# REVISION

CS561-SPRING 2012
WPI, MOHAMED ELTABAKH

# ANNOUNCEMENTS

- **April 19th**
  - Pankaj/Annies presentation (Survey part 1)
  - Final exam is out @4:00pm (Take-home exam)
- **April 24th**
  - No class
  - Submission of the Final-exam answers due @4:00pm
  - No Late submission
- **April 26th**
  - Pankaj/Annies presentation (Survey part 2)

# WHAT'S IN FINAL ???

- **Lectures covered by the instructor**
  - Object-Relational and Object-Oriented Databases
  - Distributed and Parallel Databases
  - Active Databases
  - Data Integration, Data Mining, and OLAP
  - XML model and XML Querying

# XML Querying

# XPATH EXAMPLES 1

| Path Expression | Result |
|---|---|
| bookstore | Selects all the child nodes of the bookstore element |
| /bookstore | Selects the root element bookstore<br><br>**Note:** If the path starts with a slash ( / ) it always represents an absolute path to an element! |
| bookstore/book | Selects all book elements that are children of bookstore |
| //book | Selects all book elements no matter where they are in the document |
| bookstore//book | Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element |
| //@lang | Selects all attributes that are named lang |

# XPATH EXAMPLES 2

| Path Expression | Result |
|---|---|
| /bookstore/book[1] | Selects the first book element that is the child of the bookstore element.<br><br>**Note:** IE5 and later has implemented that [0] should be the first node, but according to the W3C standard it should have been [1]!! |
| /bookstore/book[last()] | Selects the last book element that is the child of the bookstore element |
| /bookstore/book[last()-1] | Selects the last but one book element that is the child of the bookstore element |
| /bookstore/book[position()<3] | Selects the first two book elements that are children of the bookstore element |
| //title[@lang] | Selects all the title elements that have an attribute named lang |
| //title[@lang='eng'] | Selects all the title elements that have an attribute named lang with a value of 'eng' |
| /bookstore/book[price>35.00] | Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00 |
| /bookstore/book[price>35.00]/title | Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00 |

# XPATH QUERIES

**Select all titles of all books??**

**/bookstore/book/title**

**Select the title of the first book??**

**/bookstore/book[1]/title**

**Select the authors of books with price > $35??**

**/bookstore/book[/price >35]/author**

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

# XPATH QUERIES

**Select all books except the last one?**

**/bookstore/book[position() <> last()]/***

**Select books with more than 2 authors??**

**/bookstore/book[/author[last() > 2]]/***

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

# XQUERY EXAMPLE 1

**FOR ...**

**LET...**

**WHERE...**

**RETURN...**

```
for $p IN document("www.irs.gov/taxpayers.xml")//person
for $n IN document("neighbors.xml")//neighbor[ssn = $p/ssn]
return
  <person>
    <ssn> { $p/ssn } </ssn>
    { $n/name }
    <income> { $p/income } </income>
  </person>
```

# XQUERY EXAMPLE 2

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

# XQUERY

**For books above the average price, Return book title and price sorted by price**

**Let** $avg := avg(document("books.xml")/ bookstore/book/price)

**For** $x := document("books.xml")/ bookstore/book

**Where** $x/price > $avg

**Return**

```
<Book>
        <title> $x/title </title>
        <price> $x/price </price>
</Book> SortBy (price)
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

# XQUERY

**Invoice.xml**

```
<Invoice_Document>
  <invoice>
      <account_number>2 </account_number>
      <carrier>AT&T</carrier>
      <total>$0.25</total>
  </invoice>
 <invoice>
      <account_number>1 </account_number>
      <carrier>Sprint</carrier>
      <total>$1.20</total>
  </invoice>
 <invoice>
      <account_number>1 </account_number>
      <total>$0.75</total>
  </invoice>
<auditor> maria </auditor>
</Invoice_Document>
```

**Customer.xml**

```
<Customer_Document>
  <customer>
      <account>1 </account>
      <name>Tom </name>
   </customer >
   <customer>
      <account>2 </account>
      <name>George </name>
   </customer >
</Customer _Document>
```

**For every customer, count the number of invoices. If count > 3, then report the customer name and the count.**

# XQUERY EXAMPLE

- **For every customer, count the number of invoices. If count > 3, then report the customer name and the count.**

```
FOR  $c in (customers.xml)//customer
LET  $cnt = count(document(invoce.xml)//invoice[account_number = $c/account])
 If $cnt > 3  Then
        RETURN
                    <Customer>
                            <name> $c/name </name>
                            <CountInvoices> $cnt </CountInvoices>
                    </Customer>
```

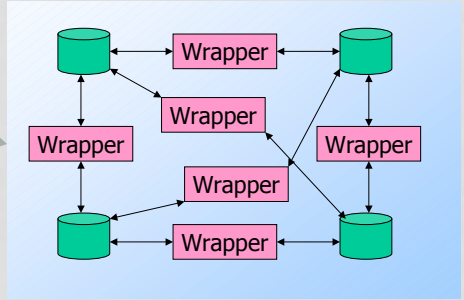
# Data Integration, Data Mining, and OLAP

# HIGHLIGHTS

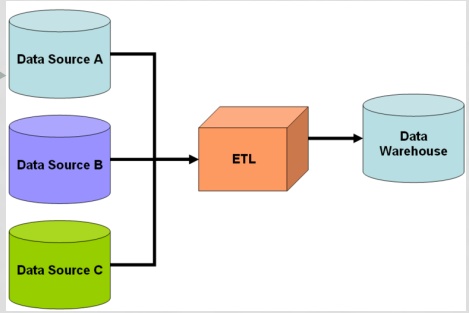
- **What are the different models of data integration?**
- **What is the difference between physical and virtual integration?**
- **May give you a scenario, and you suggest which integration model is better and why.**

# MODELS

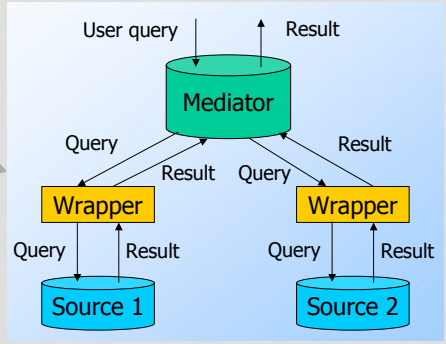
- **Federated Database**
  - **Virtual**




- **Data Warehouse**
  - **Physical**




- **Mediators**
  - **Virtual**



# DATA MINING

- **How to find frequent itemsets in a given dataset**
  - **Apriori algorithm**
- **Association rule mining**

# APRIORI EXAMPLE

| Transaction | Items |
|---|---|
| $t_1$ | Blouse |
| $t_2$ | Shoes,Skirt,TShirt |
| $t_3$ | Jeans,TShirt |
| $t_4$ | Jeans,Shoes,TShirt |
| $t_5$ | Jeans,Shorts |
| $t_6$ | Shoes,TShirt |
| $t_7$ | Jeans,Skirt |
| $t_8$ | Jeans,Shoes,Shorts,TShirt |
| $t_9$ | Jeans |
| $t_{10}$ | Jeans,Shoes,TShirt |
| $t_{11}$ | TShirt |
| $t_{12}$ | Blouse,Jeans,Shoes,Skirt,TShirt |
| $t_{13}$ | Jeans,Shoes,Shorts,TShirt |
| $t_{14}$ | Shoes,Skirt,TShirt |
| $t_{15}$ | Jeans,TShirt |
| $t_{16}$ | Skirt,TShirt |
| $t_{17}$ | Blouse,Jeans,Skirt |
| $t_{18}$ | Jeans,Shoes,Shorts,TShirt |
| $t_{19}$ | Jeans |
| $t_{20}$ | Jeans,Shoes,Shorts,TShirt |

# APRIORI EXAMPLE (CONT'D)

| Scan | Candidates | Large Itemsets |
|---|---|---|
| 1 | {Blouse},{Jeans},{Shoes},<br>{Shorts},{Skirt},{TShirt} | {Jeans},{Shoes},{Shorts}<br>{Skirt},{Tshirt} |
| 2 | {Jeans,Shoes},{Jeans,Shorts},{Jeans,Skirt},<br>{Jeans,TShirt},{Shoes,Shorts},{Shoes,Skirt},<br>{Shoes,TShirt},{Shorts,Skirt},{Shorts,TShirt},<br>{Skirt,TShirt} | {Jeans,Shoes},{Jeans,Shorts},<br>{Jeans,TShirt},{Shoes,Shorts},<br>{Shoes,TShirt},{Shorts,TShirt},<br>{Skirt,TShirt} |
| 3 | {Jeans,Shoes,Shorts},{Jeans,Shoes,TShirt},<br>{Jeans,Shorts,TShirt},{Jeans,Skirt,TShirt},<br>{Shoes,Shorts,TShirt},{Shoes,Skirt,TShirt},<br>{Shorts,Skirt,TShirt} | {Jeans,Shoes,Shorts},<br>{Jeans,Shoes,TShirt},<br>{Jeans,Shorts,TShirt},<br>{Shoes,Shorts,TShirt} |
| 4 | {Jeans,Shoes,Shorts,TShirt} | {Jeans,Shoes,Shorts,TShirt} |
| 5 | ∅ | ∅ |

# OLAP

**Complex SQL queries that involve grouping, aggregation, drill-down, and roll-up**

```
SELECT dealer, year, SUM(price)
FROM (Sales NATURAL JOIN Autos) JOIN Days ON date = day
WHERE model = 'Gobi' AND
      color = 'red' AND
      (year = 2001 OR year = 2002)
GROUP BY year, dealer;
```

**Drill-down**

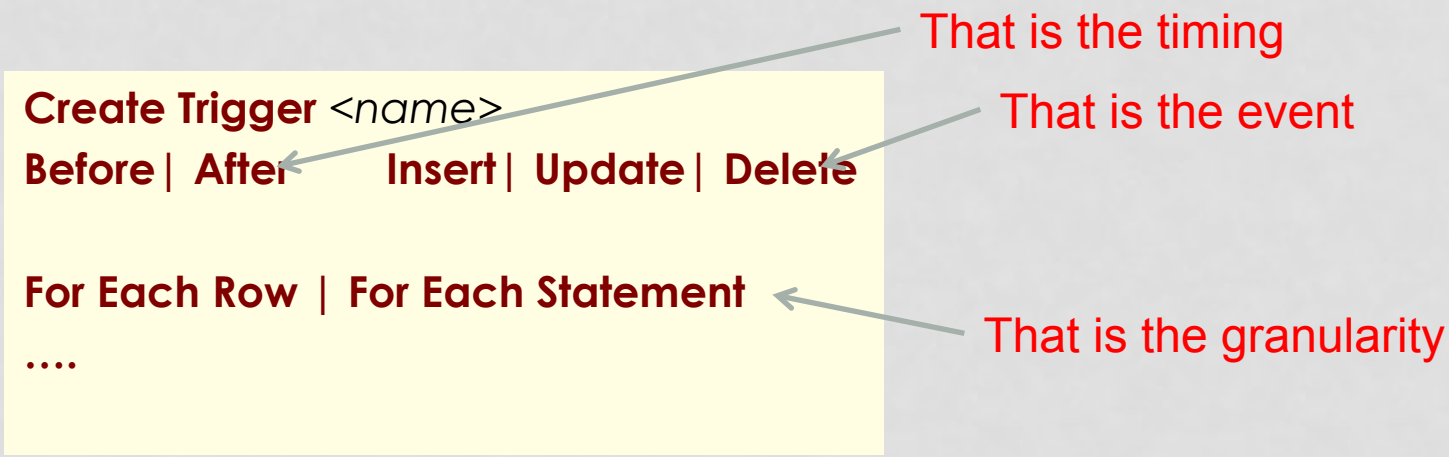
**Roll-up**

```
SELECT dealer, month, SUM(price)
FROM (Sales NATURAL JOIN Autos) JOIN Days ON date = day
WHERE model = 'Gobi' AND color = 'red'
GROUP BY month, dealer;
```

# Active Databases

# DATABASE TRIGGERS

- **What makes a database active?**
  - **Triggers**




**Given an integrity constraint, what triggers to create to enforce that constraint?**

# EXAMPLE

Doctor(**SSN, FirstName, LastName, Specialty, YearsOfExperience, city**)
Patient(**SSN, FirstName, LastName, Address, DOB, PrimaryDoctor_SSN**)
Medicine(**TradeName, UnitPrice, GenericFlag**)
Prescription(**Id, Date, Doctor_SSN, Patient_SSN, TotalCost**)
Prescription_Medicine(**Prescription Id, TradeName, NumOfUnits**)

Create database triggers to ensure that Prescription.TotalCost is always up-to-date and computed as SUM of (Prescription_Medicine.NumOfUnits * Medicine.UnitPrice) for all medicine in that prescription (You may need multiple triggers for that).

- After Insert|Update|Delete (per raw) on Prescription_Medicine table: Makes sure whenever a new record is inserted, updated, or deleted, the prescription_id of that raw will have its TotalCost re-computed and updated.
- After Update (per raw) on Medicine Table: Makes sure if the UnitPrice has changed, then the TotalCost of any prescription containing the updated TradeName is re-computed.

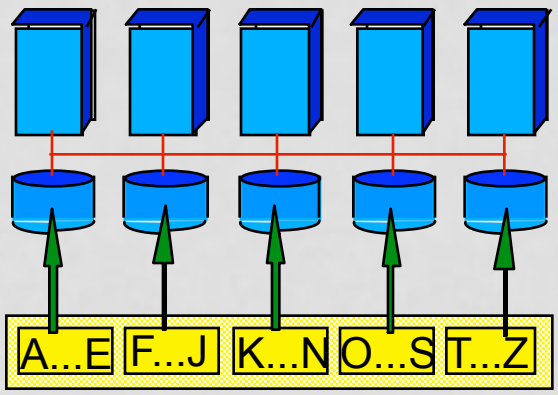
# Distributed and Parallel Databases

# HIGHLIGHTS

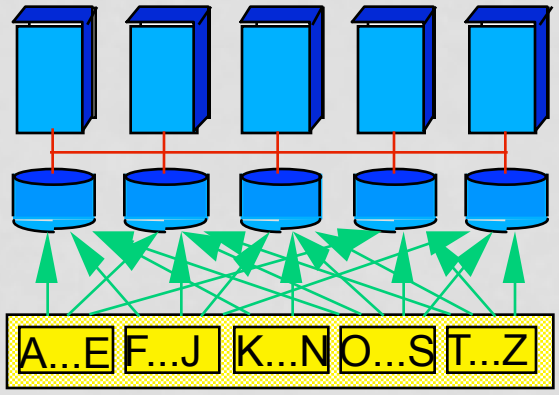
- **Parallel Algorithms**
  - Scan, Join, duplicate elimination, etc.
- **Data Layout (partitioning)**

Given a certain layout, how to execute a certain algorithm

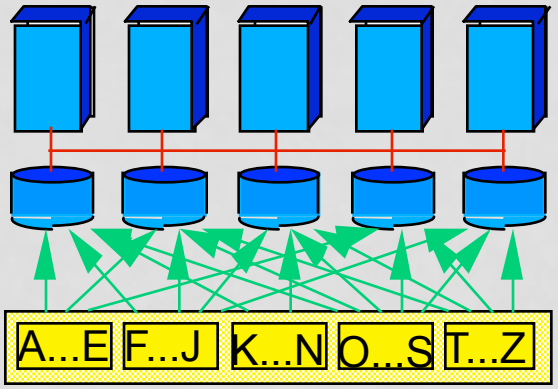
Range partitioning




Hash-based partitioning




Round-robin partitioning

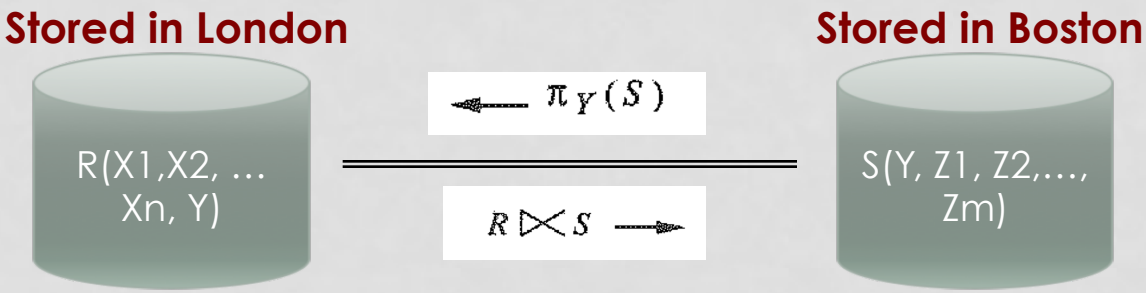


# EXAMPLE: PARALLEL DUPLICATE ELIMINATION

- **If relation is range or hash-based partitioned**
  - Identical tuples are in the same partition
  - So, eliminate duplicates in each partition independently

- **If relation is round-robin partitioned**
  - Re-partition the relation using a hash function
  - So every machine creates m partitions and send the $i^{th}$ partition to machine i
  - machine i can now perform the duplicate elimination

# EXAMPLE: DISTRIBUTED SEMI-JOIN




- Send only S.Y column to R's location
- Do the join based on Y columns in R's location ***(Semi Join)***
- Send the records of R that will join (without duplicates) to S's location
- Perform the final join in S's location

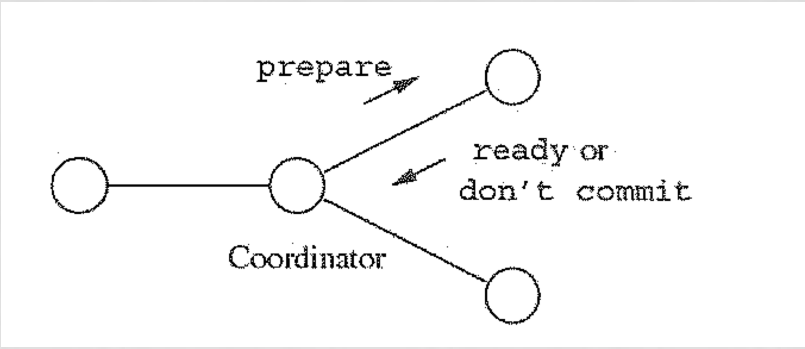
# EXAMPLE: DISTRIBUTED BLOOM JOIN

- Build a bit vector of size K in R's location (all 0's)


- **For every record in R, use a hash function(s) based on Y value (return from 1 to K)**
  - Each function hashes Y to a bit in the bit vector. Set this bit to 1
- Send the bit vector to S's location
- **S will use the same hash function(s) to hash its Y values**
  - If the hashing matched with 1's in all its hashing positions, then this Y is candidate for Join
  - Otherwise, not candidate for join
  - Send S's records having candidate Y's to R's location for join

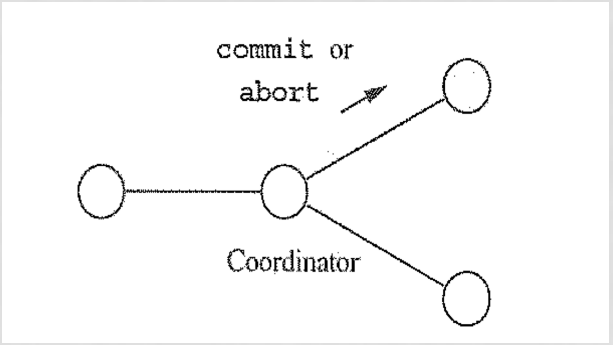
# TWO-PHASE COMMIT

- **Phase 1**
  - Site that initiates T is the ***coordinator***
  - When coordinator wants to commit (complete T), it sends a *"prepare T"* msg to all participant sites
  - Every other site receiving *"prepare T"*, either sends *"ready T"* or *"don't commit T"*
    - A site can wait for a while until it reaches a decision (Coordinator will wait reasonable time to hear from the others)
  - **These msgs are written to local logs**



# TWO-PHASE COMMIT (CONT'D)

- **Phase 2**
  - ***IF coordinator received all "ready T"***
    - Remember no one committed yet
    - Coordinator sends *"commit T"* to all participant sites
    - Every site receiving *"commit T"* commits its transaction
  - ***IF coordinator received any "don't commit T"***
    - *Coordinator sends "abort T"* to all participant sites
    - Every site receiving *"abort T"* commits its transaction




  - **These msgs are written to local logs**

**Example 2: What if all sites in Phase 1 replied "ready T", then one site crashed???**

# Object-Relational and Object-Oriented Database

# HIGHLIGHT

- Mostly syntax for creating objects and querying them ➔ Similar to HW1
  - ODL: Object Definition Language
  - OQL: Object Query Language
  - SQL-99 for Object-Relational Querying
- Revise course slides, HW1, the book chapter associated with the slides on the website

# ONE FINAL NOTE

- **Do not depend only on the slides**
- **You may go and search for something over Internet**
  - E.g., Syntax for SQL, Xquery, XPATH, ODL, etc.
  - Detailed algorithms such as 2-Phase Commit, Apriori, etc.

**Good Luck**