What Goes Around Comes Around

Michael Stonebraker, Joseph M. Hellerstein

Slides based on slides originally by Garth Shoemaker

Administrative Notes

- HW1 due now
- A few comments about the paper response sample grades
  - Most 2's needed more discussion (though a few were a tad too short on summary)
  - 2.5 – really close
  - Don't turn in late

Goals of the day:

- To cover the first paper
- To give an idea about how I would suggest presenting/leading discussion
- I'll be wearing at least three hats:
  - Presenter
  - Discusser
  - Me

Presenter Hat: Summary

- 9 epochs in database research:
- We are repeating old ideas.
- We are failing to learn from old mistakes.
- We'll cover most of the epochs and lessons

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

Pros:

- Uses 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))

Lesson 1. Physical and logical data independence are highly desirable

- IMS (hierarchical) was particularly bad at this
  - Done to avoid very bad performance
  - This is like the example we saw last week
  - You can't tune an application and guarantee that the DL/1 program can run

Lesson 2. Tree structured data models are very restrictive

- Information is repeated
  - You have to have a single parent, so sometimes you have to duplicate
- Existence depends on parents
  - What do you do if there *is* no parent value?

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

- IMS allowed 2 tree-structured databases to be combined
  - Handy thing to do, but…
  - Created a separate "view", and views were handled differently for users (a real pain)
  - Mapping the view to other databases was very, very challenging

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 6: Loading and recovering directed graphs is more complex than hierarchies

- Independence:
  - In IMS, each database could be independently loaded from a source
  - In CODASYL, it's all connected, so everything had to be loaded at once
- Need to think carefully about disk seeks (no general loading utility)

Discussion

Do you think structuring your data as a graph instead of a tree is inherently too complicated, or does this seem like an implementation issue?

Relational (70s-early 80s)

The proposal in a nutshell:
- 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"

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

- What really brought down IMS?
  - IBM had both IMS and DB/2
  - IBM put DB/2 on VAX, but IMS on mainframes
  - Mainframes had most of the DB market
  - They tried to implement DB/2 on top of IMS and failed (complexity of IMS) →
  - Releasing DB/2 *and* IMS for mainframes →
  - Curtains for IMS

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

- Surprising at the time, but true
  - Like playing chess – the computer can think of *many* more options than a human, even if not all
  - Also similar to compilers

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 can 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.

Discussion

- The last two epochs didn't make much lasting impact. Were they worth doing? Why or why not?

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"

- Absence of leverage – not good enough to just not have to write a load and unload program
- No standards
- No programming language Esperanto – if you had *any* program not written in C++, it wouldn't work

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

- OO + R

+ Some commercial success

+ put some code in DBMS

- no standards

While (as I said) all major DBMSs have some OO features (e.g., stored procedures), that's not as much as proposed in OR space.

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

- Discussion hat: this is an interesting statement. Given that Stonebraker is the person behind the technology that they're claiming had the biggest success in this epoch, how un-biased is he likely to be?

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

- Discussion hat: this is an interesting statement. Given that Stonebraker is the person behind the technology that they're claiming had the biggest success in this epoch, how un-biased is he likely to be?
- Further discussion: most papers are written by their chief proponents. How unbiased are they likely to be?

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

- Discussion hat: this is an interesting statement. Given that Stonebraker is the person behind the technology that they're claiming had the biggest success in this epoch, how un-biased is he likely to be?
- Further discussion: most papers are written by their chief proponents. How unbiased are they likely to be?
- Even further discussion: how should this impact how you read papers?

XML (late 90s to - ?)

- Semantic heterogeneity
- Schema later: best for semi-structured apps… authors claim there aren't many of these
- XML Schema:
  - Can be hierarchical, as in IMS
  - Can link 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!

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

- As mentioned last week:
  OQL (OO) →
  UnQL (unstructured) →
  StrUQL (semi-structured) →
  XMLQL (XML) →
  XQuery (XML)

  Added bonus. XQuery and SQL share a common inventor: Don Chamberlain

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 (from WebCT)

- Are we really circling around?
  - New data models learn from old ones
  - Will there always be a universal data model?

Discussion:

- The authors suggest three possible outcomes for XML schema:
  - Failure
  - Subset adopted
  - XML Schema becomes dominant, and we start the whole cycle again
- Which one of these do you think is most likely, and why?

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

- We haven't seen a whole lot of this yet, but we'll see more when we get to Data Integration

Discussion & Me Hat: Example that's not great

- The authors claim that XML still doesn't solve the semantic heterogeneity problem.
  - What is the semantic heterogeneity problem?
  - What is missing from the XML approach?

Meta comments (me hat)

- Discussion
  - Don't leave it until the end
  - Can be related to both papers
  - Should not be about "right" answer
  - Keep questions short and easy to understand
  - Look at WebCT posts
- Didn't discuss all details – even left some big chunks out
  - I'll give you a list of things not to skip

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

- Sample query: "Get unique Supplier (sno = 16)
  Until no-more{
  Get next within parent (color =red)}
- In contrast:
  SELECT distinct *
  FROM supplier
  WHERE …
- Average IMS query took 17 test runs!

Lesson 5: Directed graphs are more flexible than hierarchies, but more complex

- IMS users keep track of:
  - Current position in DB
  - Parent
- CODASYL users keep track of:
  - Last record touched by app
  - Last record of each record type touched
  - Last record of each set type touched

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 14 (1): Persistent languages will go nowhere without support of PL community

- PL folks have consistently refused to focus on I/O (and DB) →
- No built in functionality →
- Problematic to do this programming

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

- User-defined data types
- User-defined operators
- User-defined functions
- User-defined access methods

Lesson 16: Schema-later is probably a niche market