About the Final

• Open book and open notes
  – But you won’t have time to read during final!
  – No laptops, no mobile devices

• Topics: Lectures 1 through 26
  – But somewhat more on topics from the 2nd half of the course (database implementation forward)
  – No ER diagrams and no FDs. (no lecture 5)
  – Minimal, if any, transactions, serializibility
  – No Pig Latin (no lectures 22 and 23)
Review Advice

- Review the lectures mentioned in previous slide

- Review hw2 and hw3

- Review project 1, project 3, and project 4
  - But no Pig Latin on final!

- Practice sample finals posted on website
XQuery

- Standard for high-level querying of databases containing data in XML form
- Based on Quilt, which is based on XML-QL
- Uses XPath to express more complex queries

Readings
- Section 12.2
- [Nothing about XQuery in old Edition]
FLWR ("Flower") Expressions

FOR ... Zero or more
LET... Zero or more
WHERE... Zero or more
RETURN... Zero or one

Exactly one
Find all book titles published after 1995:

FOR $x$ IN document("bib.xml")/bib/book
WHERE $x$/year/text() > 1995
RETURN $x$/title

Result:
<title> abc </title>
<title> def </title>
<title> ghi </title>
FOR-WHERE-RETURN

Equivalently (perhaps more geekish)

RETURN $x$

And even shorter:

The query:

\[
\text{FOR } \$x \text{ IN document("bib.xml")/bib/book[year > 1995] /title}
\]
\[
\text{RETURN } \$x
\]

Is rewritten by the system into:

\[
\text{FOR } \$x \text{ IN document("bib.xml")/bib/book[year/text() > 1995] /title}
\]
\[
\text{RETURN } \$x
\]
FOR-WHERE-RETURN

- Find all book titles and the year when they were published:

```xml
FOR $x IN document("bib.xml")/ bib/book
RETURN <answer>
    <title>{ $x/title/text() } </title>
    <year>{ $x/year/text() } </year>
</answer>
```

Result:

```xml
<answer> <title> abc </title> <year> 1995 </year> </answer>
<answer> <title> def </title> <year> 2002 </year> </answer>
<answer> <title> ghk </title> <year> 1980 </year> </answer>
```
FOR-WHERE-RETURN

- Notice the use of "{" and "}"  
- What is the result without them?

```
FOR $x IN document("bib.xml")/ bib/book
RETURN <answer>
    <title>  $x/title/text()  </title>
    <year>  $x/year/text()  </year>
</answer>
```
FOR-WHERE-RETURN

- Notice the use of "{" and "}".
- What is the result without them?

```xml
FOR $x IN document("bib.xml")/bib/book
RETURN <answer>
    <title> $x/title/text() </title>
    <year> $x/year/text() </year>
</answer>
```

```xml
<answer> <title> $x/title/text() </title> <year> $x/year/text() </year> </answer>
```
Nesting

For each author of a book by Morgan Kaufmann, list all books she published:

```
FOR $b IN document("bib.xml")/bib,
   $a IN $b/book[publisher/text()="Morgan Kaufmann"]/author
RETURN <result>
   { $a,
     FOR $t IN $b/book[author/text()=$a/text()]/title
     RETURN $t
   }
</result>
```

In the **RETURN** clause comma concatenates XML fragments
Result

<result>
  <author>Jones</author>
  <title>abc</title>
  <title>def</title>
</result>

<result>
  <author>Smith</author>
  <title>ghi</title>
</result>
Aggregates

Find all books with more than 3 authors:

```
FOR $x IN document("bib.xml")/bib/book
WHERE count($x/author)>3
RETURN $x
```

- `count` = a function that counts
- `avg` = computes the average
- `sum` = computes the sum
- `distinct-values` = eliminates duplicates
Aggregates

Same thing:

```
FOR $x IN document("bib.xml")/bib/book[count(author)>3]
RETURN $x
```
Eliminating Duplicates

Print all authors:

```
FOR $a IN distinct-values($b/book/author/text())
RETURN <author> { $a } </author>
```

Note: distinct-values applies ONLY to values, NOT elements
The LET Clause

Find books whose price is larger than average:

```
FOR $b in document(“bib.xml”)/bib
LET $a:=avg($b/book/price/text())
FOR $x in $b/book
WHERE $x/price/text() > $a
RETURN $x
```
Flattening

- Compute a list of (author, title) pairs

**Input:**
```xml
<book>
  <title> Databases </title>
  <author> Widom </author>
  <author> Ullman </author>
</book>
```

**Output:**
```xml
<answer>
  <title> { $x } </title>
  <author> { $y } </author>
</answer>
```
Re-grouping

• For each author, return all titles of her/his books

FOR $b$ IN document("bib.xml")/bib,
   $x$ IN $b$/book/author/text()
RETURN

<answer>
  <author> efg </author>
  <title> abc </title>
  <title> klm </title>
  . . . .
</answer>

Result:

<answer>
  <author> efg </author>
  <title> abc </title>
  <title> klm </title>
  . . . .
</answer>

What about duplicate authors?
Re-grouping

• Same, but eliminate duplicate authors:

```xml
FOR $b$ IN document("bib.xml"):bib
LET $a$ := distinct-values($b/book/author/text()$)
FOR $x$ IN $a$
RETURN
  <answer>
    <author>$x</author>
    { FOR $y$ IN $b/book[author/text()=$x]/title
      RETURN $y$ }
  </answer>
```
Re-grouping

• Same thing:

```xml
FOR $b$ IN document("bib.xml")/bib,
   $x$ IN distinct-values($b$/book/author/text())
RETURN
  <answer>
    <author> $x$ </author>
    { FOR $y$ IN $b$/book[author/text()=$x$]/title
      RETURN $y$ }
  </answer>
```
SQL and XQuery Side-by-side

Product(pid, name, maker, price)  Find all product names, prices, sort by price

SELECT x.name, x.price
FROM Product x
ORDER BY x.price

FOR $x in document(“db.xml”)/db/Product/row
ORDER BY $x/price/text()
RETURN <answer>
  { $x/name, $x/price }
</answer>
XQuery’s Answer

<answer>
  <name> abc </name>
  <price> 7 </price>
</answer>

<answer>
  <name> def </name>
  <price> 23 </price>
</answer>

Notice: this is NOT a well-formed document!
(WHY ???)

......
Producing a Well-Formed Answer

```xml
<aQuery>
{  FOR $x$ in document("db.xml")/db/Product/row
    ORDER BY $x$/price/text()
    RETURN <answer>
        { $x$/name, $x$/price }
    </answer>
}
</aQuery>
```
XQuery’s Answer

```xml
<aQuery>
  <answer>
    <name> abc </name>
    <price> 7 </price>
  </answer>
</aQuery>

Now it is well-formed!

```xml
<aQuery>
  <answer>
    <name> def </name>
    <price> 23 </price>
  </answer>
</aQuery>

```xml
.....
</aQuery>
```
Find all products made in Seattle

SQL:

```
SELECT x.name
FROM Product x, Company y
WHERE x.maker = y.cid
    and y.city = "Seattle"
```

XQuery:

```
FOR $r in document("db.xml")/db,
    $x in $r/Product/row,
    $y in $r/Company/row
WHERE
    $x/maker/text() = $y/cid/text()
    and $y/city/text() = "Seattle"
RETURN { $x/name }
```

Cool XQuery:

```
FOR $y in /db/Company/row[city/text()="Seattle"],
    $x in /db/Product/row[maker/text()=$y/cid/text()]
RETURN { $x/name }
```
<product>
  <row> <pid>123</pid> <name>abc</name> <maker>efg</maker> </row>
  <row> .... </row>
  ...
</product>

<product>
  ....
</product>

....
For each company with revenues < 1M count the products over $100

```sql
SELECT y.name, count(*)
FROM Product x, Company y
WHERE x.price > 100 and x.maker=y.cid and y.revenue < 1000000
GROUP BY y.cid, y.name
```

```xml
FOR $r in document(“db.xml”)/db,
    $y in $r/Company/row[revenue/text()<1000000]
RETURN
    <proudCompany>
        <companyName> { $y/name/text() } </companyName>
        <numberOfExpensiveProducts>
            { count($r/Product/row[maker/text()=$y/cid/text()][price/text()>100]) }
        </numberOfExpensiveProducts>
    </proudCompany>
```
SQL and XQuery Side-by-side

Find companies with at least 30 products, and their average price

SQL:
```
SELECT y.name, avg(x.price)
FROM Product x, Company y
WHERE x.make = y.cid
GROUP BY y.cid, y.name
HAVING count(*) > 30
```

XQuery:
```
FOR $r in document("db.xml")/db,
   $y in $r/Company/row
LET $p := $r/Product/row[maker/text()=$y/cid/text()]
WHERE count($p) > 30
RETURN
   <theCompany>
   <companyName> { $y/name/text() }</companyName>
   <avgPrice> avg($p/price/text()) </avgPrice>
</theCompany>
```