Midterm 2 Review

- Midterm: **Friday, November 14**
- The exam will cover everything from **semi-structured data and XML through the material on timestamp-based concurrency control (pg. 151-229)**
- **Resources**
  - Homework
  - Lab
  - Coursepack
  - Lecture Recordings
  - Pre-lecture Videos
  - **Midterm Review Problems**

Topics

- BerkeleyDB (Higher level concepts, no fine-grained details)
- Semistructured Data & XML
  - Semistructured Data v.s. Structured Data
  - XML Elements, Attributes, Well-formed XML, Relationships in XML
  - XPath expressions
  - **FLWOR expressions**
- Transactions & Schedules
  - Serializability, Conflict Serializability (*Precedence Graph*), Recoverable, Cascadeless
- Concurrency Control - **Locking**
  - 2PL
  - Shared,Exclusive, Update Locks
  - Deadlock Detection (*Waits-for Graph*)
- Concurrency Control - **Timestamps**
  - Commit bits
  - Multiversion

*Disclaimer: Might not be entirely comprehensive

Well-formed XML

Sample Bank XML Database in XML

```xml
<bank>
    <account account_num="A0000001" owners="C0000001 C0000002">
        <branch>Cambridge, MA</branch>
        <balance>20000</balance>
    </account>
    <account account_num="A0000002" owners="C0000003">
        <branch>Burlington, MA</branch>
        <balance>7000</balance>
    </account>
    <account account_num="A0000003" owners="C0000003 C0000004">
        <branch>Burlington, MA</branch>
        <balance>300</balance>
    </account>
    <account account_num="A0000004" owners="C0000004">
        <branch>Burlington, MA</branch>
        <balance>1000</balance>
    </account>
    <account account_num="A0000005" owners="C0000002">
        <branch>Cambridge, MA</branch>
        <balance>1500</balance>
    </account>
    <customer customer_num="C0000001" owns="A0000001">
        <name>John Smith</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000002" owns="A0000001 A0000005">
        <name>Alice Wong</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000003" owns="A0000002 A0000003">
        <name>Jose Delgado</name>
        <address>Zero Longhorn Ave., Belmont, MA</address>
    </customer>
    <customer customer_num="C0000004" owns="A0000003 A0000004">
        <name>Tammy Terrier</name>
        <address>665 Commonwealth Ave., Boston, MA</address>
    </customer>
</bank>
```

Sample Bank XML Database in XML

```
<bank>
    <account account_num="A0000001" owners="C0000001 C0000002">
        <branch>Cambridge, MA</branch>
        <balance>20000</balance>
    </account>
    <account account_num="A0000002" owners="C0000003">
        <branch>Burlington, MA</branch>
        <balance>7000</balance>
    </account>
    <account account_num="A0000003" owners="C0000003 C0000004">
        <branch>Burlington, MA</branch>
        <balance>300</balance>
    </account>
    <account account_num="A0000004" owners="C0000004">
        <branch>Burlington, MA</branch>
        <balance>1000</balance>
    </account>
    <account account_num="A0000005" owners="C0000002">
        <branch>Cambridge, MA</branch>
        <balance>1500</balance>
    </account>
    <customer customer_num="C0000001" owns="A0000001">
        <name>John Smith</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000002" owns="A0000001 A0000005">
        <name>Alice Wong</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000003" owns="A0000002 A0000003">
        <name>Jose Delgado</name>
        <address>Zero Longhorn Ave., Belmont, MA</address>
    </customer>
    <customer customer_num="C0000004" owns="A0000003 A0000004">
        <name>Tammy Terrier</name>
        <address>665 Commonwealth Ave., Boston, MA</address>
    </customer>
</bank>
```

In a well-formed XML document, there is a single root element (<bank></bank>) that contains all other elements

ID, IDREF, IDREFS

Sample Bank XML Database in XML

```xml
<bank>
    <account account_num="A0000001" owners="C0000001 C0000002">
        <branch>Cambridge, MA</branch>
        <balance>20000</balance>
    </account>
    <account account_num="A0000002" owners="C0000003">
        <branch>Burlington, MA</branch>
        <balance>7000</balance>
    </account>
    <account account_num="A0000003" owners="C0000003 C0000004">
        <branch>Burlington, MA</branch>
        <balance>300</balance>
    </account>
    <account account_num="A0000004" owners="C0000004">
        <branch>Burlington, MA</branch>
        <balance>1000</balance>
    </account>
    <account account_num="A0000005" owners="C0000002">
        <branch>Cambridge, MA</branch>
        <balance>1500</balance>
    </account>
    <customer customer_num="C0000001" owns="A0000001">
        <name>John Smith</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000002" owns="A0000001 A0000005">
        <name>Alice Wong</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000003" owns="A0000002 A0000003">
        <name>Jose Delgado</name>
        <address>Zero Longhorn Ave., Belmont, MA</address>
    </customer>
    <customer customer_num="C0000004" owns="A0000003 A0000004">
        <name>Tammy Terrier</name>
        <address>665 Commonwealth Ave., Boston, MA</address>
    </customer>
</bank>
```

account_num is an example of an **ID** attribute — an identifier that must be unique within the document

Special Types of Attributes

- `ID` an identifier that must be unique within the document (among *all* `ID` attributes – not just this attribute)
- `IDREF` a single value that is the value of an `ID` attribute elsewhere in the document
- `IDREFS` a *list* of `ID` values from elsewhere in the document

Sample Bank XML Database in XML

```xml
<bank>
    <account account_num="A0000001" owners="C0000001 C0000002">
        <branch>Cambridge, MA</branch>
        <balance>20000</balance>
    </account>
    <account account_num="A0000002" owners="C0000003">
        <branch>Burlington, MA</branch>
        <balance>7000</balance>
    </account>
    <account account_num="A0000003" owners="C0000003 C0000004">
        <branch>Burlington, MA</branch>
        <balance>300</balance>
    </account>
    <account account_num="A0000004" owners="C0000004">
        <branch>Burlington, MA</branch>
        <balance>1000</balance>
    </account>
    <account account_num="A0000005" owners="C0000002">
        <branch>Cambridge, MA</branch>
        <balance>1500</balance>
    </account>
    <customer customer_num="C0000001" owns="A0000001">
        <name>John Smith</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000002" owns="A0000001 A0000005">
        <name>Alice Wong</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000003" owns="A0000002 A0000003">
        <name>Jose Delgado</name>
        <address>Zero Longhorn Ave., Belmont, MA</address>
    </customer>
    <customer customer_num="C0000004" owns="A0000003 A0000004">
        <name>Tammy Terrier</name>
        <address>665 Commonwealth Ave., Boston, MA</address>
    </customer>
</bank>
```

owners is an example of an **IDREFS** attribute — a list of ID values from elsewhere in the document

Remember when we didn't allow multi-valued attributes in relationship databases? We can have them in XML!

owners capture relationship between <account> and <customer>: each account can be owned by multiple customers

```xml
<bank>
    <account account_num="A0000001" owners="C0000001 C0000002">
        <branch>Cambridge, MA</branch>
        <balance>20000</balance>
    </account>
    <account account_num="A0000002" owners="C0000003">
        <branch>Burlington, MA</branch>
        <balance>7000</balance>
    </account>
    <account account_num="A0000003" owners="C0000003 C0000004">
        <branch>Burlington, MA</branch>
        <balance>300</balance>
    </account>
    <account account_num="A0000004" owners="C0000004">
        <branch>Burlington, MA</branch>
        <balance>1000</balance>
    </account>
    <account account_num="A0000005" owners="C0000002">
        <branch>Cambridge, MA</branch>
        <balance>1500</balance>
    </account>
    <customer customer_num="C0000001" owns="A0000001">
        <name>John Smith</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000002" owns="A0000001 A0000005">
        <name>Alice Wong</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000003" owns="A0000002 A0000003">
        <name>Jose Delgado</name>
        <address>Zero Longhorn Ave., Belmont, MA</address>
    </customer>
    <customer customer_num="C0000004" owns="A0000003 A0000004">
        <name>Tammy Terrier</name>
        <address>665 Commonwealth Ave., Boston, MA</address>
    </customer>
</bank>
```

Sample Bank XML Database in XML

owns is also **IDREFS**

owns also capture relationship between <account> and <customer>:

each customer can own multiple accounts

*But if we change it such that each owns can have **only one** value, then owns would be an **IDREF** attribute

XPath

Summary of XPath Expressions

/ (Begins at root)

// (Select elements anywhere in the document)

@ (When selecting an attribute)

/course[room_num < 200] (filtering by conditions, room_num must be a child of course)

[**contains**(longer_string, shorter_string)] (whenever you're dealing with IDREFS!)

/room_num[**.** < 200] (filter it's own with value less than 200)

//room_num[**../**building="CAS"] (if building element is a **sibling** of room_num element!)

```
<bank>
    <account account_num="A0000001" owners="C0000001 C0000002">
        <branch>Cambridge, MA</branch>
        <balance>20000</balance>
    </account>
    <account account_num="A0000002" owners="C0000003">
        <branch>Burlington, MA</branch>
        <balance>7000</balance>
    </account>
    <account account_num="A0000003" owners="C0000003 C0000004">
        <branch>Burlington, MA</branch>
        <balance>300</balance>
    </account>
    <account account_num="A0000004" owners="C0000004">
        <branch>Burlington, MA</branch>
        <balance>1000</balance>
    </account>
    <account account_num="A0000005" owners="C0000002">
        <branch>Cambridge, MA</branch>
        <balance>1500</balance>
    </account>
    <customer customer_num="C0000001" owns="A0000001">
        <name>John Smith</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000002" owns="A0000001 A0000005">
        <name>Alice Wong</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000003" owns="A0000002 A0000003">
        <name>Jose Delgado</name>
        <address>Zero Longhorn Ave., Belmont, MA</address>
    </customer>
    <customer customer_num="C0000004" owns="A0000003 A0000004">
        <name>Tammy Terrier</name>
        <address>665 Commonwealth Ave., Boston, MA</address>
    </customer>
</bank>
```

Write an XPath expression to obtain the **account numbers** of all accounts with a balance that is greater than 400.

Not this one, the balance is less than 400

```xml
<bank>
    <account account_num="A0000001" owners="C0000001 C0000002">
        <branch>Cambridge, MA</branch>
        <balance>20000</balance>
    </account>
    <account account_num="A0000002" owners="C0000003">
        <branch>Burlington, MA</branch>
        <balance>7000</balance>
    </account>
    <account account_num="A0000003" owners="C0000003 C0000004">
        <branch>Burlington, MA</branch>
        <balance>300</balance>
    </account>
    <account account_num="A0000004" owners="C0000004">
        <branch>Burlington, MA</branch>
        <balance>1000</balance>
    </account>
    <account account_num="A0000005" owners="C0000002">
        <branch>Cambridge, MA</branch>
        <balance>1500</balance>
    </account>
    <customer customer_num="C0000001" owns="A0000001">
        <name>John Smith</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000002" owns="A0000001 A0000005">
        <name>Alice Wong</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000003" owns="A0000002 A0000003">
        <name>Jose Delgado</name>
        <address>Zero Longhorn Ave., Belmont, MA</address>
    </customer>
    <customer customer_num="C0000004" owns="A0000003 A0000004">
        <name>Tammy Terrier</name>
        <address>665 Commonwealth Ave., Boston, MA</address>
    </customer>
</bank>
```

Write an XPath expression to obtain the **account numbers** of all accounts with a balance that is greater than 400.

/bank/account[balance > 400]/@account_num

```xml
<bank>
    <account account_num="A0000001" owners="C0000001 C0000002">
        <branch>Cambridge, MA</branch>
        <balance>20000</balance>
    </account>
    <account account_num="A0000002" owners="C0000003">
        <branch>Burlington, MA</branch>
        <balance>7000</balance>
    </account>
    <account account_num="A0000003" owners="C0000003 C0000004">
        <branch>Burlington, MA</branch>
        <balance>300</balance>
    </account>
    <account account_num="A0000004" owners="C0000004">
        <branch>Burlington, MA</branch>
        <balance>1000</balance>
    </account>
    <account account_num="A0000005" owners="C0000002">
        <branch>Cambridge, MA</branch>
        <balance>1500</balance>
    </account>
    <customer customer_num="C0000001" owns="A0000001">
        <name>John Smith</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000002" owns="A0000001 A0000005">
        <name>Alice Wong</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000003" owns="A0000002 A0000003">
        <name>Jose Delgado</name>
        <address>Zero Longhorn Ave., Belmont, MA</address>
    </customer>
    <customer customer_num="C0000004" owns="A0000003 A0000004">
        <name>Tammy Terrier</name>
        <address>665 Commonwealth Ave., Boston, MA</address>
    </customer>
</bank>
```

Write an XPath expression to obtain the **account numbers** of all accounts with a balance that is greater than 400.

/bank/account[balance > 400]/@account_num

OR

//account[balance > 400]/@account_num

```xml
<bank>
    <account account_num="A0000001" owners="C0000001 C0000002">
        <branch>Cambridge, MA</branch>
        <balance>20000</balance>
    </account>
    <account account_num="A0000002" owners="C0000003">
        <branch>Burlington, MA</branch>
        <balance>7000</balance>
    </account>
    <account account_num="A0000003" owners="C0000003 C0000004">
        <branch>Burlington, MA</branch>
        <balance>300</balance>
    </account>
    <account account_num="A0000004" owners="C0000004">
        <branch>Burlington, MA</branch>
        <balance>1000</balance>
    </account>
    <account account_num="A0000005" owners="C0000002">
        <branch>Cambridge, MA</branch>
        <balance>1500</balance>
    </account>
    <customer customer_num="C0000001" owns="A0000001">
        <name>John Smith</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000002" owns="A0000001 A0000005">
        <name>Alice Wong</name>
        <address>148 Pearl St., Watertown, MA</address>
    </customer>
    <customer customer_num="C0000003" owns="A0000002 A0000003">
        <name>Jose Delgado</name>
        <address>Zero Longhorn Ave., Belmont, MA</address>
    </customer>
    <customer customer_num="C0000004" owns="A0000003 A0000004">
        <name>Tammy Terrier</name>
        <address>665 Commonwealth Ave., Boston, MA</address>
    </customer>
</bank>
```

Write an XPath expression to obtain the **account numbers** of all accounts with a balance that is greater than 400.

/bank/account[balance > 400]/@account_num

OR

//account[balance > 400]/@account_num

OR

//account/@account_num[../balance > 400]

Midterm 2 Review Problems **Q3** (Try it yourself!)

```
<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>
```

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!)

Midterm 2 Review Problems **Q3** (Try it yourself!)

```
<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>
```

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!)

//movie[rating = “R” and contains(year, “199”)]/name

Midterm 2 Review Problems **Q3** (Try it yourself!)

```
<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>
```

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!)

//movie[rating = “R” and contains(year, “199”)]/name

//movie[rating = "R" and year >= 1990 and year <= 1999]/name

//movie/name[../rating = "R" and contains(../year, "199")]

//movie/name[../rating = "R" and ../year >= 1990 and ../year <= 1999]

XQuery - FLWOR

Write an XQuery FLWOR expression to **create new elements** of type customer that include:

We also want to **exclude** customers **with 0 accounts**

```
<customer>
  <name>Jose Delgado</name>
  <account>Burlington, MA - 7000</account>
  <account>Burlington, MA - 300</account>
</customer>
```

example customer and account elements:

```
<customer customer_num="C0000002" owns="A0000001 A0000005">
    <name>Alice Wong</name>
    <address>148 Pearl St., Watertown, MA</address>
</customer>
<account account_num="A0000001" owners="C0000001 C0000002">
    <branch>Cambridge, MA</branch>
    <balance>20000</balance>
</account>
```

Write an XQuery FLWOR expression to **create new elements** of type customer that include:

We also want to **exclude** customers **with 0 accounts**

```
<customer>
  <name>Jose Delgado</name>
  <account>Burlington, MA - 7000</account>
  <account>Burlington, MA - 300</account>
</customer>
```

example customer and account elements:

```
<customer customer_num="C0000002" owns="A0000001 A0000005">
    <name>Alice Wong</name>
    <address>148 Pearl St., Watertown, MA</address>
</customer>
<account account_num="A0000001" owners="C0000001 C0000002">
    <branch>Cambridge, MA</branch>
    <balance>20000</balance>
</account>
```

For each customer (e.g. Jose Delgado) + Get **all** the accounts associated with that customer = We're going to use a let clause!

Write an XQuery FLWOR expression to **create new elements** of type customer that include:

We also want to **exclude** customers **with 0 accounts**

```
<customer>
  <name>Jose Delgado</name>
  <account>Burlington, MA - 7000</account>
  <account>Burlington, MA - 300</account>
</customer>
```

example customer and account elements:

```
<customer customer_num="C0000002" owns="A0000001 A0000005">
    <name>Alice Wong</name>
    <address>148 Pearl St., Watertown, MA</address>
</customer>
<account account_num="A0000001" owners="C0000001 C0000002">
    <branch>Cambridge, MA</branch>
    <balance>20000</balance>
</account>
```

For each customer (e.g. Jose Delgado)

for $c in //customer

Get **all** the accounts associated with that customer

We're going to use a let clause!

Write an XQuery FLWOR expression to **create new elements** of type customer that include:

We also want to **exclude** customers **with 0 accounts**

```
<customer>
  <name>Jose Delgado</name>
  <account>Burlington, MA - 7000</account>
  <account>Burlington, MA - 300</account>
</customer>
```

example customer and account elements:

```
<customer customer_num="C0000002" owns="A0000001 A0000005">
    <name>Alice Wong</name>
    <address>148 Pearl St., Watertown, MA</address>
</customer>
<account account_num="A0000001" owners="C0000001 C0000002">
    <branch>Cambridge, MA</branch>
    <balance>20000</balance>
</account>
```

For each customer (e.g. Jose Delgado)

Get **all** the accounts associated with that customer

We're going to use a let clause!

```
for $c in //customer
let $c_accounts := //account[contains(@owners, $c/@customer_num)]
```

Write an XQuery FLWOR expression to **create new elements** of type customer that include:

We also want to **exclude** customers **with 0 accounts**

```
<customer>
  <name>Jose Delgado</name>
  <account>Burlington, MA - 7000</account>
  <account>Burlington, MA - 300</account>
</customer>
```

example customer and account elements:

```
<customer customer_num="C0000002" owns="A0000001 A0000005">
    <name>Alice Wong</name>
    <address>148 Pearl St., Watertown, MA</address>
</customer>
<account account_num="A0000001" owners="C0000001 C0000002">
    <branch>Cambridge, MA</branch>
    <balance>20000</balance>
</account>
```

For each customer (e.g. Jose Delgado)

Get **all** the accounts associated with that customer

We're going to use a let clause!

```
for $c in //customer
let $c_accounts := //account[contains(@owners, $c/@customer_num)]
```

the "join condition" must go in a **contains** since @owners is **IDREFS**

Write an XQuery FLWOR expression to **create new elements** of type customer that include:

We also want to **exclude** customers **with 0 accounts**

```
<customer>
  <name>Jose Delgado</name>
  <account>Burlington, MA - 7000</account>
  <account>Burlington, MA - 300</account>
</customer>
```

example customer and account elements:

```
<customer customer_num="C0000002" owns="A0000001 A0000005">
    <name>Alice Wong</name>
    <address>148 Pearl St., Watertown, MA</address>
</customer>
<account account_num="A0000001" owners="C0000001 C0000002">
    <branch>Cambridge, MA</branch>
    <balance>20000</balance>
</account>
```

For each customer (e.g. Jose Delgado) + Get **all** the accounts associated with that customer = We're going to use a let clause!

```
for $c in //customer
let $c_accounts := //account[contains(@owners, $c/@customer_num)]
```

customer element + [account element 1, account element 2]

Write an XQuery FLWOR expression to **create new elements** of type customer that include:

We also want to **exclude** customers **with 0 accounts**

```
<customer>
  <name>Jose Delgado</name>
  <account>Burlington, MA - 7000</account>
  <account>Burlington, MA - 300</account>
</customer>
```

example customer and account elements:

```
<customer customer_num="C0000002" owns="A0000001 A0000005">
    <name>Alice Wong</name>
    <address>148 Pearl St., Watertown, MA</address>
</customer>
<account account_num="A0000001" owners="C0000001 C0000002">
    <branch>Cambridge, MA</branch>
    <balance>20000</balance>
```

```
for $c in //customer
let $c_accounts := //account[contains(@owners, $c/@customer_num)]
where count($c_accounts) > 0
```

Write an XQuery FLWOR expression to **create new elements** of type customer that include:

We also want to **exclude** customers **with 0 accounts**

```
<customer>
  <name>Jose Delgado</name>
  <account>Burlington, MA - 7000</account>
  <account>Burlington, MA - 300</account>
</customer>
```

example customer and account elements:

```
<customer customer_num="C0000002" owns="A0000001 A0000005">
    <name>Alice Wong</name>
    <address>148 Pearl St., Watertown, MA</address>
</customer>
<account account_num="A0000001" owners="C0000001 C0000002">
    <branch>Cambridge, MA</branch>
    <balance>20000</balance>
</account>
```

```
for $c in //customer
let $c_accounts := //account[contains(@owners, $c/@customer_num)]
where count($c_accounts) > 0
return <customer>
    {
      $c/name,
      for $a in $c_accounts
      return <account>
          { string($a/branch), "-", string($a/balance) }
          </account>
    }
    </customer>
```

Use a subquery to **unfold the list** obtained in the let clause!

use string() whenever you want the literal value (no start and end tags)

Midterm 2 Review Problems **Q8** (Try it yourself!)

```
<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>
```

```
<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>
```

Write FLWOR query that produces, **for each** director born in New York state, a new element of type ny_director that has 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.

Example output element:

```
<ny_director>
     <name>James Cameron>
     <birthplace>New York City, New York, USA</birthplace>
     <directed>Titanic</directed>
     <directed>Avatar</directed>
     <directed>The LXH Movie</directed>
</ny_director>
```

Midterm 2 Review Problems **Q8** (Try it yourself!)

```
<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>
```

```
<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>
```

Write FLWOR query that produces, **for each** director born in New York state, a new element of type ny_director that has 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.

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

Midterm 2 Review Problems **Q8** (Try it yourself!)

```
<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>
```

```
<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>
```

Write FLWOR query that produces, **for each** director born in New York state, a new element of type ny_director that has 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.

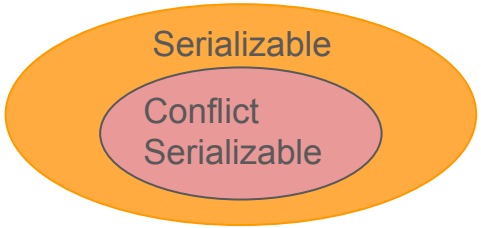
```
for $p in //person[@directed]
where contains($p/pob, "New York, USA")
return <ny_director>{
        $p/name,
        <birthplace>{ string($p/pob) }</birthplace>,
        for $m in //movie
        where contains($p/@directed, $m/@id)
        return <directed>{ string($m/name) }</directed>
}</ny_director>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

Practice FLWOR more from previous lab!

Serializable, Conflict Serializable
Recoverable, Cascadeless

Summary

- Serializable
  - If the interleaving of transactions has equivalent effect as some serial schedule
  - If it's conflict serializable, it's definitely also serializable
- Conflict Serializable
  - If there are no cycles in the precedence graph = serializable
- Recoverable
  - If no dirty read = recoverable
  - If there's dirty read
    - reader commits before writer = not recoverable
    - writer commits before reader = recoverable
- Cascadeless
  - There are dirty reads = Not cascadeless
  - There are no dirty reads = cascadeless



T1	T2
r(Y)	
	r(Y)
	w(X)
r(X)	
w(Y)	
	w(Y)
	commit
commit	

- conflict serializable?

Conflicting Pairs:

T1	T2
r(Y)	
	r(Y)
	w(X)
r(X)	
w(Y)	
	w(Y)
	commit
commit	

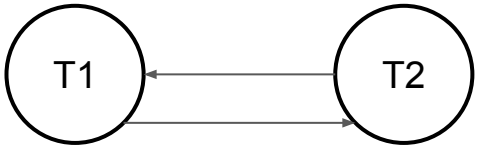
- conflict serializable?

Conflicting Pairs:
w2(x), r1(x), T2 -> T1

T1	T2
r(Y)	
	r(Y)
	w(X)
r(X)	
w(Y)	
	w(Y)
	commit
commit	

- conflict serializable?

Conflicting Pairs:
w2(x), r1(x), T2 -> T1
w1(Y), w2(Y), T1 -> T2



There's a trivial cycle, thus not conflict serializable

T1	T2
r(Y)	
	r(Y)
	w(X)
r(X)	
w(Y)	
	w(Y)
	commit
commit	

- serializable?

T1	T2
r(Y)	
	r(Y)
	w(X)
r(X)	
w(Y)	
	w(Y)
	commit
commit	

- serializable?

No.

T1 read T2's write of X
But T2 wrote the final value to Y

It is neither equivalent to T1;T2 nor T2;T1

T1	T2
r(Y)	
	r(Y)
	w(X)
r(X)	
w(Y)	
	w(Y)
	commit
commit	

- recoverable?

T1	T2
r(Y)	
	r(Y)
	w(X)
r(X)	
w(Y)	
	w(Y)
	commit
commit	

- recoverable?

Writer commits before reader = recoverable

T1	T2
r(Y)	
	r(Y)
	w(X)
r(X)	
w(Y)	
	w(Y)
	commit
commit	

- cascadeless?

No. There’s a dirty read

Midterm 2 Review Problems **Q12-14** (Try it yourself!)

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. Is this schedule recoverable?

Midterm 2 Review Problems **Q12-14** (Try it yourself!)

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. Is this schedule recoverable?

No. Writer of the dirty read (T2) commits before the reader of the dirty read (T1).

If the server crashes here (after T1 commits but before T2 commits) the system could be put into an inconsistent state, because the rollback of T2 will undo T2's write of X, and yet T1 could have performed an action based on that write.

Midterm 2 Review Problems **Q12-14** (Try it yourself!)

Consider the following schedule involving two transactions, T1 and T2.

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

13. How to make it recoverable?

Midterm 2 Review Problems **Q12-14** (Try it yourself!)

Consider the following schedule involving two transactions, T1 and T2.

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

13. How to make it recoverable?

T2 must commit before T1.

Midterm 2 Review Problems **Q12-14** (Try it yourself!)

Consider the following schedule involving two transactions, T1 and T2.

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

14. Is the original schedule serializable?

Conflicting Pairs:

Midterm 2 Review Problems **Q12-14**

Consider the following schedule involving two transactions, T1 and T2.

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

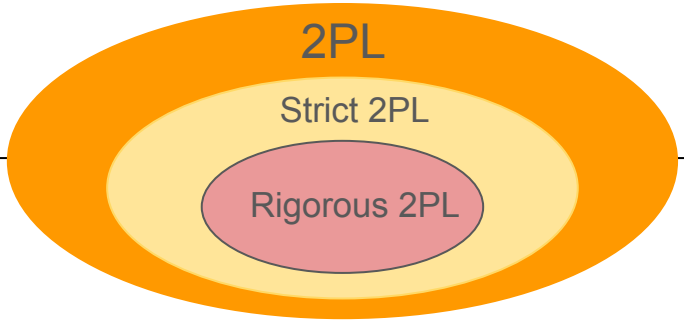
14. Is the original schedule serializable?

Conflicting Pairs:
w2(N), r1(N)

No cycle = conflict serializable = serializable

Concurrency Control - **Locking**

Summary



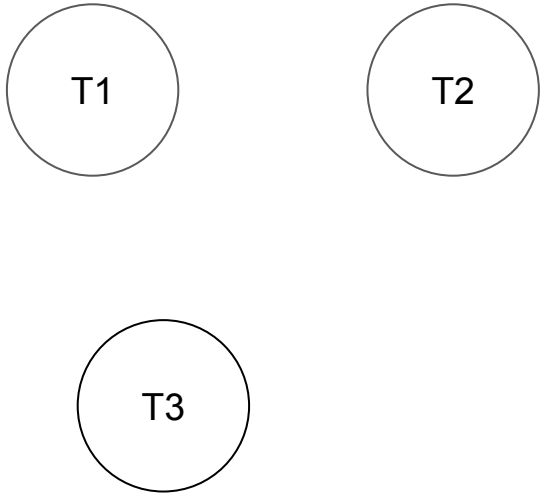
- 2 Phase Locking
  - Regular 2PL - All locks before all unlocks
    - Guarantees Conflict Serializable
  - Strict 2PL - Hold all *exclusive* locks until commit
    - Guarantees Recoverable & Cascadeless (And conflict serializable of course!)
  - Rigorous 2PL - Hold *all* locks until commit
    - Guarantees everything mentioned, also guarantees that transactions commit in the same order as they would in the equivalent serial schedule
- Lock Types
  - Shared (read)
    - can be shared
  - Exclusive (write)
  - Update (read now, but write later)
    - If it is in use, you cannot upgrade from shared to exclusive (otherwise you can)

mode of existing lock for that item (held by a *different* txn)	mode of lock requested for item: **shared**	**exclusive**	**update**
shared	yes	no	yes
exclusive	no	no	no
update	no	no	no

Rigorous 2PL
No update locks (upgrade directly from shared to exclusive)

```
r1(X); r1(Z); w2(Y); w2(X); r3(Z); w3(Y); w1(Z)
```

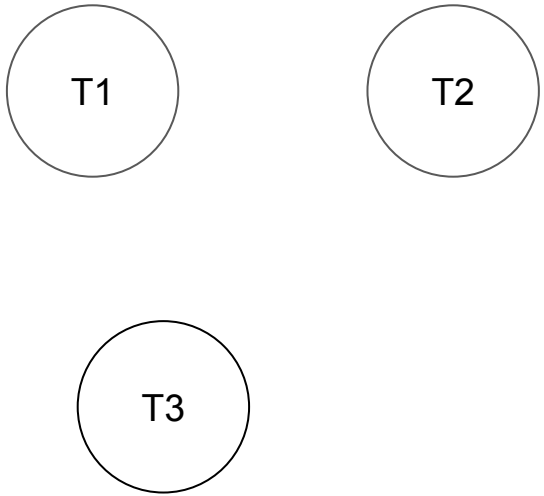
T1	T2	T3
sl(X); r(X)		



Rigorous 2PL
No update locks (upgrade directly from shared to exclusive)

```
r1(X); r1(Z); w2(Y); w2(X); r3(Z); w3(Y); w1(Z)
```

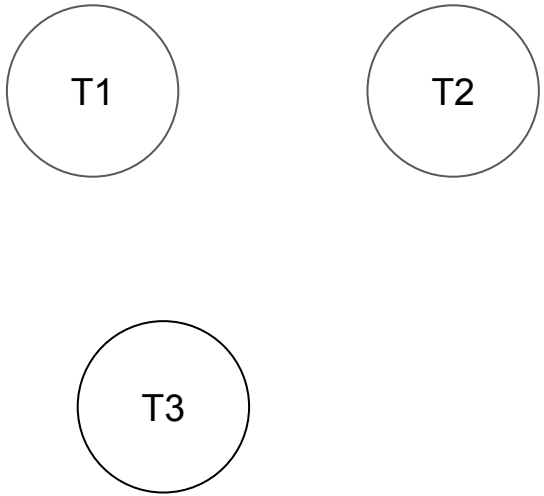
T1	T2	T3
sl(X); r(X)		
sl(Z); r(Z)		



Rigorous 2PL
No update locks (upgrade directly from shared to exclusive)

```
r1(X); r1(Z); w2(Y); w2(X); r3(Z); w3(Y); w1(Z)
```

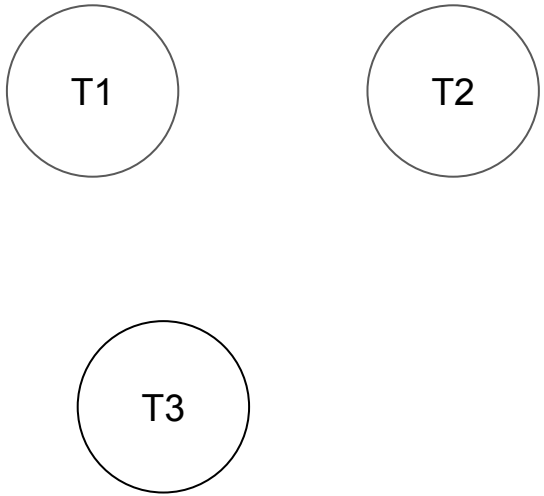
T1	T2	T3
sl(X); r(X)		
sl(Z); r(Z)		
	xl(Y); w(Y)	



Rigorous 2PL
No update locks (upgrade directly from shared to exclusive)

```
r1(X); r1(Z); w2(Y); w2(X); r3(Z); w3(Y); w1(Z)
```

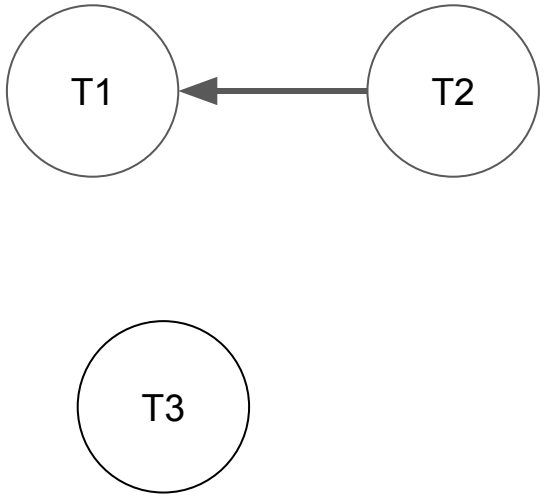
T1	T2	T3
sl(X); r(X)		
sl(Z); r(Z)		
	xl(Y); w(Y)	
	xl(X)	



Rigorous 2PL
No update locks (upgrade directly from shared to exclusive)

```
r1(X); r1(Z); w2(Y); w2(X); r3(Z); w3(Y); w1(Z)
```

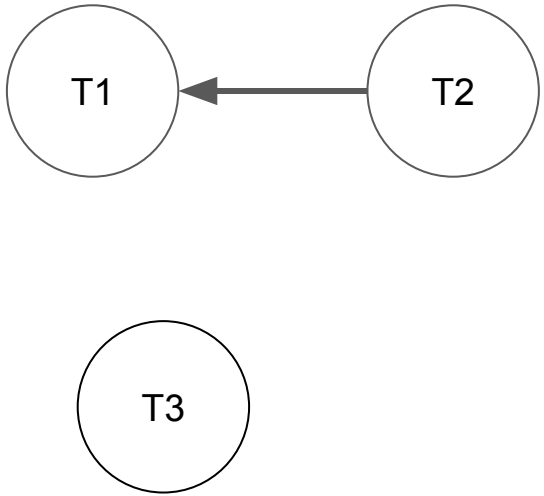
T1	T2	T3
sl(X); r(X)		
sl(Z); r(Z)		
	xl(Y); w(Y)	
	xl(X)	
	denied	
	wait for T1	



Rigorous 2PL
No update locks (upgrade directly from shared to exclusive)

```
r1(X); r1(Z); w2(Y); w2(X); r3(Z); w3(Y); w1(Z)
```

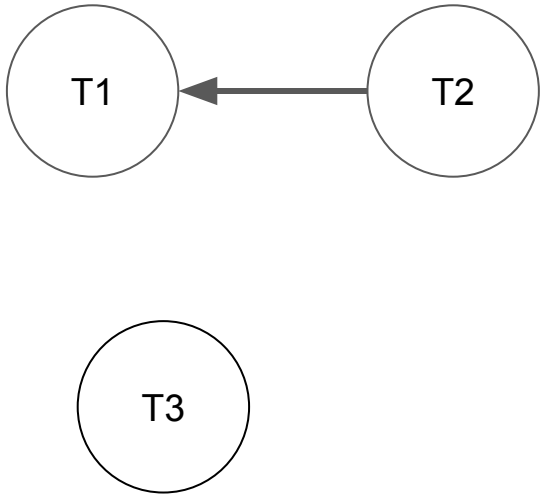
T1	T2	T3
sl(X); r(X)		
sl(Z); r(Z)		
	xl(Y); w(Y)	
	xl(X)	
	denied	
	wait for T1	
		sl(Z); r(Z)



Rigorous 2PL
No update locks (upgrade directly from shared to exclusive)

```
r1(X); r1(Z); w2(Y); w2(X); r3(Z); w3(Y); w1(Z)
```

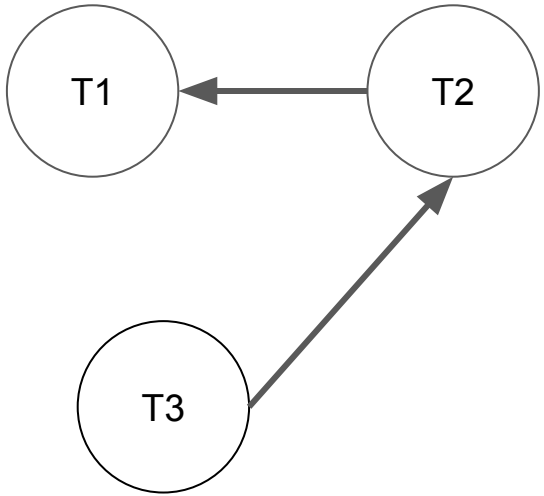
T1	T2	T3
sl(X); r(X)		
sl(Z); r(Z)		
	xl(Y); w(Y)	
	xl(X)	
	denied	
	wait for T1	
		sl(Z); r(Z)
		xl(Y)



Rigorous 2PL
No update locks (upgrade directly from shared to exclusive)

```
r1(X); r1(Z); w2(Y); w2(X); r3(Z); w3(Y); w1(Z)
```

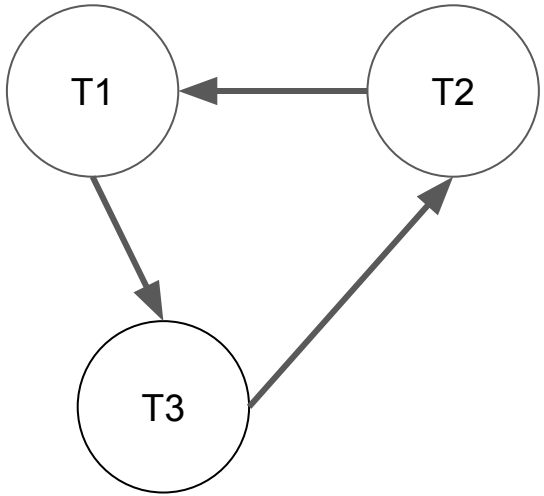
T1	T2	T3
sl(X); r(X)		
sl(Z); r(Z)		
	xl(Y); w(Y)	
	xl(X)	
	denied	
	wait for T1	
		sl(Z); r(Z)
		xl(Y)
		denied
		wait for T2



Rigorous 2PL
No update locks (upgrade directly from shared to exclusive)

```
r1(X); r1(Z); w2(Y); w2(X); r3(Z); w3(Y); w1(Z)
```

T1	T2	T3
sl(X); r(X)		
sl(Z); r(Z)		
	xl(Y); w(Y)	
	xl(X)	
	denied	
	wait for T1	
		sl(Z); r(Z)
		xl(Y)
		denied
		wait for T2
xl(Z)		
denied		
wait for T3		



Midterm 2 Review Problems **Q16**

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

16. 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.

T1	T2

Midterm 2 Review Problems **Q16**

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

16. 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.

T1	T2
sl(X); r(X)	

Midterm 2 Review Problems **Q16**

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

16. 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.

T1	T2
sl(X); r(X)	
	sl(X); r(X)

Midterm 2 Review Problems **Q16**

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

16. 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.

T1	T2
sl(X); r(X)	
	sl(X); r(X)
xl(Y); w(Y)	

Midterm 2 Review Problems **Q16**

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

16. 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.

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

Midterm 2 Review Problems **Q16**

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

16. 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.

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

Midterm 2 Review Problems **Q16**

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

16. 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.

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	

Midterm 2 Review Problems **Q16**

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

16. 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.

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)	

Midterm 2 Review Problems **Q16**

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

16. 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.

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)

Midterm 2 Review Problems **Q16**

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

16. 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.

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)

Midterm 2 Review Problems **Q16**

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

16. 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.

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)

Midterm 2 Review Problems **Q16**

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

16. 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.

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)

Concurrency Control
- **Timestamp**

Summary: Single Version without commit bits

Summary: Timestamp Rules for Reads and Writes

when not using commit bits

- When T tries to read A:
  - if TS(T) < WTS(A), roll back T and restart it
    - T's read is too late
  - else allow the read
    - set RTS(A) = max(TS(T), RTS(A))
- When T tries to write A:
  - if TS(T) < RTS(A), roll back T and restart it
    - T's write is too late
  - else if TS(T) < WTS(A), ignore the write and let T continue
    - in the equiv serial sched, T's write would be overwritten
  - else allow the write
    - set WTS(A) = TS(T)

Summary: Single Version with commit bits

Summary: Timestamp Rules for Reads and Writes

when using commit bits

- When T tries to read A:
  - if TS(T) < WTS(A), roll back T and restart it
    - T's read is too late
  - else allow the read **(but if c(A) == false, make it wait)**
    - set RTS(A) = max(TS(T), RTS(A))

- When T tries to write A:
  - if TS(T) < RTS(A), roll back T and restart it
    - T's write is too late
  - else if TS(T) < WTS(A), ignore the write and let T continue **(but if c(A) == false, make it wait)**
    - in the equiv serial sched, T's write would be overwritten
  - else allow the write
    - set WTS(A) = TS(T) **(and set c(A) to false)**

Summary: Single Version with commit bits 2 (intuitive way)

Single Version with Commit Bits	
Rules for Reads (Txn T wants to read A) ● Is the read too late? ○ If TS(T) < WTS(A) → Deny and Roll back ● Would this be a dirty read? ○ If c(A) = false → Deny and WAIT * ● Otherwise, Allow and update RTS(A) if necessary *Exception: If TS(T) = WTS(A), just allow. (We will not see this today)	Rules for Writes (Txn T wants to write A) ● Is the write too late? ○ If TS(T) < RTS(A) → Deny and Roll back ● Should we ignore this write? ○ If TS(T) < WTS(A) ■ If c(A) = false → Deny and WAIT ■ If c(A) = true → IGNORE ● Otherwise, Allow. Update WTS(A) and make sure c(A) is false

Summary: Multiversion Timestamps

Multiversion Timestamps	
Rules for Reads: ● Transaction T wants to read A: ALWAYS Allow ● Find the version of A that T should read, according to timestamps ○ Update the RTS of that version of A accordingly	Rules for Writes: ● Determine the version of A that T would be overwriting ● Is the write too late? ○ If TS(T) < RTS(That version of A) → Deny and roll back ● Otherwise create a NEW version of A with RTS = 0

With commit bits; Only one version for each data item

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300
r2(Z)	allowed	RTS is not changed (see below)

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300
r2(Z)	allowed	RTS is not changed (see below)
w3(Y)	allowed	WTS(Y) = 300, C(Y) remains false

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300
r2(Z)	allowed	RTS is not changed (see below)
w3(Y)	allowed	WTS(Y) = 300, C(Y) remains false
r3(X)	denied; make wait	TS(T3) >= WTS(X) but c(X) == false

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300
r2(Z)	allowed	RTS is not changed (see below)
w3(Y)	allowed	WTS(Y) = 300, C(Y) remains false
r3(X)	denied; make wait	TS(T3) >= WTS(X) but c(X) == false
c2	allowed	c(X) = true

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300
r2(Z)	allowed	RTS is not changed (see below)
w3(Y)	allowed	WTS(Y) = 300, C(Y) remains false
r3(X)	denied; make wait	TS(T3) >= WTS(X) but c(X) == false
c2	allowed	c(X) = true
r3(X)	allowed!	RTS(X) = 300

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300
r2(Z)	allowed	RTS is not changed (see below)
w3(Y)	allowed	WTS(Y) = 300, C(Y) remains false
r3(X)	denied; make wait	TS(T3) >= WTS(X) but c(X) == false
c2	allowed	c(X) = true
r3(X)	allowed!	RTS(X) = 300
c3	allowed	c(Y) = true

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300
r2(Z)	allowed	RTS is not changed (see below)
w3(Y)	allowed	WTS(Y) = 300, C(Y) remains false
r3(X)	denied; make wait	TS(T3) >= WTS(X) but c(X) == false
c2	allowed	c(X) = true
r3(X)	allowed!	RTS(X) = 300
c3	allowed	c(Y) = true
w1(Y)	ignored	TS(T1) >= RTS(Y) but TS(T1) < WTS(Y) and c(Y) == true

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300
r2(Z)	allowed	RTS is not changed (see below)
w3(Y)	allowed	WTS(Y) = 300, C(Y) remains false
r3(X)	denied; make wait	TS(T3) >= WTS(X) but c(X) == false
c2	allowed	c(X) = true
r3(X)	allowed!	RTS(X) = 300
c3	allowed	c(Y) = true
w1(Y)	ignored	TS(T1) >= RTS(Y) but TS(T1) < WTS(Y) and c(Y) == true
r1(X)	denied; roll back	TS(T1) < WTS(X); RTS(Y) = 0

With commit bits; Only one version for each data item

```
s1; r1(Y); s2;
w2(Y); w2(X); s3;
r3(Z); r2(Z);
w3(Y); r3(X); c2;
c3; w1(Y); r1(X);
c1
```

request	response	state changes and/or explanation
r1(Y)	allowed	RTS(Y) = 100
w2(Y)	allowed	WTS(Y) = 200, c(Y) = false
w2(X)	allowed	WTS(X) = 200, c(X) = false
r3(Z)	allowed	RTS(Z) = 300
r2(Z)	allowed	RTS is not changed (see below)
w3(Y)	allowed	WTS(Y) = 300, C(Y) remains false
r3(X)	denied; make wait	TS(T3) >= WTS(X) but c(X) == false
c2	allowed	c(X) = true
r3(X)	allowed!	RTS(X) = 300
c3	allowed	c(Y) = true
w1(Y)	ignored	TS(T1) >= RTS(Y) but TS(T1) < WTS(Y) and c(Y) == true
r1(X)	denied; roll back	TS(T1) < WTS(X); RTS(Y) = 0
c1	doesn't happen	T1 has already been rolled back

Midterm 2 Review Problems **Q18** (Try it yourself!)

18. 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.

Midterm 2 Review Problems **Q18**

18. 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.

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				

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

a. regular timestamp-based concurrency control **<u>without</u> commit bits**

T1	T2
TS = 10 r(M) ***r(N)*** commit	 TS = 20 r(M) w(N) commit

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

a. regular timestamp-based concurrency control **<u>without</u> commit bits**

T1	T2
TS = 10 r(M) ***r(N)*** commit	 TS = 20 r(M) w(N) commit

Response	M	N
allowed	RTS = 10	

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

a. regular timestamp-based concurrency control **<u>without</u> commit bits**

T1	T2
TS = 10 r(M) ***r(N)*** commit	 TS = 20 r(M) w(N) commit

Response	M	N
allowed	RTS = 10	
allowed	RTS = 20	

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

a. regular timestamp-based concurrency control **<u>without</u> commit bits**

T1	T2
TS = 10 r(M)	
	TS = 20 r(M) w(N)
r(N) commit	
	commit

Response	M	N
allowed	RTS = 10	
allowed	RTS = 20	
allowed		WTS = 20

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

a. regular timestamp-based concurrency control **<u>without</u> commit bits**

T1	T2
TS = 10 r(M)	
	TS = 20 r(M) w(N)
r(N) commit	
	commit

Response	M	N
allowed	RTS = 10	
allowed	RTS = 20	
allowed		WTS = 20
Denied; Roll back		

TS(T) < RTS(N)
10 < 20
Read is too late!

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

b. regular timestamp-based concurrency control **<u>with</u> commit bits**

T1	T2
TS = 10 r(M) ***r(N)*** commit	 TS = 20 r(M) w(N) commit

Response M N

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

b. regular timestamp-based concurrency control **<u>with</u> commit bits**

T1	T2
TS = 10 r(M) ***r(N)*** commit	 TS = 20 r(M) w(N) commit

Response	M	N
allowed	RTS = 10	
allowed	RTS = 20	

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

b. regular timestamp-based concurrency control **<u>with</u> commit bits**

T1	T2
TS = 10 r(M)	
	TS = 20 r(M) w(N)
r(N) commit	
	commit

Response	M	N
allowed	RTS = 10	
allowed	RTS = 20	
allowed		WTS = 20, c=false

Denied; Roll back

TS(T) < RTS(N)
10 < 20
Read is too late!

Summary: Timestamp Rules for Reads and Writes **when using commit bits**

- When T tries to read A:
  - if TS(T) < WTS(A), roll back T and restart it
    - T's read is too late
  - else allow the read **(but if c(A) == false, make it wait)**
    - set RTS(A) = max(TS(T), RTS(A))

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

c. **multiversion** timestamp-based concurrency control **<u>without</u> commit bits**.

T1	T2
TS = 10 r(M) ***r(N)*** commit	 TS = 20 r(M) w(N) commit

Response M(0) N(0)

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

c. **multiversion** timestamp-based concurrency control **<u>without</u> commit bits**.

T1	T2
TS = 10 r(M) ***r(N)*** commit	 TS = 20 r(M) w(N) commit

Response	M(0)	N(0)
allowed to read M(0)	RTS=10	

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

c. **multiversion** timestamp-based concurrency control **<u>without</u> commit bits**.

T1	T2
TS = 10	
r(M)	
	TS = 20
	r(M)
	w(N)
r(N)	
commit	
	commit

Response	M(0)	N(0)
allowed to read M(0)	RTS=10	
allowed to read M(0)	RTS=20	

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

c. **multiversion** timestamp-based concurrency control **<u>without</u> commit bits**.

T1	T2	Response	M(0)	N(0)	N(20)
TS = 10					
r(M)		allowed to read M(0)	RTS=10		
	TS = 20				
	r(M)	allowed to read M(0)	RTS=20		
	w(N)	allowed to create N(20)			Created RTS=0
r(N)					
commit					
	commit				

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

c. **multiversion** timestamp-based concurrency control **<u>without</u> commit bits**.

T1	T2
TS = 10	
r(M)	
	TS = 20
	r(M)
	w(N)
r(N)	
commit	
	commit

Response	M(0)	N(0)	N(20)
allowed to read M(0)	RTS=10		
allowed to read M(0)	RTS=20		
allowed to create N(20)			Created RTS=0
allowed to read N(0)		RTS=10	

Midterm 2 Review Problems **Q22**

What would be the response of the system to the request by **T1** to **read N** shown above, assuming all prior actions in the schedule have been handled as usual and that the system is using:

c. **multiversion** timestamp-based concurrency control **without commit bits**.

T1	T2
TS = 10	
r(M)	
	TS = 20
	r(M)
	w(N)
r(N)	
commit	
	commit

Response	M(0)	N(0)	N(20)
allowed to read M(0)	RTS=10		
allowed to read M(0)	RTS=20		
allowed to create N(20)			Created RTS=0
allowed to read N(0)		RTS=10	

In multiversion, we never roll back a read, and never ignore a write