Information Organization
Exercises Part 2 (secuirty and transaction, XML)

from the textbook by Ramakrishnan and Gehrke + α

S-1 What is the main idea behind discretionary access control? What is the idea behind mandatory access control? What are the relative merits of these two approaches?

S-2 What is a Trojan Horse attack and how can it compromise discretionary access control? Explain how mandatory access control protects against Trojan horse attacks.

S-3 Describe the privileges recognized in SQL. In particular, describe privileges regarding SELECT, INSERT, UPDATE, DELETE, and REFERENCES. For each privilege, indicate who acquires it automatically on a given table.

S-4 What is the difference between symmetric and public-key encryption? Give examples of well-known encryption algorithms of both kinds. What is the main weakness of symmetric encryption and how is this addressed in public key encryption?

S-5 Explain the need for each of the following limits in a statistical database system:

1. A maximum on the number of queries a user can pose.
2. A minimum on the number of tuples involved in answering a query.
3. A maximum on the intersection of two queries (i.e., on the number of tuples that both queries examine).

T-1 Define these terms: atomicity, consistency, isolation, durability, schedule, dirty read, unrepeatable read, serializable schedule, cascading abort.

T-2 Describe Strict 2PL.

T-3 What is the phantom problem? Can it occur in a database where the set of database objects is fixed and only the values of objects can be changed?

T-4 Consider the following actions taken by transaction T1 on database objects X and Y:

$$R(X), W(X), R(Y), W(Y)$$

1. Give an example of another transaction T2 that, if run concurrently to transaction T1 without some form of concurrency control, could interfere with T1.

2. Explain how the use of Strict 2PL would prevent interference between the two transactions. Use symbol $s(X)$ for a shared-lock on X and $x(X)$ for an exclusive lock on X.

3. Strict 2PL is used in many database systems. Give two reasons for its popularity.

T-5 Consider a database with objects X and Y and assume that there are two transactions T1 and T2. Transaction T1 reads objects X and Y and then writes object X. Transaction T2 reads objects X and Y and then writes objects X and Y.

1. Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a write-read conflict.

2. Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a read-write conflict.

3. Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a write-write conflict.

4. For each of the three schedules, show that Strict 2PL disallows the schedule.

T-6 We call a transaction that only reads database object a read-only transaction, otherwise the transaction is called a read-write transaction. Give brief answers to the following questions:

1. What is lock thrashing and when does it occur?

2. What happens to the database system throughput if the number of read-write transactions is increased?

3. What happens to the database system throughput if the number of read-only transactions is increased?

4. Describe three ways of tuning your system to increase transaction throughput.

T-7 In a *recoverable* schedule, transactions commit only after all transactions whose changes they read commit. Below is an *unrecoverable* schedule:

$$T1 : R(A), T1 : W(A), T2 : R(A), T2 : W(A), T2 : R(B), T2 : W(B), T2 : \mathit{Commit}, T1 : \mathit{Abort}$$

$T2$ has read a value for A that should should never have been there (aborted transactions' efforts are not supposed to be visible to other transactions.) If $T2$ had not yet committed, we could deal with the situation by *cascading* the abort of $T1$ and also aborting $T2$; this process recursively aborts any transaction that read data written by $T2$, and so on. But $T2$ has already committed, and so we cannot undo its actions.

If transactions read only the changes of committed transactions, not only is the schedule recoverable, but also aborting a transaction can be accomplished without cascading the abort to other transactions. Such a schedule is said to *avoid cascading aborts*.

A schedule is said to be *strict* if a value written by a transaction T is not read or overwritten by other transactions until T either aborts or commits. Strict schedules are recoverable, do not require cascading aborts, and actions of aborted transactions can be undone by restoring the original values of modified objects.

Now, consider the following classes of schedules: *serializable, conflict-serializable, view-serializable, recoverable, avoids-cascading-aborts*, and *strict*. For each of the following schedules, state which of the preceeding classes it belongs to. If you cannot decide whether a schedule belongs in a certain class based on the listed actions, explain briefly.

The actions are listed in the order they are scheduled and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort or commit must follow all the listed actions.

1. $T1 : R(X), T2 : R(X), T1 : W(X), T2 : W(X)$
2. $T1 : W(X), T2 : R(Y), T1 : R(Y), T2 : R(X)$
3. $T1 : R(X), T2 : R(Y), T3 : W(X), T2 : R(X), T1 : R(Y)$
4. $T1 : R(X), T1 : R(Y), T1 : W(X), T2 : R(Y), T3 : W(Y), T1 : W(X), T2 : R(Y)$
5. $T1 : R(X), T2 : W(X), T1 : W(X), T2 : \mathit{Abort}, T1 : \mathit{Commit}$
6. $T1 : R(X), T2 : W(X), T1 : W(X), T2 : \mathit{Commit}, T1 : \mathit{Commit}$
7. $T1 : W(X), T2 : R(X), T1 : W(X), T2 : \mathit{Abort}, T1 : \mathit{Commit}$
8. $T1 : W(X), T2 : R(X), T1 : W(X), T2 : \mathit{Commit}, T1 : \mathit{Commit}$
9. $T1 : W(X), T2 : R(X), T1 : W(X), T2 : \mathit{Commit}, T1 : \mathit{Abort}$
10. $T2 : R(X), T3 : W(X), T3 : \mathit{Commit}, T1 : W(Y), T1 : \mathit{Commit}, T2 : R(Y), T2 : W(Z),$
    $T2 : \mathit{Commit}$
11. $T1 : R(X), T2 : W(X), T2 : \mathit{Commit}, T1 : W(X), T1 : \mathit{Commit}, T3 : R(X), T3 : \mathit{Commit}$
12. $T1 : R(X), T2 : W(X), T1 : W(X), T3 : R(X), T1 : \mathit{Commit}, T2 : \mathit{Commit}, T3 : \mathit{Commit}$

T-8 Answer each of the following questions briefly. The questions are based on the following relational schema:

Emp(<u>*eid*: integer</u>, *ename*: string, *age*: integer, *salary*: real, *did*: integer)
Dept(<u>*did*: integer</u>, *dname*: string, *floor*: integer)

and on the following update command:

replace (salary = 1.1 * EMP.salary) where EMP.ename = 'Santa'

1. Give an example of a query that would conflict with this command (in a concurrency control sense) if both were run at the same time. Explain what could go wrong, and how locking tuples would solve the problem.
2. Give an example of a query or a command that would conflict with this command, such that the conflict could not be resolved by just locking individual tuples or pages but requires index locking.
3. Explain what index locking is and how it resolves the preceding conflict.

T-9 Consider a database organized in terms of the following hierarchy of objects: The database itself is an object (D), and it contains two files ($F1$ and $F2$), each of which contains 1000 pages (P1...P1000 and P1001...P2000, respectively). Each page contains 100 records, and records are identified as $p : i$, where p is the page identifier and i is the slot of the record on that page.

Multiple-granularity locking is used, with S, X, IS, IX and SIX locks, and database-level, file-level, page-level and record-level locking. For each of the following operations, indicate the sequence of lock requests that must be generated by a transaction that wants to carry out (just) these operations:

1. Read record P1200:5.
2. Read records P1200:98 through P1205:2.
3. Read all (records on all) pages in file $F1$.
4. Read pages $P500$ through $P520$.
5. Read pages $P10$ through $P980$.
6. Read all pages in $F1$ and (based on the values read) modify 10 pages.
7. Delete record $P1200 : 98$. (This is a blind write.)
8. Delete the first record from each page. (Again, these are blind writes.)
9. Delete all records.

X-1 What is the main concepts of HTML? What are the short comings of HTML, and how does XML address them?

X-2 Explain the following terms and describe what they are used for: HTML, URL, XML, XPath, DTD, cookie, HTTP, HTTPS.

X-3 Give an example of a DTD. Also give an XML document that is valid against the DTD but not well-formed, and vice versa.

X-4 The following pages contain XPath language specification and tutorial:
http://www.w3.org/TR/xpath
http://www.w3schools.com/Xpath/

Construct a sample XML document, and write XPath query examples that use the XPath language constructs.

X-5

1. Design an algorithm for testing whether given XML documents are well-formed.
2. Design an algorithm for testing whether the DOM tree of a given well-formed XML document is valid against a given DTD. Try using nondeterministic finite automata (NFA).
3. Design an algorithm answering XPath queries over given XML documents. You may restrict the language constructs of XPath to a subset (for example, just considering a subset of document axis, and excluding predicates, etc).

X6 Consider the following XML document d_1:

```
<lunch>
  <menu name="ramen" type="noodle">
    <price> 500  </price>
    <options>
      <option price="100"> large_noodle </option>
      <option price="200"> extra_meet </option>
    </options>
  </menu>
  <menu name="udon" type="noodle">
    <price>300</price>
    <options>
      <option price="150">large_noodle</option>
    </options>
  </menu>
</lunch>
```

1. Write the answers to the following XPath queries applied on d_1.

   (a) //*[@price > ”100”]/text()

   (b) options/ancestor::*/price

   (c) *[preceeding-sibling::*/attribute::price]

2. Write an XPath query which returns the following:

(a) Find 'ramen' elements such that **all** of their 'option' descendants have 'price' larger than 100.

(b) Find elements such that whose name is either 'menu' or 'option' and whose attribute value or text value is 'noodle'.

(c) Find elements that have a sibling which has an attribute named 'special'.

3. Write a DTD e such that document d_1 is valid against e.

4. Write a DTD f such that document d_1 is also valid against f, and there is another document d_2 such that d_2 is valid against e but not valid against f. Show such DTD f and document d_2.