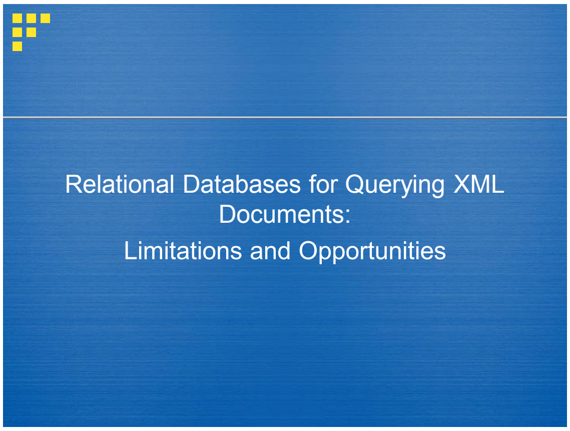
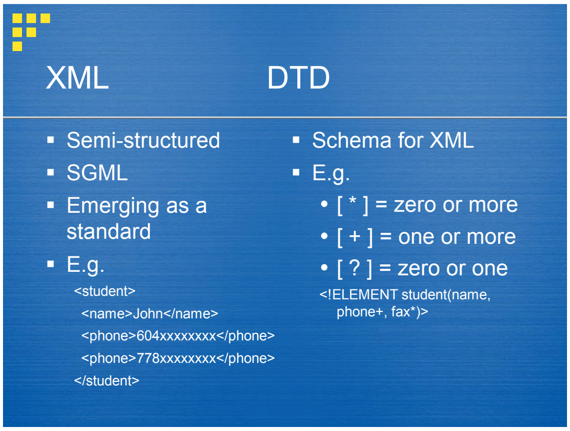


Relational Databases for Querying XML Documents:
Limitations and Opportunities



XML

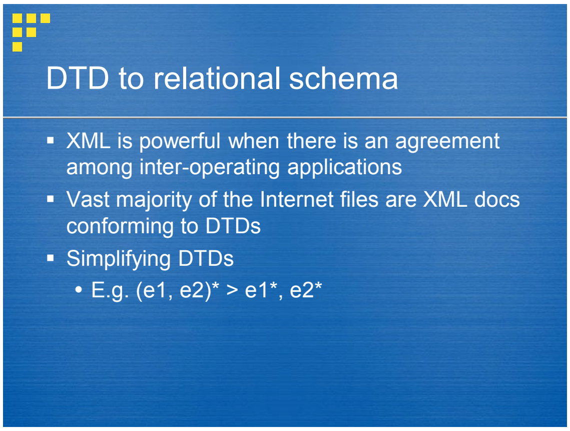
- Semi-structured
- SGML
- Emerging as a standard
- E.g.

```
<student>
  <name>John</name>
  <phone>604xxxxxxxx</phone>
  <phone>778xxxxxxxx</phone>
</student>
```

DTD

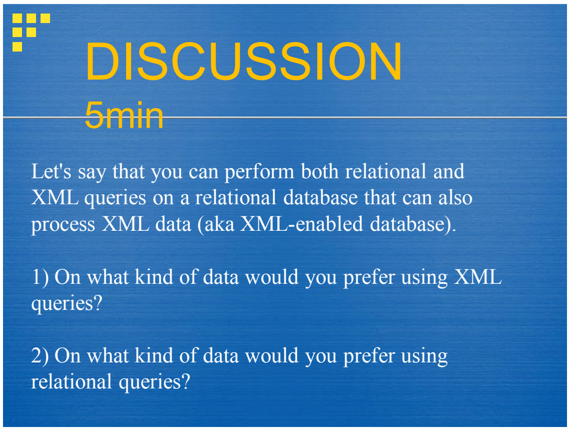
- Schema for XML
- E.g.
  - [*] = zero or more
  - [+] = one or more
  - [?] = zero or one

```
<!ELEMENT student(name,
    phone+, fax*)>
```



DTD to relational schema

- XML is powerful when there is an agreement among inter-operating applications
- Vast majority of the Internet files are XML docs conforming to DTDs
- Simplifying DTDs
  - E.g. (e1, e2)* > e1*, e2*



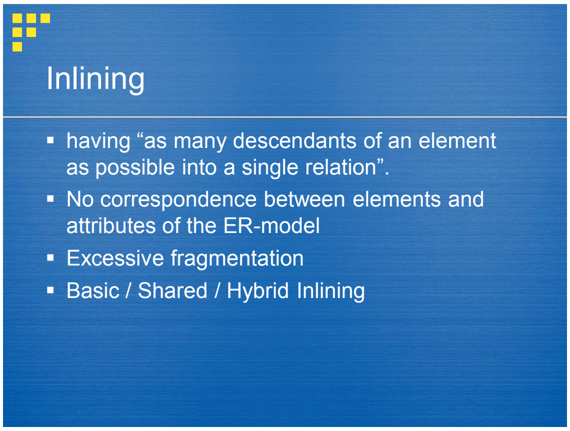
DISCUSSION

5min

Let's say that you can perform both relational and XML queries on a relational database that can also process XML data (aka XML-enabled database).

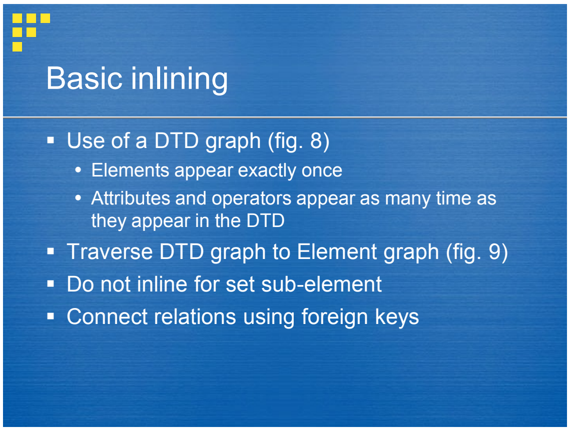
1) On what kind of data would you prefer using XML queries?

2) On what kind of data would you prefer using relational queries?



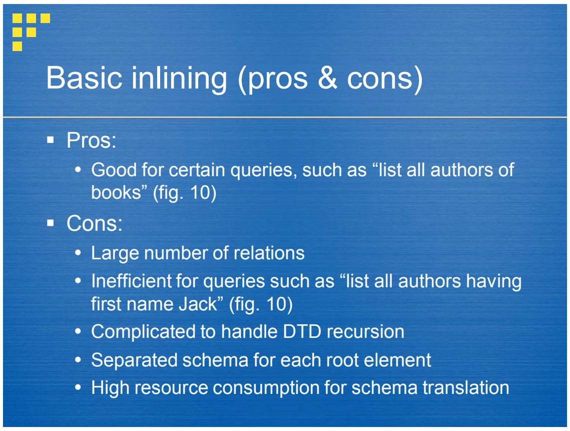
Inlining

- having "as many descendants of an element as possible into a single relation".
- No correspondence between elements and attributes of the ER-model
- Excessive fragmentation
- Basic / Shared / Hybrid Inlining



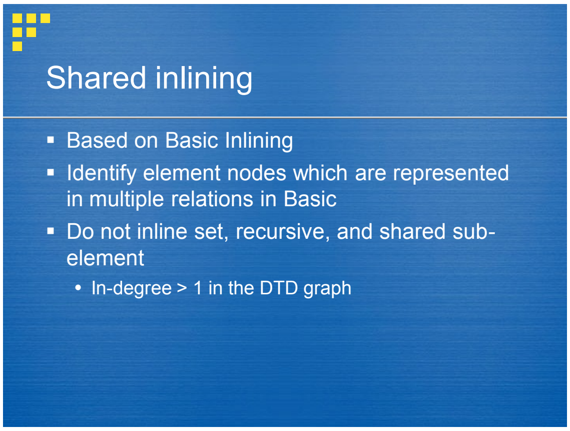
Basic inlining

- Use of a DTD graph (fig. 8)
  - Elements appear exactly once
  - Attributes and operators appear as many time as they appear in the DTD
- Traverse DTD graph to Element graph (fig. 9)
- Do not inline for set sub-element
- Connect relations using foreign keys



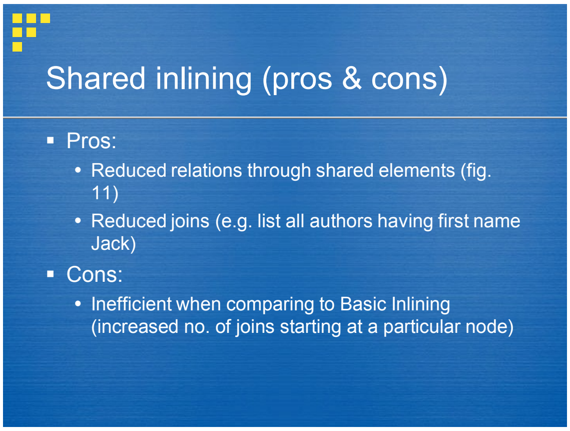
Basic inlining (pros & cons)

- Pros:
  - Good for certain queries, such as “list all authors of books” (fig. 10)
- Cons:
  - Large number of relations
  - Inefficient for queries such as “list all authors having first name Jack” (fig. 10)
  - Complicated to handle DTD recursion
  - Separated schema for each root element
  - High resource consumption for schema translation



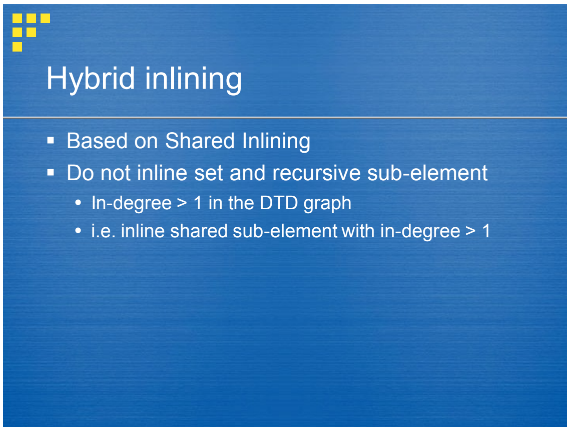
Shared inlining

- Based on Basic Inlining
- Identify element nodes which are represented in multiple relations in Basic
- Do not inline set, recursive, and shared sub-element
  - In-degree > 1 in the DTD graph



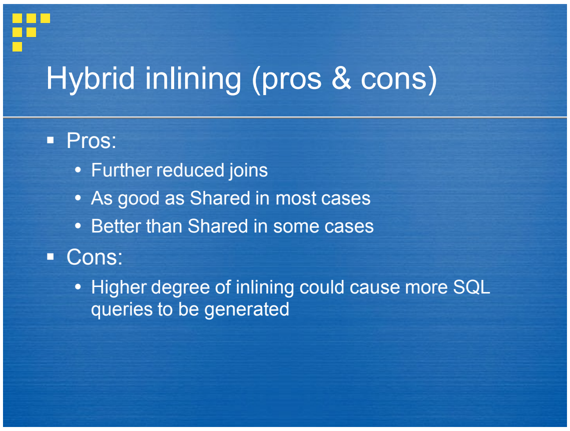
Shared inlining (pros & cons)

- Pros:
  - Reduced relations through shared elements (fig. 11)
  - Reduced joins (e.g. list all authors having first name Jack)
- Cons:
  - Inefficient when comparing to Basic Inlining (increased no. of joins starting at a particular node)



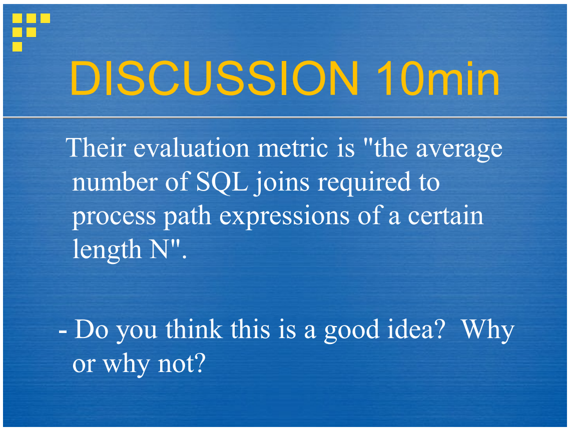
Hybrid inlining

- Based on Shared Inlining
- Do not inline set and recursive sub-element
  - In-degree > 1 in the DTD graph
  - i.e. inline shared sub-element with in-degree > 1



Hybrid inlining (pros & cons)

- Pros:
  - Further reduced joins
  - As good as Shared in most cases
  - Better than Shared in some cases
- Cons:
  - Higher degree of inlining could cause more SQL queries to be generated



DISCUSSION 10min

Their evaluation metric is "the average number of SQL joins required to process path expressions of a certain length N".

- Do you think this is a good idea? Why or why not?

DISCUSSION 10min

The paper concludes that it is possible to use standard Relational DB to evaluate queries over XML data but with limitations.

NOW, If you were to build a XML database, which approach would you take?

- 1) Start with a standard relational technology and try to remove these limitations.
- 2) Start with a new native XML technology and try to add the power and sophistication of current relational DB.

What Goes Around Comes Around

Michael Stonebraker, Joseph M. Hellerstein

Hierarchical (IMS) (late 60s-70s)

Pros:

- facilitates simple data manipulation language (DL/I)

Cons:

- Information is repeated
- Existence depends on parents
- no physical data independence (can't tune physical level without tuning app)
- Not much logical data independence either (can't tune schema without changing app (think views))

Lessons From Hierarchical:

Lesson 1. Physical and logical data independence are highly desirable

Lesson 2. Tree structured data models are very restrictive

Lesson 3. It's a challenge to provide sophisticated logical reorganizations of tree structured data

Lesson 4. Record-at-a-time user interface forces manual query optimization (hard!)

Directed Graph (CODASYL) (70s)

Pros:

- Yeah! Graphs, not trees!
- Can model many-to-many relationships

Cons:

- Still no physical data independence.
- Much more complex than IMS
- Lesson 5: Directed graphs are more flexible than hierarchies, but more complex
- Lesson 6: Loading and recovering directed graphs is more complex than hierarchies

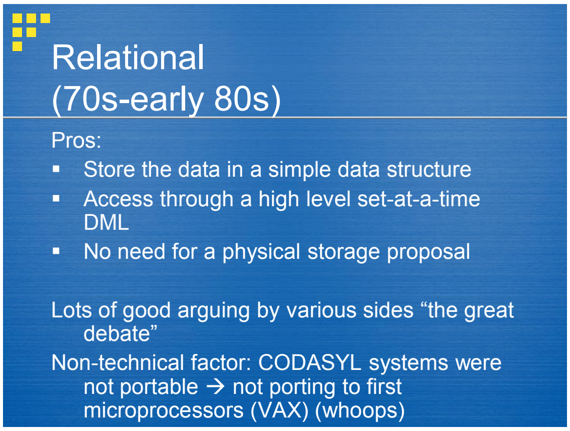
DISCUSSION 5min

The paper says,,,

The XML data model is really nothing different from CODASYL (and others) and CODASYL failed. Don't repeat history!

Do you think that we should try to avoid focussing on ideas that have failed before?

- Why or why not?



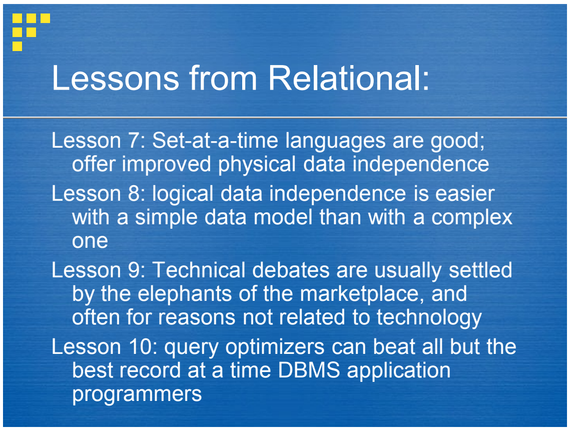
Relational (70s-early 80s)

Pros:

- Store the data in a simple data structure
- Access through a high level set-at-a-time DML
- No need for a physical storage proposal

Lots of good arguing by various sides "the great debate"

Non-technical factor: CODASYL systems were not portable → not porting to first microprocessors (VAX) (whoops)



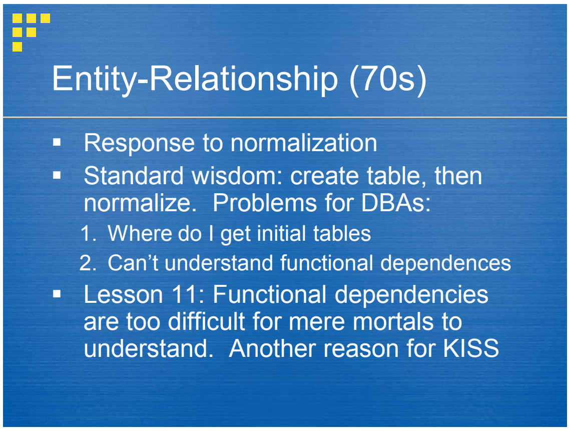
Lessons from Relational:

Lesson 7: Set-at-a-time languages are good; offer improved physical data independence

Lesson 8: logical data independence is easier with a simple data model than with a complex one

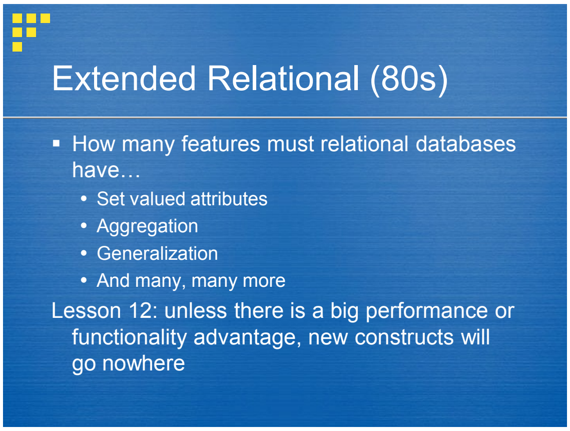
Lesson 9: Technical debates are usually settled by the elephants of the marketplace, and often for reasons not related to technology

Lesson 10: query optimizers can beat all but the best record at a time DBMS application programmers



Entity-Relationship (70s)

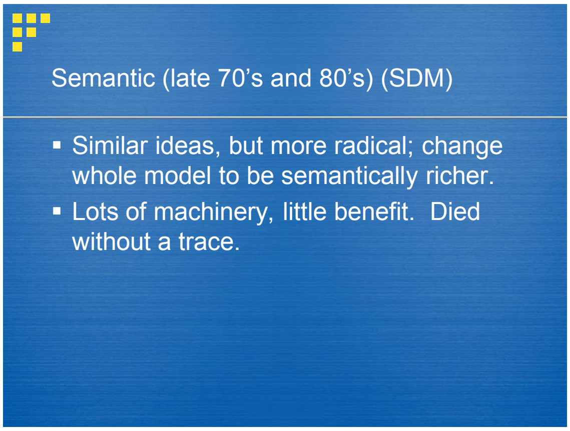
- Response to normalization
- Standard wisdom: create table, then normalize. Problems for DBAs:
  1. Where do I get initial tables
  2. Can't understand functional dependences
- Lesson 11: Functional dependencies are too difficult for mere mortals to understand. Another reason for KISS



Extended Relational (80s)

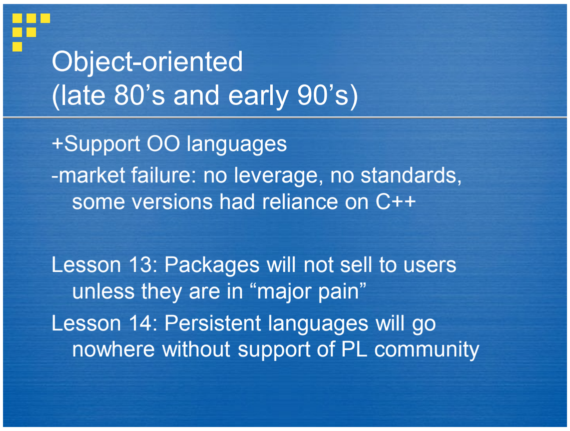
- How many features must relational databases have…
  - Set valued attributes
  - Aggregation
  - Generalization
  - And many, many more

Lesson 12: unless there is a big performance or functionality advantage, new constructs will go nowhere



Semantic (late 70's and 80's) (SDM)

- Similar ideas, but more radical; change whole model to be semantically richer.
- Lots of machinery, little benefit. Died without a trace.



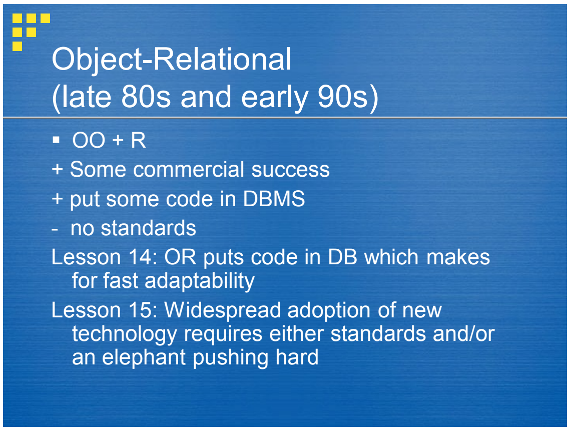
Object-oriented (late 80's and early 90's)

+Support OO languages

-market failure: no leverage, no standards, some versions had reliance on C++

Lesson 13: Packages will not sell to users unless they are in "major pain"

Lesson 14: Persistent languages will go nowhere without support of PL community



Object-Relational (late 80s and early 90s)

- OO + R

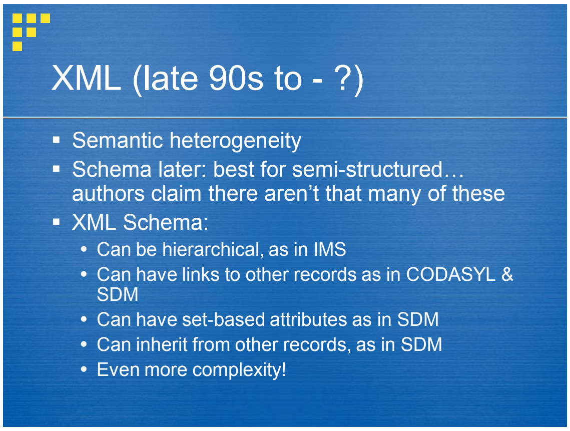
+ Some commercial success

+ put some code in DBMS

- no standards

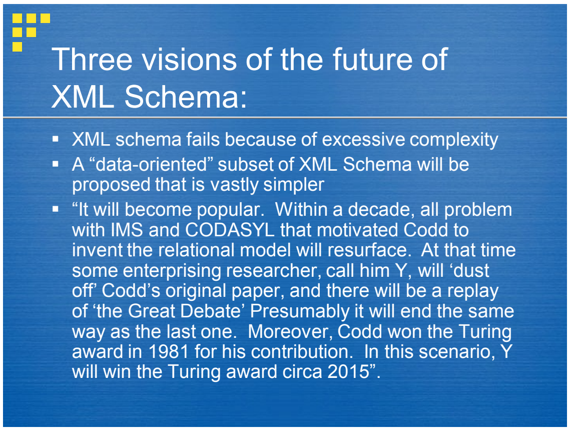
Lesson 14: OR puts code in DB which makes for fast adaptability

Lesson 15: Widespread adoption of new technology requires either standards and/or an elephant pushing hard



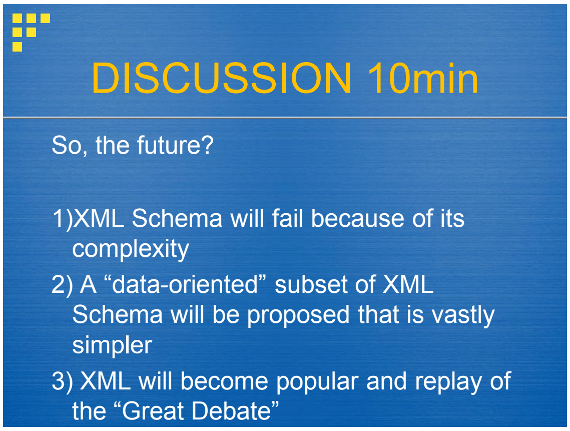
XML (late 90s to - ?)

- Semantic heterogeneity
- Schema later: best for semi-structured… authors claim there aren't that many of these
- XML Schema:
  - Can be hierarchical, as in IMS
  - Can have links to other records as in CODASYL & SDM
  - Can have set-based attributes as in SDM
  - Can inherit from other records, as in SDM
  - Even more complexity!



Three visions of the future of XML Schema:

- XML schema fails because of excessive complexity
- A "data-oriented" subset of XML Schema will be proposed that is vastly simpler
- "It will become popular. Within a decade, all problem with IMS and CODASYL that motivated Codd to invent the relational model will resurface. At that time some enterprising researcher, call him Y, will 'dust off' Codd's original paper, and there will be a replay of 'the Great Debate' Presumably it will end the same way as the last one. Moreover, Codd won the Turing award in 1981 for his contribution. In this scenario, Y will win the Turing award circa 2015".



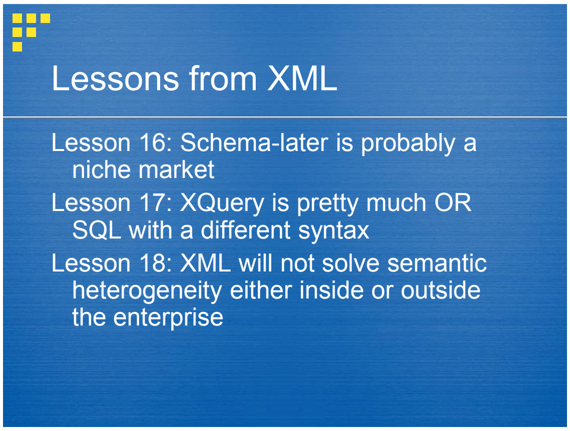
DISCUSSION 10min

So, the future?

1)XML Schema will fail because of its complexity

2) A "data-oriented" subset of XML Schema will be proposed that is vastly simpler

3) XML will become popular and replay of the "Great Debate"

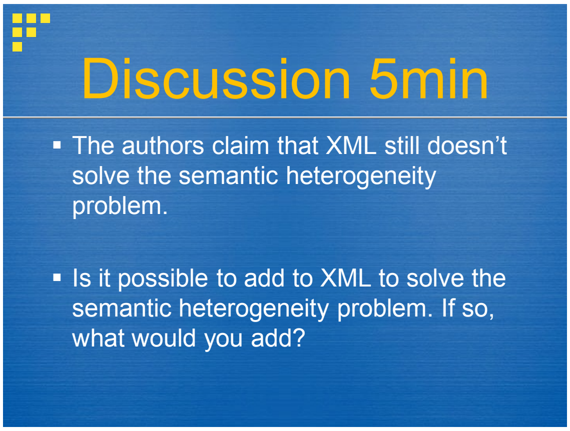


Lessons from XML

Lesson 16: Schema-later is probably a niche market

Lesson 17: XQuery is pretty much OR SQL with a different syntax

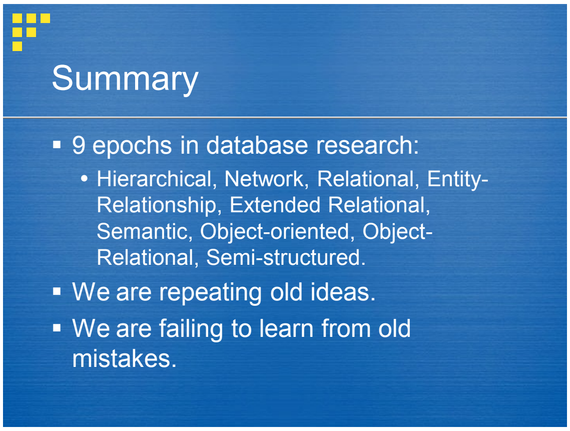
Lesson 18: XML will not solve semantic heterogeneity either inside or outside the enterprise



Discussion 5min

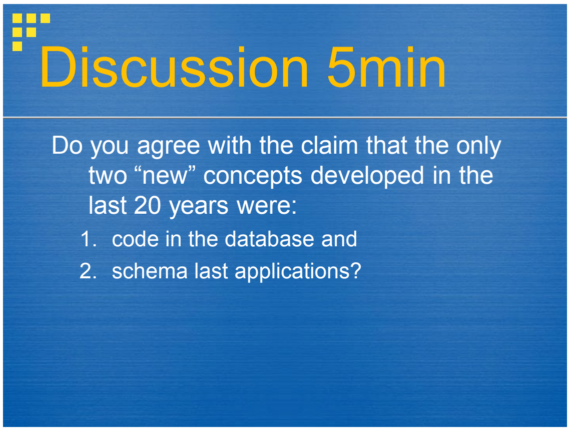
- The authors claim that XML still doesn't solve the semantic heterogeneity problem.

- Is it possible to add to XML to solve the semantic heterogeneity problem. If so, what would you add?



Summary

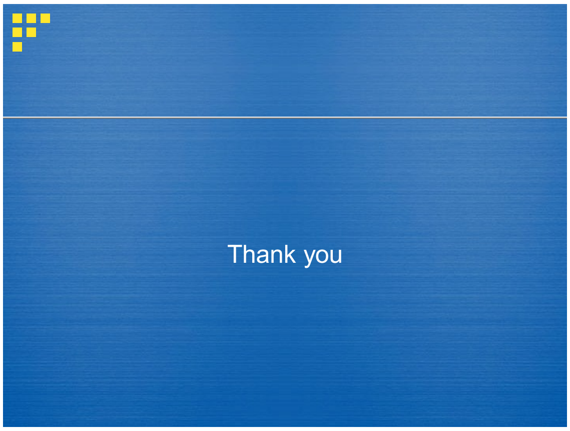
- 9 epochs in database research:
  - Hierarchical, Network, Relational, Entity-Relationship, Extended Relational, Semantic, Object-oriented, Object-Relational, Semi-structured.
- We are repeating old ideas.
- We are failing to learn from old mistakes.



Discussion 5min

Do you agree with the claim that the only two “new” concepts developed in the last 20 years were:

1. code in the database and
2. schema last applications?



Thank you