# What Goes Around Comes Around

Slides based partially on those originally by Garth Shoemaker

## 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
  - 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…
  - Create 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)

## Relational (70s-early 80s)

The proposal in a nutshell:

- Store the data in a simple data structure (tables)
- Access it 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”

# Discussion (Group of 4)

Discuss **one** side of the “Great Debate,” do you think that side’s arguments were reasonable/unreasonable? Why?

| Relational Databases | CODASYL |
|---|---|
| • Nothing as complex as CODASYL can possibly be a good idea<br>• CODASYL does not provide acceptable data independence<br>• Record-at-a-time programming is too hard to optimize<br>• CODASYL and IMS are not flexible enough to easily represent common situations (such as marriage ceremonies) | • COBOL programmers cannot possibly understand the new-fangled relational languages<br>• It is impossible to implement the relational model efficiently<br>• CODASYL can represent tables, so what’s the big deal? |

## 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
  - IMS 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 optimizations 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

## ER (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.

## Discussion (Pairs)

The previous two epochs (ER & Semantic) 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 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

Lesson 14 (the first one): 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

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

## OR lessons

Lesson 14 (the second one): The major  benefits of OR is two-fold: putting code in the database (and thereby blurring the distinction between code and data) and a general purpose extension mechanism that allows OR DBMSs to quickly respond to market requirements

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!

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

OQL (OO) →

UnQL (unstructured) →

StrUQL (semi-structured) →

XMLQL (XML) →

XQuery (XML)

## 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”. Note: Stonebraker won the Turing award in 2014.

## Discussion (Group of 4)

What do you think makes a research era worth revisiting? Think about advancements in other fields, changes in user demand, etc.

What do you think people should get out of revisiting past research?

Are there any examples of successful revisits from your fields?

## Other lessons from XML

Lesson 16: Schema-later is probably a niche market

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

## Discussion (Pairs)

Of all the lessons, which one do you find the most important and which one do you think will likely repeat itself?