Information Organization Exercises Part 2 (Information Retrieval)

S-1

Description

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?

Solution

DAC: A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject.

MAC: Subjects and objects each have a set of security attributes. Whenever a subject attempts to access an object, an authorization rule enforced by the operating system kernel examines these security attributes and decides whether the access can take place. Any operation by any subject on any object will be tested against the set of authorization rules (aka policy) to determine if the operation is allowed.

1. MAC provides access based on levels while DAC provides access based on identity.
2. DAC is more labor intensive than MAC.
3. DAC is more flexible than MAC.
4. MAC access can only be changed by admins while DAC access can be provided by other users.

S-2

Description

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.

Solution

A Trojan horse attack can be designed to accomplish any number of goals, but typically the intent is either pecuniary gain or spreading mayhem. For example, upon bringing the infected file onto your hard drive, the Trojan horse program might locate and send your bank information to the developer. The only limit to what a Trojan horse attack can accomplish is dependent on the limits of the developer's imagination and talent.

Example:

In DAC, there are a high level user A and a low level user B in the system, and a table T which only A can read. Suppose B is malicious and give a Trojan horse program to A which on the surface does some useful work. Now A runs the program, but without the notice of A, the program reads T and writes the content to another table T' which B can read. Thus the information in T is leaked to unauthorized user B.

In MAC, if T' has high security level, then B cannot read it; if it has low security level, the Trojan horse program, which has the same high security level as A, cannot write it.

S-3

Description

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.

Solution

SELECT: Can read all columns (including those added later via ALTER TABLE command).
INSERT (col-name): Can insert tuples with non-null or non-default values in this column. INSERT means same right with respect to all columns.
UPDATE: Can change the data of one or more records in a table. Either all the rows can be updated, or a subset may be chosen using a condition.
DELETE: Can delete tuples.
REFERENCES (col-name): Can define foreign keys (in other tables) that refer to this column.

Owner (Creator or Administrator) can automatically get privilege on a given table. User who was gave privilege through GRANT OPTION by a privileged user.

S-4

Description

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?

Solution

Differences: Symmetric encryption is a single shared, private key between communicating nodes. There is only 1 key involved.

Public Key encryption requires a pair of keys; a public and a private key for exchanging data in a secure manner.

Examples: Symmetric encryption: DES, AES. Public Key encryption: RSA.

Weakness: Encryption key = decryption key; all authorized users know decryption key. Sharing the secret key in the beginning is a problem in symmetric key encryption. It has to be exchanged in a way that ensures it remains secret.

Public key encryption makes use of two keys: a public key and a private key. The public key is made publicly available and is used to encrypt messages by anyone who wishes to send a message to the person that the key belongs to. The private key is kept secret and is used to decrypt received messages.

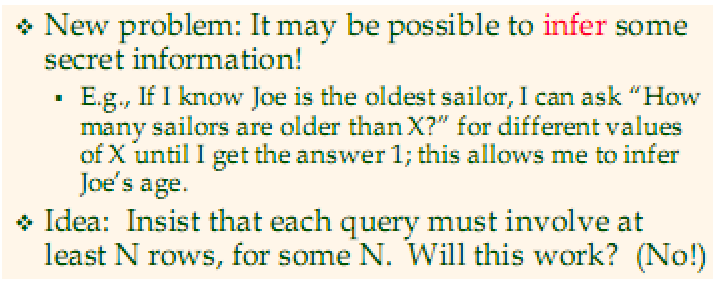
S-5

Description

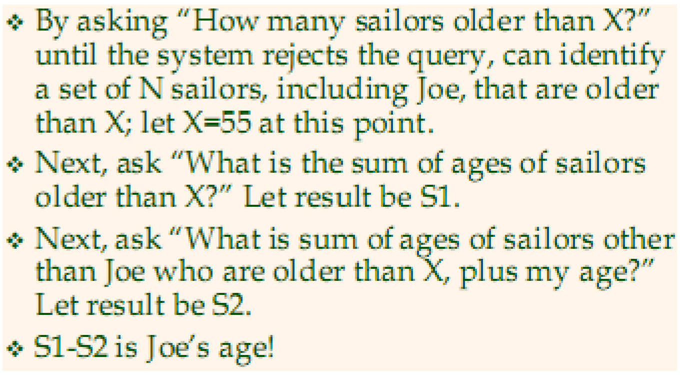
S-5 Explain the need for each of the following limits in a statistical database system:

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).

Solution



- New problem: It may be possible to infer some secret information!
  - E.g., If I know Joe is the oldest sailor, I can ask "How many sailors are older than X?" for different values of X until I get the answer 1; this allows me to infer Joe's age.
- Idea: Insist that each query must involve at least N rows, for some N. Will this work? (No!)



- By asking "How many sailors older than X?" until the system rejects the query, can identify a set of N sailors, including Joe, that are older than X; let X=55 at this point.
- Next, ask "What is the sum of ages of sailors older than X?" Let result be S1.
- Next, ask "What is sum of ages of sailors other than Joe who are older than X, plus my age?" Let result be S2.
- S1-S2 is Joe's age!

T-1

Description

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

Solution

Atomicity (原子性) requires that each transaction is "all or nothing": If one part of the transaction fails, the entire transaction fails, and the database state is left unchanged.

The **consistency (一致性)** property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including but not limited to constraints, cascades, triggers, and any combination thereof.

The **isolation (隔离性)** property ensures that the concurrent execution of transactions results in a system state that could have been obtained if transactions are executed serially.

Durability (耐久性) means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently.

Schedule (or history) of a system is an abstract model to describe execution of transactions running in the system. Often it is a list of operations (actions) ordered by time, performed by a set of transactions that are executed together in the system.

A **dirty read** occurs when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed.

A **non-repeatable read** occurs, when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.

Serializable schedule: A schedule that is equivalent to some serial execution of the transactions.

Cascading abort: A single transaction abort leads to a series of transaction rollback. Strategy to prevent cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule.

T-2

Description

T-2 Describe Strict 2PL.

Solution

Strict Two-phase Locking (Strict 2PL) Protocol:

Each Xact must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing.

All locks held by a transaction are released when the transaction completes.

If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

T-3

Description

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?

Solution

A **phantom read** occurs when, in the course of a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first.

Yes it can.

T-4

Description

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.

Solution

1. If the transaction T2 performed W(Y) before T1 performed R(Y), and then T2 aborted, the value read by T1 would be invalid and the abort would be cascaded to T1 (i.e. T1 would also have to abort.).

2. Strict 2PL would require T2 to obtain x(Y) before writing to it. This lock would have to be held until T2 committed or aborted; this would block T1 from reading Y until T2 was finished, but there would be no interference.

3. Strict 2PL is popular for many reasons. One reason is that it ensures only 'safe' interleaving of transactions so that transactions are recoverable, avoid cascading aborts, etc. Another reason is that strict 2PL is very simple and easy to implement. The lock manager only needs to provide a lookup for exclusive locks and an atomic locking mechanism (such as with a semaphore).

T-5

Description

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.

Solution

1.

T1:	R(x)	R(Y)	W(X)			Abort		
T2:				R(X)	R(Y)		W(X)	W(Y)

2. 有问题

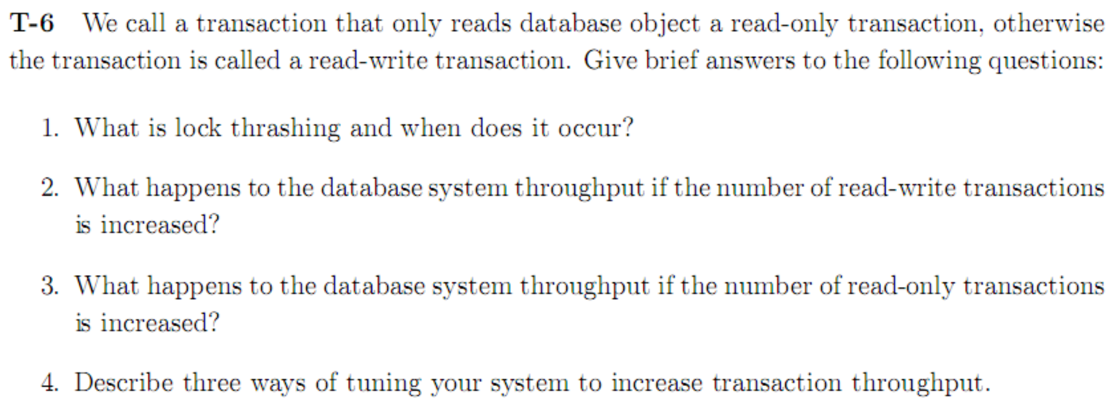
3.

T1:	R(x)	R(Y)						W(X)	C
T2:			R(X)	R(Y)	W(X)	W(Y)	C		

4. 用优先图判断环

T-6

Description



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.

Solution

1. Lock Thrashing is the point where system performance(throughput) decreases with increasing load (adding more active transactions). It happens due to the contention of locks. Transactions waste time on lock waits.

2. At the beginning, with the increase of transactions, throughput will increase, but the increase will stop at the thrashing point, after the point the throughput will drop with the increasing number of transactions.

3. For read-only transactions, there is no lock waiting. The throughput will increase with the increasing number of concurrent transactions.

4. Three ways of tuning your system to increase transaction throughput:

   a. By locking the smallest sized objects possible (reducing the likelihood that two transactions need the same lock).
   b. By reducing the time that transaction hold locks (so that other transactions are blocked for a shorter time).
   c. By reducing hot spots. A hot spot is a database object that is frequently accessed and modified, and causes a lot of blocking delays. Hot spots can significantly affect performance.

T-7

Description

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

$T1 : R(A), T1 : W(A), T2 : R(A), T2 : W(A), T2 : R(B), T2 : W(B), T2 : Commit, T1 : Abort$

$T2$ 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. $T1 : R(X), T2 : W(X), T1 : W(X), T2 : Abort, T1 : Commit$
6. $T1 : R(X), T2 : W(X), T1 : W(X), T2 : Commit, T1 : Commit$
7. $T1 : W(X), T2 : R(X), T1 : W(X), T2 : Abort, T1 : Commit$
8. $T1 : W(X), T2 : R(X), T1 : W(X), T2 : Commit, T1 : Commit$
9. $T1 : W(X), T2 : R(X), T1 : W(X), T2 : Commit, T1 : Abort$
10. $T2 : R(X), T3 : W(X), T3 : Commit, T1 : W(Y), T1 : Commit, T2 : R(Y), T2 : W(Z),$ $T2 : Commit$
11. $T1 : R(X), T2 : W(X), T2 : Commit, T1 : W(X), T1 : Commit, T3 : R(X), T3 : Commit$
12. $T1 : R(X), T2 : W(X), T1 : W(X), T3 : R(X), T1 : Commit, T2 : Commit, T3 : Commit$

Solution

1. Serizability (or view) cannot be decided but NOT conflict serizability. It is recoverable and avoid cascading aborts; NOT strict
2. It is serializable, conflict-serializable, and view-serializable regardless which action (commit or abort) follows It is NOT avoid cascading aborts, NOT strict; We can not decide whether it's recoverable or not, since the abort/commit sequence of these two transactions are not specified.
3. It is the same with 2.
4. Serizability (or view) cannot be decided but NOT conflict serizability. It is NOT avoid cascading aborts, NOT strict; We can not decide whether it's recoverable or not, since the abort/commit sequence of these transactions are not specified.
5. It is serializable, conflict-serializable, and view-serializable; It is recoverable and avoid cascading aborts; it is NOT strict.
6. It is NOT serializable, NOT view-serializable, NOT conflict-serializable; it is recoverable and avoid cascading aborts; It is NOT strict.
7. It belongs to all classes
8. It is serializable, NOT view-serializable, NOT conflict-serializable; It is NOT recoverable, therefore NOT avoid cascading aborts, NOT strict.
9. It is serializable, view-serializable, and conflict-serializable; It is NOT recoverable, therefore NOT avoid cascading aborts, NOT strict.
10. It belongs to all above classes.
11. It is NOT serializable and NOT view-serializable, NOT conflict-serializable; it is recoverable, avoid cascading aborts and strict.
12. It is NOT serializable and NOT view-serializable, NOT conflict-serializable; it is recoverable, but NOT avoid cascading aborts, NOT strict.

T-8

Description

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

Emp(*eid*: integer, *ename*: string, *age*: integer, *salary*: real, *did*: integer)
Dept(*did*: integer, *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.

Solution

1. select salary from EMP where EMP.ename='Santa'
2. insert into EMP values(2, 'Santa', 18, 10000, 2)
3. Index locking is a technique used to maintain index integrity. A portion of an index is locked during a database transaction when this portion is being accesses by the transaction as a result of attempt to access related user data.

   When a portion of index is locked by a transaction, other transactions may be blocked from accessing this index portion (blocked from modifying, and even from reading it, depending on lock type and needed operation)

T-9

Description

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.

Solution

1. D: IS F2: IS P1200: IS P1200:5: S
2. P1200: IS P1200:98: S
3. D: IS F1: S
4. P500: S
5. P10: S
6. D: SIX F1: SIX 10pages: X
7. D: IX F2: IX P1200: IX P1200:98: X
8. D: IX F1: IX F2: IX all pages: IX P1:1, P2:1, …, P2000:1: X
9. D: IX F1: IX F2: IX P1, P2, …, P2000: X

X-1

Description

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

Solution

Hypertext Markup Language (HTML) is the main markup language for displaying web pages and other information that can be displayed in a web browser.

HTML does not provide meaning to the data contained in HTML tags. HTML cannot accommodate highly refined searches across data.

XML is extensible; it has limitless ability to define new languages or data sets. You can include your data and a description of what the data represents. This is useful for defining your own language or protocol.

X-2

Description

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

Solution

Hypertext Markup Language (**HTML**) is the main markup language for displaying web pages and other information that can be displayed in a web browser.

In computing, a uniform resource locator (**URL**) (originally called universal resource locator) is a specific character string that constitutes a reference to an Internet resource.

Extensible Markup Language (**XML**) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

XPath, the XML Path Language, is a query language for selecting nodes from an XML document. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document.

A Document Type Definition (**DTD**) is a set of markup declarations that define a document type for an SGML-family markup language (SGML, XML, HTML).

A **cookie** is usually a small piece of data sent from a website and stored in a user's web browser while a user is browsing a website. When the user browses the same website in the future, the data stored in the cookie can be retrieved by the website to notify the website of the user's previous activity. Cookies were designed to be a reliable mechanism for websites to remember the state of the website or activity the user had taken in the past. This can include clicking particular buttons, logging in, or a record of which pages were visited by the user even months or years ago.

The Hypertext Transfer Protocol (**HTTP**) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

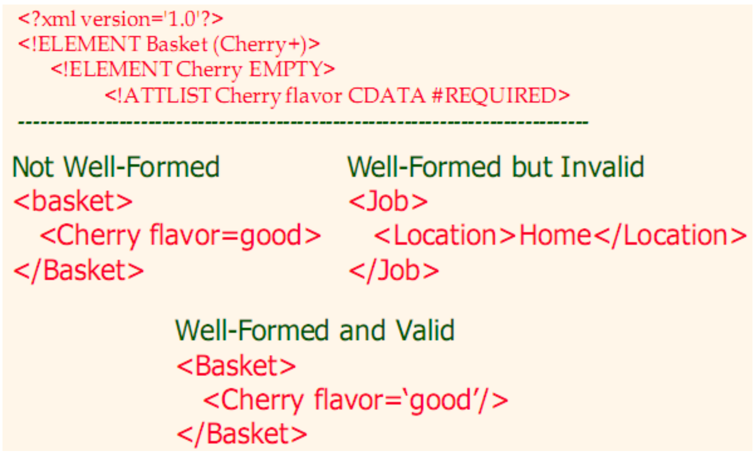
Hypertext Transfer Protocol Secure (**HTTPS**) is a widely used communications protocol for secure communication over a computer network, with especially wide deployment on the Internet. Technically, it is not a protocol in itself; rather, it is the result of simply layering the Hypertext Transfer Protocol (HTTP) on top of the SSL/TLS protocol, thus adding the security capabilities of SSL/TLS to standard HTTP communications.

X-3

Description

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.

Solution



```
<?xml version='1.0'?>
<!ELEMENT Basket (Cherry+)>
    <!ELEMENT Cherry EMPTY>
        <!ATTLIST Cherry flavor CDATA #REQUIRED>
--------------------------------------------------------------------------
```

Not Well-Formed

```
<basket>
  <Cherry flavor=good>
</Basket>
```

Well-Formed but Invalid

```
<Job>
  <Location>Home</Location>
</Job>
```

Well-Formed and Valid

```
<Basket>
  <Cherry flavor='good'/>
</Basket>
```

X-4

Description

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

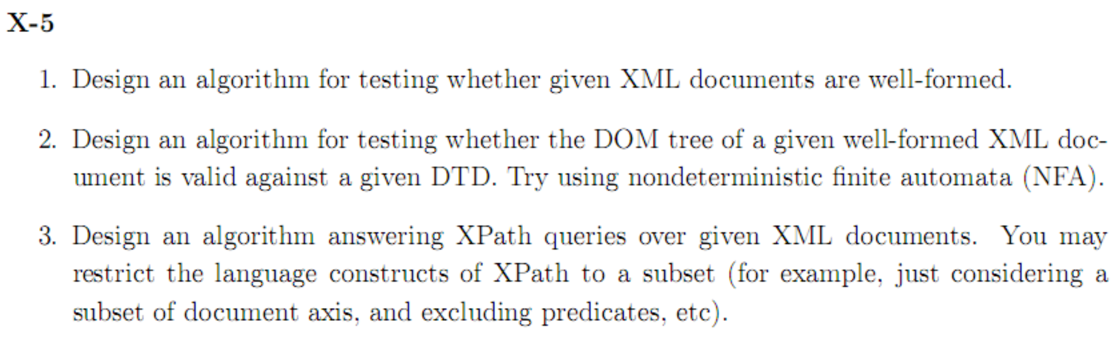
Construct a sample XML document, and write XPath query examples that use the XPath language constructs.

Solution

So…easy…

X-5

Description



X-5

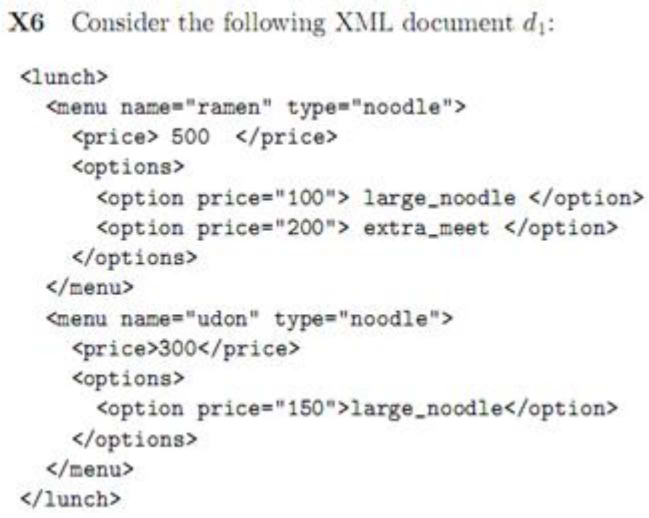
1. Design an algorithm for testing whether given XML documents are well-formed.
2. Design an algorithm for testing whether the DOM tree of a given well-formed XML document is valid against a given DTD. Try using nondeterministic finite automata (NFA).
3. Design an algorithm answering XPath queries over 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).

Solution

So…hard…

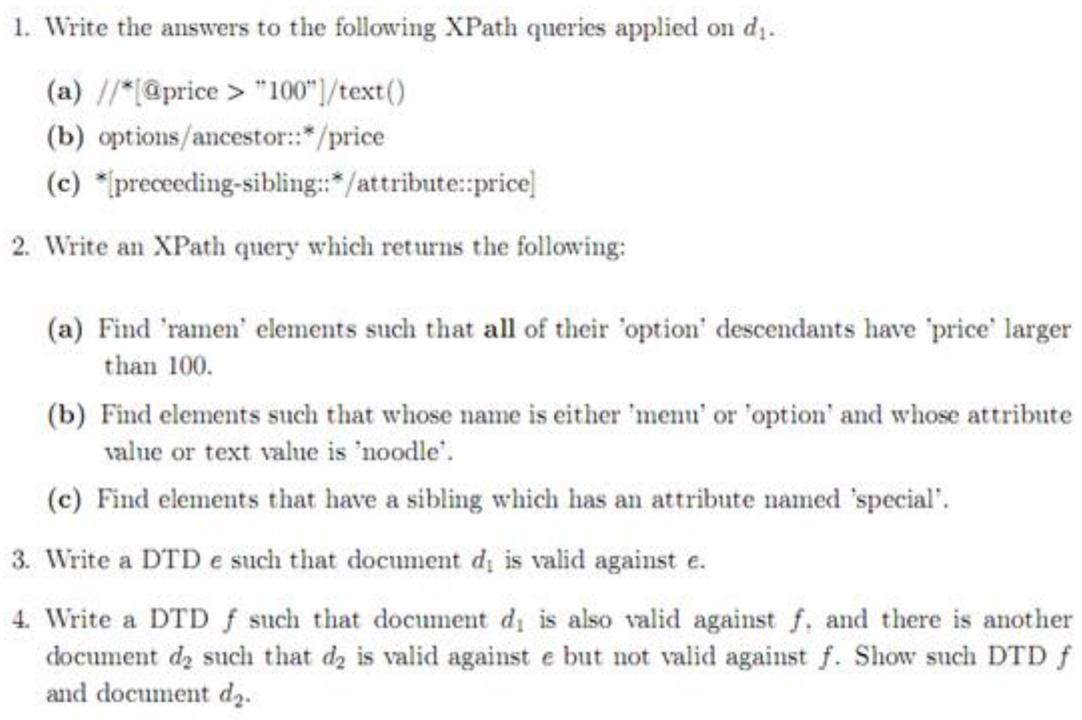
X-6

Description



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) *[preceding-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.

Solution

1. (a). large_noodle   extra_meet
   (b). <price>500</price>   <price>300</price>
   (c). <option price="200">extra_meet</option>

2. *[@name="ramen"] [option/@price>"100"]
   * [name() = menu|option] [@*="noodle" | text()="noodle"]
   * [preceding-sibling::* | following-sibling::* /attribute::special]

3.
```
<?xml version='1.0' ?>
<!ELEMENT lunch(menu*)>
   <!ELEMENT menu(price*, options*)>
   <!ATTLIST menu name CDATA #REQUIRED>
   <!ATTLIST menu type CDATA #REQUIRED>
      <!ELEMENT price EMPTY>
      <!ELEMENT options(option*)>
         <!ELEMENT option EMPTY>
         <!ATTLIST option price CDATA #REQUIRED>
```

4. f :   将 e 中的*改成+
   d2:   <lunch></lunch>

附加题 1

Exercise 17.4 Consider the following sequences of actions, listed in the order in which they are submitted to the DBMS:

- Sequence S1: T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1: Commit, T2:Commit, T3:Commit
- Sequence S2: T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

For each sequence and for each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the sequence.

Assume that the timestamp of transaction T_i is i. For lock-based concurrency control mechanism, add lock and unlock requests to the above sequence of actions as per the locking protocol. If a transaction is blocked, assume that all of its actions are queued until it is resumed; the DBMS continues with the next action (according to the listed sequence) of an unblocked transaction.

1. Strict 2PL with timestamps used for deadlock prevention.
2. Strict 2PL with deadlock detection. Show the waits-for graph if a deadlock cycle develops.

Answer:

1. Assume we use Wait-Die policy.

   Sequence S1:

   T1 acquires shared-lock on X;

   for an exclusive-lock on X, since T2 has a lower priority, it will be aborted When T2 asks;

   T3 now gets exclusive-lock on Y;

   When T1 also asks for an exclusive-lock on Y, which is still held by T3, since T1 has higher priority, T1 will be blocked waiting;

   T3 now finishes write, commits and releases all the lock;

   T1 wakes up, acquires the lock, proceeds and finishes;

   T2 now can be restarted successfully.

   Sequence S2:

   The sequence and consequence are the same with sequence S1, except T2 was able to advance a little more before it gets aborted.

2. In deadlock detection, transactions are allowed to wait, they are not aborted until a deadlock has been detected. (Compared to prevention schema, some transactions may have been aborted prematurely.)

   Sequence S1:

   T1 gets a shared-lock on X;

   T2 blocks waiting for an exclusive-lock on X;

   T3 gets an exclusive-lock on Y;

   T1 blocks waiting for an exclusive-lock on Y;

   T3 finishes, commits and releases locks;

   T1 wakes up, get an exclusive-lock on Y, finishes up and releases lock on X and Y;

   T2 now gets both an exclusive-lock on X and Y, and proceeds to finish.

   No deadlock.

   Sequence S2:

   There is a deadlock. T1 waits for T2, while T2 waits for T1.

附加题 2

Exercise 18.4 Consider the execution shown in the figure below.

LSN	LOG
00	update: T1 writes P2
10	update: T1 writes P1
20	update: T2 writes P5
30	update: T3 writes P3
40	T3 commit
50	update: T2 writes P5
60	update: T2 writes P3
70	T2 abort

1. Extend the figure to show prevLSN and undonextLSN values.
2. Describe the actions taken to rollback transaction T2.
3. Show the log after T2 is rolled back, including all **prevLSN** and **undonextLSN** values in log records.

Answer:

Question 1:

LSN	LOG	prevLSN	undoNextLSN
00	Update: T1 writes P2	-	-
10	Update: T1 writes P1	00	00
20	Update: T2 writes P5	-	-
30	Update: T3 writes P3	-	-
40	T3 commit	30	Not an update log record
50	Update: T2 writes P5	20	20
60	Update: T2 writes P3	50	50
70	T2 abort	60	Not an update log record

Question 2:

Step 1: Restore P3 to the before-image stored in LSN 60.

Step 2: Restore P5 to the before-image stored in LSN 50.

Step 3: Restore P5 to the before-image stored in LSN 20.

Question 3:

The log tail should look something like this:

LSN	prevLSN	transID	Type	pageID	undoNextLSN
80	70	T2	CLR	P3	50
90	80	T2	CLR	P5	20
100	90	T2	CLR	P5	-
110	100	T2	END	-	-