and they all responded affirmatively.
  - (b) It has told all servers to commit, but has not waited for a response.

Does the transaction manager still need to keep a record of transaction T? If not, why not? If so, for how long?

3. The available copies algorithm will abort a transaction if any site that the transaction read from or wrote to while executing has failed between that time and commit time. Would this still be necessary if there were a separate lock manager that cannot fail. How would you then modify the available copies commit rules given in the class notes? What additional check would be needed if the lock manager could fail?
4. Suppose we are trying to decide whether to put a non-clustering index on attribute B for relation R to support equality selections on B. The relation has 10 million records. Each page can store 10 records. There are 10,000 different values of B. A sequential scan will fetch 10 pages per read. Explain why you would or would not want to include a non-clustering index.
5. A company with 100,000 employees has an employee relation including name, rank, department, salary. The accounting department issues queries concerning the statistics of salaries (their total per department, their average per department, max per department, min per department, etc..) several times a day, so you have decided to construct a redundant relation holding that information. That information must be updated whenever employees get raises, are hired, are demoted, or are fired. Assume the redundant relation is worthwhile now. Decide which of the following events would cause you to reconsider its usefulness. Say why.
  - a. Your locking mechanism changes from page-level locking to record-level locking.
  - b. All secretarial employees are to receive a 10% salary increase. (There are many secretaries; they are scattered over the departments.)
  - c. The four senior vice-presidents are to receive a 20% salary increase.
  - d. You decide to add a non-clustering index on salary to employee.
6. Suppose that each of relations R, S, T, V, W has A as its only key. Which of the following queries may output a different number of records if DISTINCT is removed? Prove your answer.

```
a. SELECT DISTINCT R.A, S.A
FROM R, S
WHERE R.B = S.C
```

```
b. SELECT DISTINCT R.A
FROM R, S
WHERE R.B = S.C
```

```
c. SELECT DISTINCT R.A
FROM R, S
WHERE R.B = S.A
```

```
d. SELECT DISTINCT R.A
FROM R, S, T, V, W
WHERE R.B = S.A
AND R.D = T.D
AND R.C = V.A
AND T.A = S.B
AND W.A = S.D
```

```
e. SELECT DISTINCT R.A
FROM R, S, T
WHERE R.B = T.A
AND R.B = S.C
```

7. Suppose that the following four transactions are the only ones that execute during some interval (R stands for read, W for write, and different letter arguments represent different data items).

T1: R(A) W(B) R(C) W(D)

T2: W(A) W(C)

T3: R(D) R(B)

T4: R(A) R(E)

- What is the finest chopping of T1 assuming that its reads and writes cannot be reordered? Show the chopping.
  - What is the finest chopping of T1 assuming that its reads and writes can be reordered? Show the reordering, then the chopping.
  - Can any other transactions be chopped?
8. Suppose you were designing a concurrency control mechanism for an office system. In this system, users may read the database, modify the relation in real time (by modifying records explicitly) and then write the result back into the database. This would be a single transaction and could take long. For this reason, we want a mechanism that penalizes particularly slow users (e.g. the ones that go out for a coffee break). Decide on a concurrency control mechanism for this environment. Why would two phase locking not be appropriate? How about an optimistic protocol? There is no single correct answer, but your reasons must be backed up by provable properties of the protocols.
9. Tuning assignments (worth 40 points; leave yourself time).

Go to [www.kx.com](http://www.kx.com) Download k2 Go to our web site, download gentable.k from <http://cs.nyu.edu/cs/faculty/shasha/papers/gentable.k>

Also, the sample file salesspec can be found here <http://cs.nyu.edu/cs/faculty/shasha/papers/salesspec>

Alternatively, you can paste it from here:

```
sale 1.0
item 0.2 normal
```

```
customer 0.2 fractal
store 100.0
n 100.0 fractal
n 50.0 normal
```

Then type:

```
k gentable 3000 sales salespec 1
```

You should get a 3000 row table having 3000 different sales ids, about  $0.2 * 3000$  item ids based on a normal distribution, about the same number of customer ids but based on a fractal distribution, 100 storeids distributed uniformly, and then two integer columns having fractal and normal distributions.

Advanced point: Suppose that table T1 has a field A and table T2 has a field B with foreign key T1.A. Suppose that T1.A has 2500 numeric values (perhaps the number of rows in T1). Then the specification for T2.B would be n 2500 If T1.A had 32000 non-numeric values beginning with the string foo, then the specification line for T2.B would be foo 32000

Here are your three assignments. Each is worth 15 points.

#### I. Hotel assignment

```
roomtype(hotelid, roomtypeid, numberavailable, description)
-- hotelid, roomtypeid is the key
== These are the kinds of rooms each hotel has.
```

```
k gentable 3000 roomtype roomtypespec 2
```

```
roomtypespec:
hotel 0.25 normal
n 4.0
n 100.0 fractal
desc 16.0
```

```
inventory(hotelid, roomtypeid, daynumber, numbertaken)
-- hotelid, roomtypeid, daynumber is the key
-- These are the number of rooms that are already taken on a particular day.
-- Some may be invalid either because a given hotel doesn't have a
-- given room type or because the number taken is larger than the
-- number that roomtype says is available.
```

```
k gentable 80000 inventory inventoriespec 3
```

```
inventoriespec:
hotel 2000
n 4.0
day 110
n 5.0
```

```
reservations specify a hotel, roomtypeid and daynumber
```

```
k gentable 1000 reservation reservationspec 0
```

```
reservationspec:  
hotel 0.2  
n 4.0  
day 150
```

If you create a table for the reservations, let the schema be  
reservation( hotelid, roomtypeid, daynumber)

Your mission:

0. Generate the files.
1. Import the files into roomtype and inventory.
2. Decide on your indexes.
3. Eliminate inventory rows that have numbertaken > numberavailable or for which there is no corresponding hotel-roomtype pair.
4. For each row in reservation (for a particular hotel-roomtype-day combination, update the appropriate inventory record by setting numbertaken to numbertaken+1 (unless that exceeds the number available for that night or there is no room of that type in the given hotel). If there is no such appropriate inventory record (in other words, no record in inventory having that hotel-roomtype-day combination, but the roomtype is present), then insert one with numbertaken = 1. This is called an upsert.
5. Consider an alternate implementation in which there is an inventory record for all possible days up to 180 with based on the available roomtypes. In some cases numbertaken is 0. Then the upserts would always be updates. Which is faster?

To generate the alternative table which we'll call inventoryfull

```
inventoryfull(hotelid, roomtypeid, daynumber, numbertaken)
```

You may find the following table to be useful

```
day(daynum)
```

```
k gentable 10000 day dayspec 1  
dayspec:  
day 180
```

## II. Data warehouse assignment

```
lineitem(orderid, itemid, storeid, day, quantity, price)  
-- orderid, itemid, storeid are keys
```

100,000 rows

```

total day range is 100 days

k gentable 100000 lineitem lineitemspec 3

lineitemspec:
order 0.1
item 0.01
store 1000.0 normal
n 100.0
n 50.0
n 1000.0 fractal

storecity(storeid, city)
-- storeid is the key, 100 cities normal distribution
-- 1000 rows

k gentable 1000 storecity storecityspec 1
store 1.0
city 100 normal

citycountry(city, country)
-- cityid is the key, normal distribution in 10 countries
-- 100 rows

k gentable 100 citycountry citycountryspec 1
city 1.0
country 10 normal

countryregion(country,region)
-- country is the key
-- 10 rows
-- two regions uniform distribution

k gentable 10 countryregion countryregionspec 1
country 1.0
region 2.0

itemtype(item, type)
-- item is the key
-- 1000 items, 50 types normal distribution

k gentable 1000 itemtype itemtypespec 1
item 1.0
type 50 normal

typecategory(type, category)
-- type is the key
-- 100 types
-- 10 categories normal distribution

```

```
k gentable 100 typecategory typecategoryspec 1
type 1.0
category 10 normal
```

Your mission:

1. Create these tables.
2. Ensure that foreign key constraints hold.  
Which are they?
3. Decide on indexes and perhaps table redesign to support the following.  
What is the average dollar amount of sales (i.e.,  
per order number) per day in some region over  
a specified day range?  
Which city had the best sales (dollar amount) for some category?
4. Insert 10,000 lineitems.  
Repeat the above.
5. Repeat steps 1-4 on a cold  
system, timing the queries if you use sampling by 10%.  
Also see how accurate this is.

### III. Superlinearity Exercise

```
sales(id, itemid, customerid, storeid, amount, price)
-- id is the key
```

```
k gentable 100000 sales salesspec 1
```

```
salespec:
sale 1.0
item 0.02 normal
customer 0.02
store 0.02
n 100.0 fractal
n 50.0 normal
```

```
item(itemid)
-- itemid is the key
-- 1000 rows
```

```
k gentable 1000 item itemspec 1
item 1.0
```

```
customer(customerid)
-- customerid is the key
-- 1000 rows
```

```
k gentable 1000 customer customerspec 1
customer 1.0
```

```
store(storeid)
```

```
-- storeid is the key
-- 1000 rows

k gentable 1000 store storespec 1
store 1.0
```

You want to produce two tables.  
A sale row goes into  
successfulsales if all of the foreign keys are present.  
Otherwise it goes into unsuccessfulsales.  
Notice that this means that one row with a given saleid  
may go into successfulsales and a different row with  
that same saleid may go into unsuccessfulsales.

We looked at several implementations.  
First we looked at alternatives for building successfulsales:

```
insert successfulsales
select sales.*
from sales
where sales.itemid in (select itemid from item)
and sales.customerid in (select customerid from customer)
and sales.storeid in (select storeid from store)
```

```
insert successfulsales
select sales.*
from sales, item, customer, store
where sales.itemid = item.itemid
and sales.customerid = customer.customerid
and sales.storeid = store.storeid
```

Then we looked at ways to build unsuccessfulsales:

```
insert into unsuccessfulsales
select * from sales;

delete from unsuccessfulsales
where id in (select id from successfulsales)
```

And other ways:

```
insert unsuccessfulsales
select sales.*
from sales
where sales.itemid not in (select itemid from item)
or sales.customerid not in (select customerid from customer)
or sales.storeid not in (select storeid from store)
```

Then we looked at an outer join approach:

```
insert into successfulsales
```

```
select sales.id, item.itemid,  
customer.customerid, store.storeid, sales.amount, sales.price  
from  
((sales left outer join item on sales.itemid = item.itemid)  
left outer join customer on sales.customerid = customer.customerid)  
left outer join store on sales.storeid = store.storeid;  
  
insert into unsuccessfulsales  
select *  
from successfulsales  
where itemid is null  
or customerid is null  
or storeid is null;  
  
delete from successfulsales  
where itemid is null  
or customerid is null  
or storeid is null
```

Maybe you can find even a better method.