Midterm2 Review

Practice Test Questions

Question1

Imagine you are given the task of creating a database for a hospital. Among other things, you need to keep track of information about patients, doctors, and the relationships between them. If you decide to use an XML database,

describe two different ways that you could capture the relationships between patients and doctors in the database

Answer

We can either choose to use IDREFS and have attributes act as the relation or make a child element nested in the parent. Thus we have 4 answers

- Patients has a child element called doctor for each doctor that has operated on them
- Patients has idrefs to the `doctor_id` (say) in the doctor table

OR

- doctor has a child element of patients
- doctor has an idref to patients

Question2

Questions 2-5
Recall the XML version of our movie database. The root element of the database is called `imdb`, and that root element has three child elements:

- one called `movies`, which has a nested `movie` element for each movie in the database
- one called `people`, which has a nested `person` element for each movie in the database
- one called `oscars`, which has a nested `oscar` element for each Oscar award in the database.

Here is an example of one of the `movie` elements:

```
<movie id="M0120338" directors="P0000116"
       actors="P0000138 P0000701 P0000708 P0000870 P0000200"
       oscars="O19980000000 O19980000116">
  <name>Titanic</name>
  <year>1997</year>
  <rating>PG-13</rating>
  <runtime>194</runtime>
  <genre>DR</genre>
  <earnings_rank>9</earnings_rank>
</movie>
```

The actors attribute above is an example of what type of attribute? How is this type of attribute similar to a foreign key? How is it different?

Answer 2

- Actors is a IDREFS attribute used to capture relations with another document or database in XML. It's similar to foreign keys in linking the Movies to the Peoples who acted in them however foreign keys are unique for each element, IDREFS are not

Question 3

Write an XPath expression that obtains the names of all movies with an R rating from the 1990s. (There are many possible answers here. See if you can come up with at least two!)

Answer 3

- `//movie[rating="R" and year>=1990 and year < 1999]/name`
- `//movie[rating="R" and contains(year, "199")]/name`
- `//movie/name[../rating="R" and ../year>=1990 and ../year <=1999]`

Question 4

Write a FLWOR expression that solves the same problem as Question 3. Order the results alphabetically by name.

Answer4

```
for $m in //movie[rating="R" and year>=1990 and year<=1999]
order by $m/name
return $m/name

for $m in //movie
where $m[rating="R" and year>=1990 and year<=1999]
```

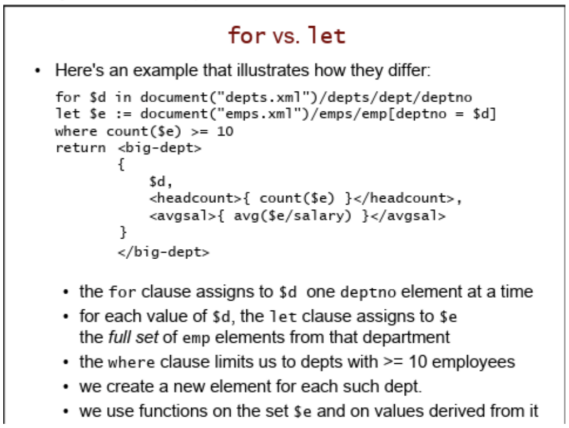
```
order by $m/name
return $m/name
```

Question 5

Write a FLWOR expression that finds the average runtime of all movies with an R rating from the 1990s. Your query should return a single element of type average

Answer 5

The way I think about it is that a let clause is sort of like a nested for loop but it returns the full set instead of one variable at a time, it allows you to do things like use aggregates. See images from the coursepack below.



for vs. let

- Here's an example that illustrates how they differ:

```
for $d in document("depts.xml")/depts/dept/deptno
let $e := document("emps.xml")/emps/emp[deptno = $d]
where count($e) >= 10
return <big-dept>
       {
           $d,
           <headcount>{ count($e) }</headcount>,
           <avgsal>{ avg($e/salary) }</avgsal>
       }
       </big-dept>
```

- the `for` clause assigns to `$d` one `deptno` element at a time
- for each value of `$d`, the `let` clause assigns to `$e` the *full set* of `emp` elements from that department
- the `where` clause limits us to depts with >= 10 employees
- we create a new element for each such dept.
- we use functions on the set `$e` and on values derived from it

I think this may be helpful to other students, so I'll go ahead and make this public!

```
let $e := //movies/movie[rating="R" and contains(year, "19
9")]/runtime
return <avg_runtime>avg($e)</avg_runtime>
```

Question 6-9

Questions 6-9

Here is an example of one of the `person` elements from our XML movie database:

```
<person id="P0000243" directed="M2671706"
    actedIn="M0097441 M0107818 M0139654 M2671706"
    oscars="O20020000243 O19900000243">
  <name>Denzel Washington</name>
  <dob>1954-12-28</dob>
  <pob>Mount Vernon, New York, USA</pob>
</person>
```

6. Recall that a person element only includes the `directed` attribute if that person is a director – i.e., if they have directed one or more of the movies in the database. Write a query that returns the `name` and `pob` elements of all directors who were born in New York state (i.e., whose `pob` value ends with "New York, USA".

7. Write a query that produces, for every movie directed by a person born in New York state, a pair of elements for the name of that movie and the name of the director. Since both of the relevant elements are called `name`, you should reformat the results so that the element for the movie's name is called `movie` and the element for the director's name is called `director`.

8. Now write a query that produces, for each director born in New York state, a new element of type `ny_director` that each have the following nested child elements: the existing `name` element of the director, one called `birthplace` for their place of birth, and one or more elements of type `directed`, each of which has as its value the name of one of the movies that the person directed.

9. Finally, revise your query for Problem 8 so that you only include a director in the results if they have directed more than one of the movies in the database.

Answer 6-9

6)

```
for $p in //person[@directed]
where contains(pob, "New York, USA")
return ($p/name, $p/pob, " ")

for $p in //person[@directed and contains(pob, "New York, US
A")]
return ($p/name, $p/pob, " ")
```

7)

```
for $p in //person[@directed],
        $m in //movies
where contains(pob, "New York, USA") and contains($p/@directe
d, $m/@id)
return <movie>{string ($m/name)}</movie>, <director>{string
($p/name)}</director>
```

8)

```
for $p in //person[@directed]
return
<ny_director>{
    $p/name
    <birthplace>string($p/pob)</birthplace>
    for $m in //movie
    where contains($p/@directed, $m/@id)
    return <directed> {string($m/name)}</directed>
}
</ny_director>
```

9)

```
for $p in //person[@directed]
let $m:= //movie[contains($p/@directed, $m/@id)]
where coutn($m) > 1
return
<ny_director>{
    $p/name
    <birthplace>string($p/pob)</birthplace>
    for $m in //movie
    where contains($p/@directed, $m/@id)
    return <directed> {string($m/name)}</directed>
}
</ny_director>
```

Questions 10-11

Questions 10 and 11
Consider the following relation:

Staff(id CHAR(5) PRIMARY KEY, name VARCHAR(30),
status VARCHAR(10), salary REAL, specialty_type VARCHAR(20))

Assume that we're taking the approach to marshalling from PS 3. The primary-key value (i.e., the value of *id*) is stored in the key portion of the key/value pair, and the rest of the column values are stored in the value portion, which is a record that begins with field offsets. In addition, you should assume that we're using 1-byte characters, 2-byte offsets, and an 8-byte double for the salary.

10. What would the marshalled key and value look like for the following tuple?
    `('12345', 'Doogie Howser', 'MD', NULL, 'GP')`

11. What steps would the DBMS have to take to unmarshall (i.e., extract) the value of the status field from an arbitrary marshalled tuple of this relation?

Answer 10-11

10)

```
mashalled values
(5 + 1)*2 = 12 byte offset

|-2|12|25| -1|27|29|Doogie Howser|MD|GP
```

11)

- To unmarshall, first we'll find the column of status
- We need to first get to status which is the 4th offset (2*4)since its the second column
- Read 2 bytes at the offset to get to offset value of 25
- Since status is not primary column, and its archer, we loop until no non-null values are left or non -2, and then read the length of the fields value which is 27-25 = 2

- Read 2 bytes from offset S to get the value of the filed

Answer 12-14

Questions 12-14
Consider the following schedule involving two transactions, T1 and T2.

T1	T2
r(M)	
	r(N)
	w(N)
r(N)	
w(M)	
commit	
	commit

12. Explain briefly why this schedule is *not* recoverable.

13. Describe what change or changes you would need to make to turn this schedule into a recoverable schedule.

14. Is the original schedule serializable? Explain briefly why or why not.

Answer 12-14

12)

- we have a dirty read which is why the schedule is not recoverable, recoverability ensures that if T2 crashes the system does not go into an inconsistent state however here since T1 commits before T1 we dont know if after restarting T2, T1 would behave the same way or not.

13) To become consistent, either T2 can commit before T1 so we can move T1's commit after T2's or we can commit T2 right after the `w(N)` and before T1's `r(N)`

14) The original schedule is serializable since its conflict serializable.

Question 15-19

Questions 15-20
Consider the following potential schedule involving two transactions:

s1; r1(X); s2; r2(X); w1(Y); r2(Y); w2(Y); w2(Z); c2; w1(Z); c1

15. Would this schedule execute to completion if it were attempted on a system that uses rigorous two-phase locking and no update locks? If so, show the full schedule in table form including lock requests, unlock actions, and any times when transactions are made to wait. If not, show the partial schedule and explain why it cannot be completed. Assume that transactions acquire a lock immediately before they first need it and upgrade shared locks as needed.

16. Repeat the previous problem, but change T1's write of Z to be a write of X.

17. Would the original schedule above execute to completion if it were attempted on a system that uses timestamp-based concurrency control *without* commit bits? If so, show the full schedule in table form including columns for the state of the data items. If not, show the partial schedule and explain why it cannot be completed. Assume timestamps of 10 and 20 are assigned to the transactions.

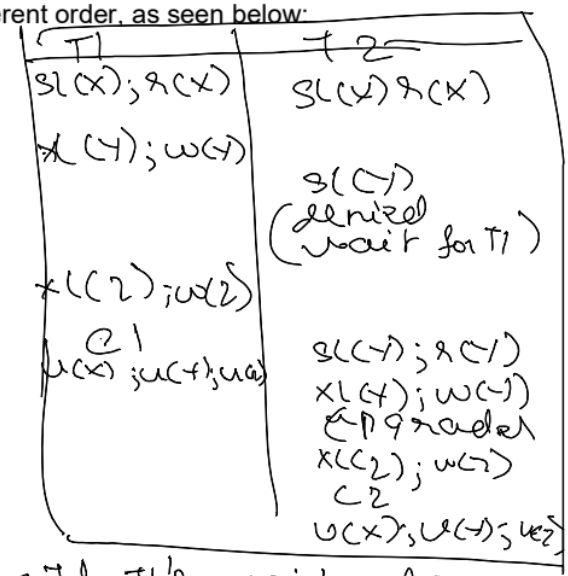
18. Repeat Question 17 for a system that uses timestamp-based concurrency control *with* commit bits.

19. Repeat Question 17 for a system a system that uses *multiversion* timestamp-based concurrency control *without* commit bits.

20. Repeat Question 17, but change T1's write of Z to be a write of X.

15. It would complete, but with some actions in a different order, as seen below:

T1	T2
sl(X); r(X)	
	sl(X); r(X)
xl(Y); w(Y)	
	sl(Y) denied; wait for T1
xl(Z); w(Z) commit u(X); u(Y); u(Z)	
	sl(Y); r(Y) xl(Y); w(Y) xl(Z); w(Z) commit u(X); u(Y); u(Z)



16. The revised schedule would produce deadlock:

T1	T2
sl(X); r(X)	
	sl(X); r(X)
xl(Y); w(Y)	
	sl(Y) denied; wait for T1
xl(X) denied; wait for T2	

If T1's write of Z changes to X, we would try for an xl of x but get a deadlock

17. It would complete in the original order, as seen below:

T1	T2	X	Y	Z
		RTS = WTS = 0	RTS = WTS = 0	RTS = WTS = 0
TS = 10				
r(X)		RTS = 10		
	TS = 20			
	r(X)	RTS = 20		
w(Y)			WTS = 10	
	r(Y)		RTS = 20	
	w(Y)		WTS = 20	
	w(Z)			WTS = 20
	commit			
w(Z); ignore				
commit				

→ with commit bit, will wait for T since c=F

18. It would complete, but with some actions in a different order:

T1	T2	X	Y	Z
		RTS = WTS = 0	RTS = WTS = 0	RTS = WTS = 0
		c = true	c = true	c = true
TS = 10				
r(X)		RTS = 10		
	TS = 20			
	r(X)	RTS = 20		
w(Y)			WTS = 10; c = false	
	r(Y) denied; wait			
w(Z)				WTS = 10; c = false
commit			c = true	c = true
	r(Y)		RTS = 20	
	w(Y)		WTS = 20; c = false	
	w(Z)			WTS = 20; c = false
	commit		c = true	c = true

19. It would complete in the original order:

T1	T2	X(0)	Y(0)	Y(10)	Y(20)	Z(0)	Z(10)	Z(20)
		RTS = 0	RTS = 0			RTS = 0		
TS = 10								
r(X)		RTS = 10						
	TS = 20							
	r(X)	RTS = 20						
w(Y)				created; RTS = 0				
	r(Y)			RTS = 20				
	w(Y)				created; RTS = 0			
	w(Z)							created; RTS = 0
	commit							
w(Z)							created; RTS = 0	
commit								

Question 21

What does it mean for a schedule to be cascadeless? How can a DBMS guarantee this property if it uses locks for concurrency control? How can a DBMS guarantee this property if it uses a timestamp-based approach?

Answer 21

Cascadeless means that rolling back one transaction does not lead to a cascading rollback of another one. For a DBMS, using strict or rigorous 2PL ensures cacadelessness, for multi-version timestamps we need to use commit bits.

Question 22-25

Questions 22-25
Imagine that you are charged with implementing a bank's database, and that the database will be distributed across the bank's 6 branches.

22. You decide to replicate the table that contains account balances across the 6 branches using synchronous replication. How would you explain the decision to use replication to the bank's managers? Next, how would you explain to them why you have decided to use *synchronous* replication rather than asynchronous replication?

23. You are planning to use voting-based replication, and your intern has proposed three possible configurations for the voting:
    a. update 4 copies, read 3 copies
    b. update 2 copies, read 4 copies
    c. update 3 copies, read 5 copies

    Which of these would work if you configure the system to use primary-copy locking?

24. How would your answer to the previous question change if you used fully distributed locking instead?

25. Which of the six configurations from the previous two questions would you recommend if you know that the bank's workload includes a large number of updates? There may be more than one possible good choice, and you should discuss the advantages and drawbacks of your chosen configuration.

Answer 22-25

22)

- Synchronous replication because user's need to be updated on money matters, everyone should read the latest updates and all branches should be

updated in real time.

- replication allows for
  - fault tolerance
  - load balancing
  - efficient computation instead of a centralized source.

23)

`a and c` would be used, b would not be because we could update 2 copies and read from the remaining 4

24) Since we are not using primary locking, a single machine does not acquire exclusive or shared locks we need a majority of locks. For voting, we can clearly see that only `a` works because we have more than 3 updates `w>n/2` and proper number of read `r>n-w`

25) We should use 23 c in case of primary copy locking since it only required one copy to be locked and useful for expensive workload updates. We have reduced availability because if the primary site is unavailable we cannot do anything. however its less expensive

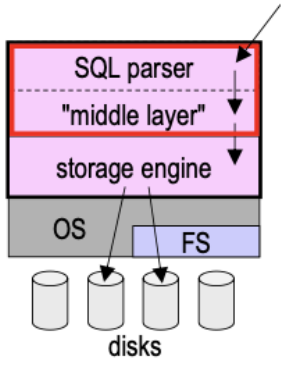
using 24 a, we need to update an extra copy however we never have to worry about 1 failure making reads and writes impossible.

Berkeley DB practice

- Catalog: a class that maintains the per-table metadata the methods are static
- one example of a type of noSQL database known as a key-value store.
- **BDB: related databases are grouped together into an environment.**

Your Task

- On the homework, you will implement portions of the logical-to-physical mapping for a simple relational DBMS.
- We're giving you:
  - a SQL parser
  - a storage engine: Berkeley DB
  - portions of the code needed for the mapping, and a framework for the code that you will write
- In a sense, we've divided the logical layer into two layers:
  - a SQL parser
  - everything else – the "middle layer"
    - you'll implement parts of this



The `Table` and `Column` classes

The `Table` class is a blueprint for objects that represent one of the tables in the database. The `Column` class is a blueprint for objects that represent one of the columns in a table.

Review:

- the API of the `Table` and `Column` classes
- the actual code in those classes.

and answer the following questions:

1. What does a `Table` object's `open()` method do? If you try to call this method for a table that doesn't exist, what does the method return? How should your code deal with that return value?
2. What `Table` method allows you to determine whether a table has a primary key?
3. What `Table` method allows you to obtain the columns associated with the table? What method call would you use to obtain the leftmost column in the table?
4. What `Column` method allows you to determine the data type of the column? What are its possible return values?
5. What `Column` method allows you to determine the length of the values that will go in that column? What does it return for `VARCHAR` columns?

1)

- `open()` lets you open an table to start the underlying database for **existing tables only**
- if the table doesn't exits we get an error `OperationStatus.NOTFOUND`

2) `primaryKeyColumn()` determines whether current column is primary key or not

3) `getColumn()` returns the column, for leftmost column get pass in 0 to this function

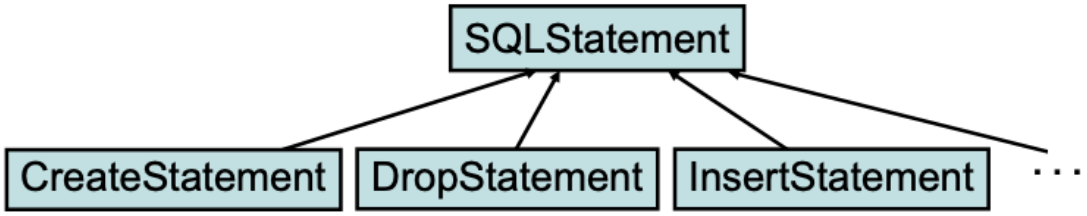
4) For the datatype of the column, run `getType()`, we get

- Column.CHAR
- Column.INTEGER
- Column.REAL
- Column.VARCHAR

5) We need to run the `getLength()` method to get the column's length, for VARCHAR, return maximum possible length it can acquire

The Parser

- Takes a string containing a SQL statement
- Creates an instance of a subclass of the class `SQLStatement`:



- `SQLStatement` is an *abstract class*.
  - contains fields and methods inherited by the subclasses
  - includes an *abstract* `execute()` method
    - just the method header, not the body
- Each subclass implements its own version of `execute()`
  - you'll do this for some of the subclasses

The `SQLStatement` class and its subclasses

`SQLStatement` is an abstract superclass that includes the fields and methods needed by one or more types of SQL commands. Each type of SQL command is represented by a different subclass of `SQLStatement`, and the subclasses inherit much of their functionality from `SQLStatement`.

Review:

- the API of the `SQLStatement` class
- the actual code in that class

and answer the following questions:

6. What method allows you to determine the table or tables on which a given SQL command should operate?

7. What methods allow you to obtain information about the other components of the command (e.g., the columns)?

6) `getTable()`

7) We have functions to get

- column - `getColumn()`
- value in column - `getColumnVal()`
- get where the column
- get number of columns and number of values in it

8. Every column will have an offset in the offset table. This includes the primary key–even though it will be stored in the value portion of the key/value pair. What offset should you use for the primary key?

9. What offset should you use for a `NULL` value?

10. Consider the following sequence of SQL commands:

    ```
    CREATE TABLE Foo (a INT PRIMARY KEY, b VARCHAR(20)
    INSERT INTO Foo VALUES (1, 'hello');
    INSERT INTO Foo VALUES (2, NULL);
    ```

    Describe the key/value pairs that the marshalling scheme would create for the two rows inserted by the `INSERT` commands above. For each row, what will the key look like and what will the value would look like?

11. Now consider the following sequence of SQL commands:

    ```
    CREATE TABLE Bar (a INT, b CHAR(4) PRIMARY KEY, c
    INSERT INTO Bar VALUES (1, '1234', 'hello', 12.5);
    INSERT INTO Bar VALUES (2, '4567', 'wonderful', NU
    ```

    Describe the key/value pairs that your marshalling scheme would create for the two rows inserted by these `INSERT` commands.

12. The`RowOutput` methods that you will use for writing offsets and column values are inherited from the `DataOutputStream` class, so you should make sure to review the API of that class.

    Which methods of this class will you use, and for what purpose will you use each method?

- We wil use -2 for primary key offset
- The function to calculate the offset is in the `insertRow()` class to marshal data.
  - We first get the total number of columns and multiply by 2 to get the size of the offset bytes
  - then in a for loop we check

        - if the column is a primary key column and assign it -2
        - if the column is null, assign the index -1
        - otherwise, we have a running total offset length variable which calculates the current offset we are on. Depending on the type of the column, we return either run the `col.getLength()` function of just the string length function for VarChar.
        - the for loop iterates through the column length
        - finally we set `offset[offset.length-1] = currOffsetLengthCalc`
    - We make 2 buffers, one for the key and one for the value and depending on the type of the data, we use the `writeShort(), writeInt(), writeByte()` functions

```
private int getLengthForColumn(int i) throws IOException{
        try {
            Column col = this.table.getColumn(i);
            int typeOfCol = col.getType();
            // TEST:System.out.println("Column type is  "+
typeOfCol + " value is " + col);
            if (typeOfCol != 3){
                // WARN: using inbuilt function if not a v
archar
                // System.out.println("Length of the colum
n is " + col.getLength());
                return col.getLength();
            } else {
                // WARN: if varchar, then return the lengt
h of the string after casting
                // System.out.println("Length of the colum
n is " + ((String)columnVals[i]).length());
                return ((String)columnVals[i]).length();
            }
        } catch (Exception e) {
            System.out.println("Error in getting the lengt
```

```
 h of the column" + e);
         }
       }
```

9) For null, we use `-1`

10)

```
(2+1)*3
|-2| 6| 11| hello|
Value(2, null)
|-2| -1| 6|
```

12)

```
  writeShort() for offset values, which are 2 bytes
  writeInt() for integer data values
  writeDouble() for real numbers
    writeBytes() for CHAR and VARCHAR values.
```

Unmarshalling

You will add code to the provided `TableIterator` class to perform the necessary unmarshalling. To do so, you will make use of `RowInput` objects, which allow you to read values from an underlying buffer (i.e., a byte array).

Review:

- the lecture notes on unmarshalling
- the API of the `TableIterator` and `RowInput` classes
- the actual code in those classes

and answer the following questions:

13. How will you retrieve a particular column value from a row? In other words, will you go from a key/value pair to the value of a specific column?

14. What methods from the `RowInput` class will you use to read a value from the buffer that contains the marshalled values?

15. When unmarshalling, how will you determine the length of a `VARCHAR` attribute so that you can read in the appropriate number of bytes?

- getColumnVal() is the function to get the column value
  - if -1 return null
  - of -2, read from the key portion
  - otherwise read from the value portion and we have types depending on INT, DOUBLE, CHAR or VARCHAR.
    - For varchar, iterate until you avoid all -1 or -2 values, read the length of that offset, get the difference in offset lengths and return this difference from the original offset

14) functions like

- readShortAtOffset - read 2*length of offsets to get the input value from which we start using the nextShort() or nextInt() etc

- readIntAtOffset
- readDoubleAtOffset
- readBytesAtOffset

17) What remains to be done in the rest of the `execute()` method, and what Berkeley DB methods will you need to use in the code that you write to complete that method?

- Get row output objects for the key and value using getKeyBuffer() and getValueBuffer()
- turn each into a database entry
- use the `putNoOverwrite()` method to atomically input data and throw errors if it already exists

---

XML Review

- in semi-structure data there is no schema, its not too rigid
  - self-documenting
  - You can ignore bulk of information using XML and only focus on the useful portions

XML Elements

- An XML *element* is:
  - a begin tag
  - an end tag (in some cases, this is merged into the begin tag)
  - all info. between them.
  - example:

```
<name>CS 460</name>
```

- An element can include other nested *child elements.*

```
<course>
    <name>CS 460</name>
    <begin>1:25</begin>
    …
</course>
```

- Related XML elements are grouped together into *documents*.
  - may or may not be stored as an actual text document

- XML is a **markup language and extensible**

Semistructured Data (cont.)

- Its features facilitate:
  - the integration of information from different sources
  - the exchange of information between applications
- Example: company A receives data from company B
  - A only cares about certain fields in certain types of records
  - B's data includes:
    - other types of records
    - other fields within the records that company A cares about
  - with semistructured data, A can easily recognize and ignore unexpected elements
  - the exchange is more complicated with structured data

- element is everything between the begin and end tags, elements inside are children. Many elements form a document
- 2 ways to capture relationships

Capturing Relationships in XML

- Two options:

1. store references from one element to other elements using `ID`, `IDREF` and `IDREFS` attributes:

```
<course cid="C20119" teacher="P123456">
  <cname>CS 111</cname>
  …
</course>

<course cid="C20268" teacher="P123456">
  <cname>CS 460</cname>
  …
</course>

<person pid="P123456" teaches="C20119 C20268">
  <pname>
     <last>Sullivan</last>
     <first>David</first>
  </pname>
</person>
```

   - where have we seen something similar?

Capturing Relationships in XML (cont.)

2. use child elements:

```
<course cid="C20119">
  <cname>CS 111</cname>
  <teacher id="P123456">David Sullivan</teacher>
</course>

 …
<person pid="P123456">
  <pname>
     <last>Sullivan</last>
     <first>David</first>
  </pname>
  <courses-taught>
     <course-taught>CS 111</course-taught>
     <course-taught>CS 460</course-taught>
  </courses-taught>
</person>
```

- There are pluses and minuses to each approach.
  - we'll revisit this design issue later in the course

XPath Expressions (cont.)

- Attribute names are preceded by an `@` symbol:
  - example: `//person/@pid`
    - selects all `pid` attributes of all `person` elements
- We can specify a particular document as follows:

  `document("`*doc-name*`")`*path-expression*

  - example:

    `document("university.xml")//course/start`

Predicates in XPath Expressions (cont.)

```
<room>
   <building>CAS</building><room_num>212</room_num>
</room>
<room>
   <building>CAS</building><room_num>100</room_num>
</room>
<room>
   <building>KCB</building><room_num>101</room_num>
</room>
<room>
   <building>PSY</building><room_num>228D</room_num>
</room>
```

- Use `.` to represent nodes selected by the preceding path.

  `//room/room_num[. < 200]`
  - selects all `room_num` elements with values < 200

  `//room[room_num < 200]`
  - selects all `room` elements with `room_num` child values < 200

- **let select everything, while for is for each element. In count operations, we use `let`**

FLWOR Expressions

```
for $r in //room[contains(name, "CAS")],
    $c in //course
let $e := //person[contains(@enrolled, $c/@id)]
where $c/@room = $r/@id and count($e) > 20
order by $r/name
return ($r/name, $c/name)
```

- The `for` clause is like the `FROM` clause in SQL.
  - the query iterates over all combinations of values from its XPath expressions (like Cartesian product!)
  - query above looks at combos of CAS `rooms` and `courses`
- The `let` clause is applied to each combo. from the `for` clause.
  - each variable gets the *full set* produced by its XPath expr.
  - unlike a `for` clause, which assigns the results of the XPath expression one value at a time

Reshaping the Output (cont.)

```
<course id="C20119" teacher="P123456" room="011">
   <name>CS 111</name><start>10:10</start><end>11:00</end>
</course>
<course id="C20268" teacher="P123456">
   <name>CS 460</name><start>13:25</start><end>14:15</end>
</course>
<course id="C20757" teacher="P778787" room="789">
   <name>CS 112</name><start>11:30</start><end>12:45</end>
</course>
```

```
for $c in //course
where $c/start > "11:00"
return <after11-course>
       { string($c/name), " - ", string($c/start) }
       </after11-course>
```

- The result will look something like this:

```
<after11-course>CS 460 - 13:25</after11-course>
<after11-course>CS 112 - 11:30</after11-course>
```

TopHat

- // - finding all instances of the begin tag
- / - starting from the root

Consider the XML elements shown below, which are examples of elements that could be used to store info about courses in our university database. You should assume that:
- The database includes other course elements that we have not shown below.
- All of the course elements are children of the root element, which is of type university-data.

Which of the following XPath expressions could be used to obtain the full `course` elements for all courses that end at 11:00 (i.e., that have an `end` child element whose value is "11:00")?

expression 1: `/university-data/course[end="11:00"]`
expression 2: `//course[end="11:00"]`
expression 3: `/course[end="11:00"]`

```
<course cid="C20119" teacher="P123456" room="CAS 522">
  <name>CS 111</name><start>10:10</start><end>11:00</end>
</course>
<course cid="C20268" teacher="P123456">
  <name>CS 460</name><start>1:25</start><end>2:15</end>
</course>
<course cid="C20757" teacher="P778787" room="COM 101">
  <name>CS 112</name><start>11:30</start><end>12:45</end>
</course>
```

Select an answer and submit. For keyboard navigation, use the up/down arrow keys to select an answer.

a	only expression 1
b	only expression 2
c	only expression 3
d	expressions 1 and 2, but not 3 ✓ Your answer
e	expressions 2 and 3, but not 1

Show Submitted Answer | Show Correct Answer | Check My Answer

Question 3
Review

Given the XML elements shown below and the assumptions that we made in Question 1, which of the following XPath expressions could be used to obtain the `start` child elements for all courses that end at 11:00?

expression 1: `//course[end="11:00"]/start`
expression 2: `//course/start[end="11:00"]`
expression 3: `/course/start[../end="11:00"]`

```
<course cid="C20119" teacher="P123456" room="CAS 522">
   <name>CS 111</name><start>10:10</start><end>11:00</end>
</course>
<course cid="C20268" teacher="P123456">
   <name>CS 460</name><start>1:25</start><end>2:15</end>
</course>
<course cid="C20757" teacher="P778787" room="COM 101">
   <name>CS 112</name><start>11:30</start><end>12:45</end>
</course>
```

Select an answer and submit. For keyboard navigation, use the up/down arrow keys to select an answer.

a	only expression 1
b	only expression 2
c	only expression 3
d	both expressions 1 and 2, but not 3
e	both expressions 1 and 3, but not 2 ✓ Your answer
f	both expressions 2 and 3, but not 1

Show Submitted Answer | Show Correct Answer | Check My Answer

Question 2
Review

Consider the XML elements shown below, which are examples of elements that could be used to store info about courses and rooms in our university database.

When the following partial FLWOR expression is executed, what will be assigned to the variables `$r` and `$c` at a given point in time?

```
for $r in //room
let $c := //course[@room=$r/@id]
...
```

```
<course cid="C20119" teacher="P123456" room="R01157">
   <name>CS 111</name><start>10:10</start><end>11:00</end>
</course>

<room rid="R01157">
   <name>CAS 522</name>
   <capacity>200</capacity>
</room>
```

Select an answer and submit. For keyboard navigation, use the up/down arrow keys to select an answer.

a	`$r` will be assigned one `room` element at a time, but `$c` will be assigned a set of `course` elements	✓ Your answer

Assume that:
- The `room` attribute in `course` elements is an `IDREF` attribute that holds the value of the `rid` attribute from a `room` element elsewhere in the database. This allows us to capture the relationships between courses and the rooms in which they meet.
- Computer science course are the only ones that have the string "CS" as part of their name.

When the following FLWOR expression is executed, what will it do?

```
for $r in //room
let $c := //course[contains(name, "CS") and @room=$r/@id]
return <room_info> {
 <number>{ count($c) }</number>,
 for $c2 in $c
 return $c2/name
} </room_info>
```

```
<course cid="C20119" teacher="P123456" room="R01157">
   <name>CS 111</name><start>10:10</start><end>11:00</end>
</course>

<room rid="R01157">
   <name>CAS 522</name>
   <capacity>200</capacity>
</room>
```

Select an answer and submit. For keyboard navigation, use the up/down arrow keys to select an answer.

a	for each room in the database, output a new element of type `room_info` that includes child elements for the number of CS courses that meet in that room, and the names (if any) of each of those courses ✓ Your answer
b	for each room in which a CS course meets, output a new element of type `room_info` that includes child elements for the number of CS courses that meet in that room, and the names of each of those courses
c	for each CS course in the database, output a new element of type `room_info` that includes child elements for the number of rooms in which that course meets and the names of those rooms
d	output a single new element of type `room_info` that includes child elements for the total number of rooms in which CS courses meet and the names of those rooms

a (if any) distinction important

b is false because we not looking for rooms with cs course in the for, thats the let cause

c is talking about courses even though we started with rooms

d doesn't acknowledge the for

Transactions

Question 2
Review

Consider again the same schedule as in the previous question
$w_1(A); r_2(B); r_2(A); w_1(B); r_3(D); r_1(D)$

Which of the following constraints are imposed by the conflicting actions in the schedule?
1: T1 --> T2 (i.e., T1 must come before T2 in any equivalent schedule)
2: T2 --> T1
3: T3 --> T1

Select an answer and submit. For keyboard navigation, use the up/down arrow keys to select an answer.

a	only constraint 1
b	only constraint 2
c	only constraint 3
d	constraints 1 and 2, but not 3 ✓ Your answer
e	constraints 1 and 3, but not 2
f	constraints 2 and 3, but not 1

- in waits-for-graph, reverse arrows!
- PL guarantees serializability, strict 2PL allows all

Strict Locking

- *Strict locking* makes txns hold all *exclusive* locks until they commit or abort.
  - doing so prevents dirty reads, which means schedules will be recoverable and cascadeless

T_1	T_2
	xl(A); w(A) sl(C) *u(A)*
xl(A); r(A)	r(C); u(C)
w(A); *u(A)* commit	commit

T_1	T_2
	xl(A); w(A) sl(C)
xl(A); wait	r(C); u(C)
	commit *u(A)*
xl(A); r(A)	
w(A) commit *u(A)*	

T1 can't acquire the lock for A until after T2 commits.
Thus, its read of A is not dirty!

- strict + 2PL = strict 2PL

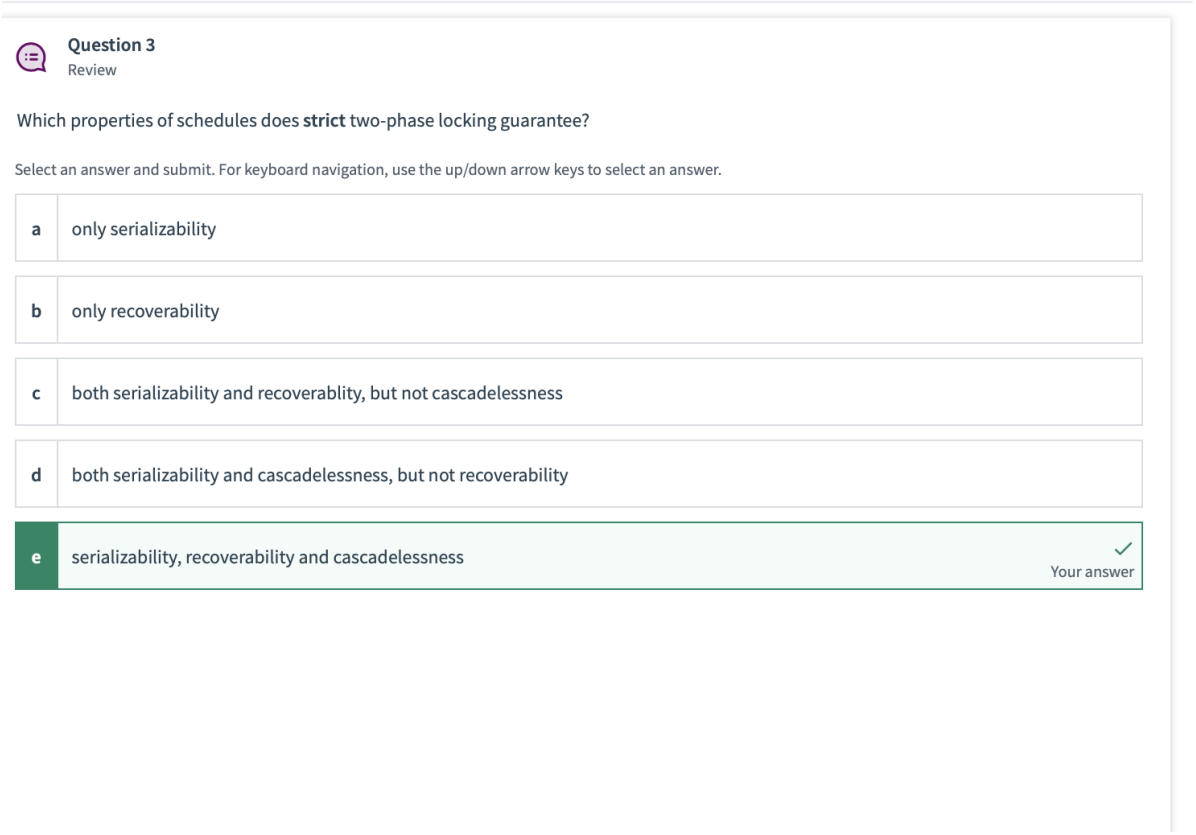
Rigorous Locking

- Under strict locking, it's possible to get something like this:

T_1	T_2	T_3
… sl(A); r(A) u(A)		
…	xl(A); w(A) commit u(A)	
		sl(A); r(A) commit u(A) print A
commit print A		

  - T3 reports A's new value.
  - T1 reports A's old value, even though it commits after T3.
  - the ordering of commits (T2,T3,T1) is not same as the equivalent serial ordering (T1,T2,T3)

- *Rigorous locking* requires txns to hold *all* locks until commit/abort.
- It guarantees that transactions commit in the same order as they would in the equivalent serial schedule.
- rigorous + 2PL = rigorous 2PL



- get update lock AFTER shared lock but not the reverse

Question 1
Review

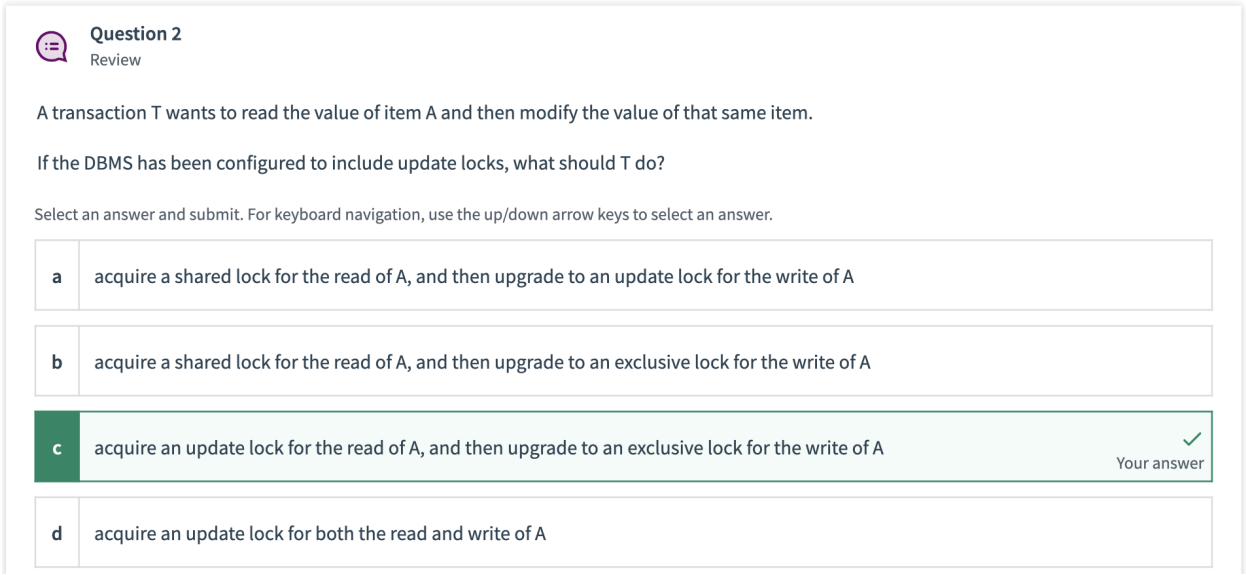
Imagine that a transaction T1 holds:
- an exclusive lock for item A
- a shared lock for item B
- an update lock for item C

Which of the following actions by a second transaction T2 will cause it to wait?
action 1: requesting an update lock for item A
action 2: requesting an update lock for item B
action 3: requesting an update lock for item C

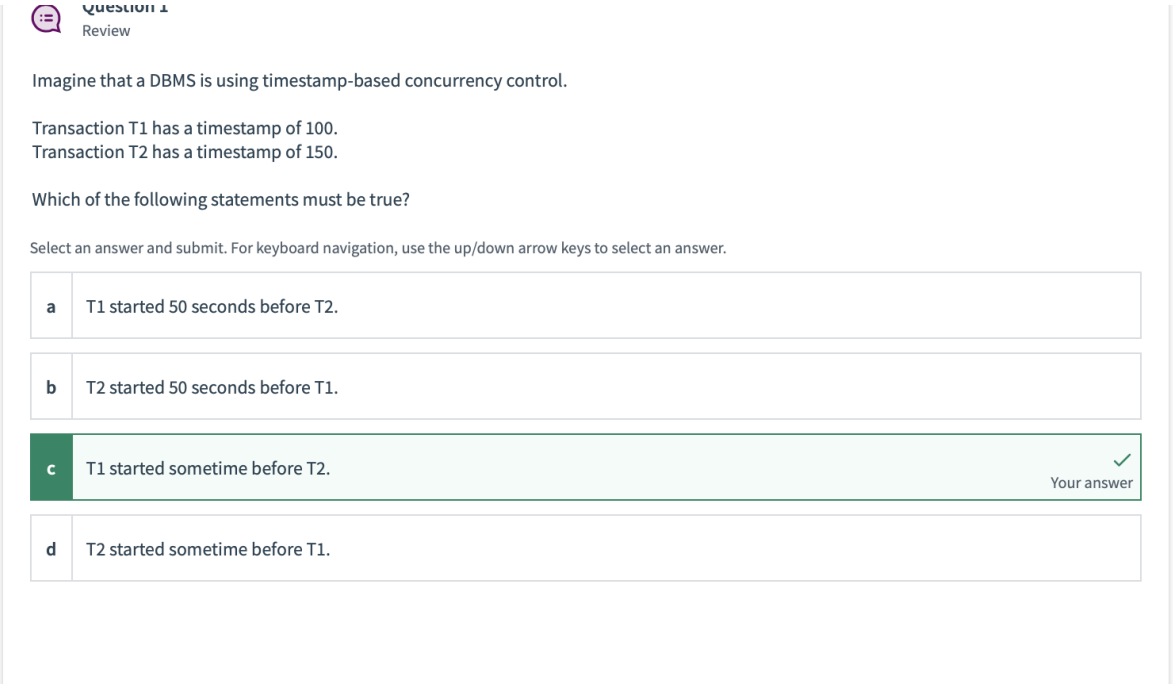
Select an answer and submit. For keyboard navigation, use the up/down arrow keys to select an answer.

a	only action 1
b	only action 2
c	only action 3
d	actions 1 and 2, but not 3
e	actions 1 and 3, but not 2 ✓ Your answer
f	actions 2 and 3, but not 1
g	all three actions

Show Submitted Answer | Show Correct Answer | Check My Answer



- You dont know the exact start time, thats when the first transaction occurred



Question 3
Review

Imagine that a DBMS is using regular timestamp-based concurrency control.

Transaction T1 has a timestamp of 100.
Transaction T2 has a timestamp of 150.

T2 is allowed to write item A.
If T1 then tries to write item A, what should the system do?

(**Note:** You should be able to answer this question without knowing the read and write timestamps of A. Rather, you should consider how the DBMS should respond to T1's write request in light of T2's write of A and the equivalent serial schedule that stems from the timestamps of the two transactions.)

Select an answer and submit. For keyboard navigation, use the up/down arrow keys to select an answer.

a	deny the write request and roll back T1
b	allow T1's write to occur
c	ignore T1's write request and let T1 continue ✓ Your answer

- wait because of commit bits

Imagine that a DBMS is using timestamp-based concurrency control **with commit bits**.

Transaction T1 has a timestamp of 100.
Transaction T2 has a timestamp of 150.

T1 is allowed to write item A.
T1 has more work to do, but T2 is scheduled next and tries to read item A.
What should the DBMS do?

(**Note:** You should be able to answer this question without knowing the read and write timestamps of A. Rather, you should consider how the DBMS should respond to T2's read request in light of T1's write of A, the equivalent serial schedule that stems from the timestamps of the two transactions, and the presence of commit bits.)

Select an answer and submit. For keyboard navigation, use the up/down arrow keys to select an answer.

a	deny the read request and roll back T2
b	deny the read request and make T2 wait ✓ Your answer
c	ignore T2's read request and let T2 continue
d	allow T2's read to occur

Correct Answer:

✓ **b** - deny the read request and make T2 wait

- we can write for a valid read.

Question 2
Review

Imagine that a DBMS is using timestamp-based concurrency control with commit bits.

Transaction T1 has a timestamp of 100.
Transaction T2 has a timestamp of 150.

Before either transaction started, the state associated with data item A is:
RTS(A) = 0; WTS(A) = 0; c = true

After T1 and T2 have performed some operations but before either of them has committed, which of the following could be the state of item A?
option 1: RTS(A) = 0; WTS(A) = 150; c = True
option 2: RTS(A) = 100; WTS(A) = 150; c = False
option 3: RTS(A) = 150; WTS(A) = 100; c = False

Select an answer and submit. For keyboard navigation, use the up/down arrow keys to select an answer.

a	only option 1
b	only option 2
c	only option 3
d	option 1 or 2, but not 3
e	option 1 or 3, but not 2
f	option 2 or 3, but not 1

Correct Answer:

✓ **b** - only option 2