“What Goes Around Comes Around”
Michael Stonebraker and Joe Hellerstein

Michael Cafarella
(With slides from Kristen LeFevre and Jim Steinberger)

Administration

- A few people have not yet sent their paper preferences
- I will make final paper assignments tonight

Paper Overview

- Primary Contribution: Retrospective Survey of Data Models
- Also presents opinions for discussion:
  - “Lessons” to learn from past data models
  - Does XML repeat history? In a bad way?

Hierarchical - IMS (~1968)

- Hierarchical model
- Overview:
  - Record types arranged as hierarchy
  - Each type has single parent

  “Type Hierarchy” (Schema)

  Sample Instances

  16. General Supply, Boston, MA
  27. Power Saw, 7, silver, 100, $20

  (Each record has a key)

Some Problems

- Information repeated
  - Schema 1: Part info repeated for each supplier that supplies the part
  - Schema 2: Supplier info repeated for each part it supplies
- Existence depends on parent data
  - Schema 1: What if there is a part not currently supplied by anyone?

DL/1 (Programming Language for IMS)

- “Record-at-a-time” language
- Programmer constructs an algorithm for solving her query; IMS executes it

  Find red parts supplied by Supplier 16

  Get unique Supplier (sno = 16) Until no-more {
    Get next within parent (color = red)
    Get next Part (color = red)
  }
**Lesson 1:**
Physical / Logical data independence == good
- Lifespan of data < Lifespan of apps
  - (Really?)
- Changes to physical/logical data …
  - … should not require changes to apps (ideally).
  - … should not require expensive changes to apps.

**Lesson 2:**
Tree-structured data models are restrictive

**Lesson 3:**
Difficult to reorganize tree-structured data

**Lesson 4:**
Record-at-a-time delegates optimization to the programmer

**Graph / Network - CODASYL (1969)**
- Graph / Network model
- Schema:
  - Improvements:
    - Entities may exist without their “parent (s)”
  - Limitations
    - Still using record-at-a-time DML
    - Still no physical data independence
    - Some logical data independence, but IMS’ s was more flexible
    - More difficult to program against a complex graph than a tree
Lesson 5:
Graphs are more flexible (allowing many-to-many) relationships, but more complex.

Lesson 6:
Loading and recovering graphs is more complex than hierarchies.
- The entire graph must be bulk-loaded at once; IMS trees could be individually loaded.

Relational Model (1970)
- Started with Ted Codd’s 1970 proposal
  - Motivated by heavy maintenance required with IMS applications
    - Recall: IMS provided limited logical data independence, no physical data independence
- Overview:
  - Data stored in tables
  - High-level, set-oriented, DML
  - Underlying physical storage is up to vendors

The Great Debate
- Ideological battle throughout the 1970s
  - Ted Codd & co. advocating relational
  - Charlie Bachman & co. advocating CODASYL (graph/network)
- CODASYL too complex
- Too much dependence on data
- Record-at-a-time too hard to optimize
- Relational model better for complex relationships
- Relational languages too hard
- Implementing relational model efficiently too difficult
- CODASYL can pretend to be relational...

Result 1: Both parties adopted many of each other’s policies while pretending to remain at opposite sides of the ideological spectrum

Result 2: IBM advocated the relational model, and won in the marketplace due to its dominant position in microcomputers

Lesson 7:
Set-at-a-time languages offer better physical data independence.
- Up to the DBMS to optimize based on physical structure.

Lesson 8:
Simpler data models lend themselves to better logical data independence

Lesson 9:
Technological debates are often settled by dollars rather than ideas

Lesson 10:
Query optimizers almost always better than a programmer optimizing manually
Entity-Relationship (mid-1970s)
- Proposed by Peter Chen
  - (not our Peter Chen)
- Novelty: relationships with attributes and multiplicities

As physical model:
- Never caught on (due to little benefit)
As conceptual model:
- Widely used for database schema design
  - Normalization is hard without tables to normalize
  - The E-R model offers a methodology for creating those initial tables
  - Some normalization on an E-R model can be done automatically

Lesson 11:
“Relationships” are easier to understand than “functional dependencies”.

R++ (Early 1980s)
- Following relational success, lots of proposals for new features
  - Examples:
    - Mechanical & VLSI CAD
    - Text Management
    - Time
    - Graphics
    - Set-valued attributes
    - Inheritance
- Offered lots of new functionality, but:
  - Most could be simulated within the existing relational models
  - Did little to improve performance

R++ (Early 1980s)
- Algorithm for writing SIGMOD papers (circa 1984):
  - Consider an application X
  - Try to implement X on a relational DBMS
  - Show why queries are difficult, or poor performance
  - Add a new “feature” to relational model to solve the problem

Semantic Data Model (Early 1980s)
- Viewing relations as “classes”
  - Multiple inheritance, “class”-wide attributes
Semantic Data Model (Early 1980s)
- Same limitations as R++ proposals:
  - This model could already be simulated with the relational model
  - Vendors were more concerned with performance

Lesson 12:
Without large performance / functionality advantages, new constructs will go nowhere.

OO DBs (Mid-1980s)
- Attempts to solve “impedance mismatch”
  - Difficulties in writing database-backed applications
  - Mapping relations to PL objects like “gluing an apple onto a pancake”
- Goal: Integrate data persistence into OO programming languages

Impedance Mismatch
Struct Part{
  int number;
  char* name;
  char* size;
  char* color;
};

OO DBs (Mid-1980s)
- General Idea: Extend a programming language (e.g., C++) with database functionality to support data persistence
  - Initial work targeted toward engineering niche market (i.e., CAD)

Persistent Part p;
Persistent int i;
i = i+1;

OO DBs (Mid-1980s)
- Problems:
  - There are lots of programming languages
    - Adding persistence to all = huge chore
  - Resistance from PL community
  - Getting rid of embedded SQL not enough of a benefit
  - No standards; different OODBs incompatible
  - (persistent C++) Record-at-a-time access + no transaction support = unsuitable for business data processing
Lesson 13:
New systems will not sell to users unless they are in “major pain”

Lesson 14:
Persistent languages require the support of the programming language community

Object-Relational DBs (Mid-1980s)
- Motivated by spatial queries
  - Circa 1982, INGRES team had “haunting” interest in GIS (geographical information systems)
  - Recall: B-trees inefficient for these sorts of queries
  - Recall: R-trees require 2+ dimensional nodes, rather than single-dimensional numeric ranges

Object-Relational DBs (Mid-1980s)
- OR Proposal: User-extension and user-customization to a relational DB
  - User-defined data types (e.g., box)
  - User-defined operators (e.g., box-intersects-box)
  - User-defined functions (e.g., box-intersects-box implementation)
  - User-defined access methods (e.g., R-tree indexing)

Object-Relational DBs (Mid-1980s)
- Major prototype: Postgres
  - Contribution: showed how to build a DBMS engine so new types/functions/etc. could be plugged in
    - Contemporary systems hard-coded their supported sets of data types, access paths, etc.
- Also: Sybase
  - Contribution: stored procedures
    - Using UDFs for application-logic, not just operator-implementation
    - Performance benefit for these operations

Object-Relational DBs (Mid-1980s)
- Postgres: commercialized by Illustra
- Then: Informix acquired Illustra
  - Illustra brought UDTs/UDFs to the table
  - Informix brought market share and transaction-management
- Informix successful with GIS and large-content-repository markets
  - Little success elsewhere

Lesson 14:
OR’s contributions are great!

Lesson 15:
Widespread adoption requires standards or a market giant

Every ORDB has a proprietary way of doing UDFs

- Two (+1) main points exemplified by this work:
  - Schema evolution / "schema later"
  - Complex graph-oriented data model
  - Also: Response to growth of web services / XML as a messaging standard

Schema Later

- Conventional Setting:
  - DBA defines a schema (e.g., Parts)
  - Inserted data must conform to the schema
- "Schema Later":
  - (Interpretation 1) No fixed schema; data is self-describing
    - Primary motivation: "Semi-Structured" data (next slides)
  - (Interpretation 2) Schema is easily changed

Semi-Structured Data

- Motivating application class:
  - Rigidly-structured data
    - Schema-first
  - Rigidly-structured data with text fields
    - e.g. web/business form
    - Schema-first
  - Semi-structured data
    - e.g. Classifieds/personals
    - Schema-last
  - Free Text
    - Schema not-at-all

Semi-Structured Data

- Semi-structured Example

  Person:
  Name: Joe Jones
  Wage: 14.75
  Employer: My_accounting
  Hobbies: skiing, bicycling
  Works for: ref (Fred Smith)
  Favorite joke: Why did the chicken cross the road? To get to the other side
  Office number: 247
  Major skill: accountant

  End Person

  Person:
  Name: Smith, Vanessa
  Wage: 2000
  Favorite coffee: Arabian
  Pastimes: sewing, swimming
  Works_for: Between jobs
  Favorite restaurant: Panera
  Number of children: 3

  End Person:

Semantic Heterogeneity:
- Different sets of attributes
- Same attributes have different formats
- Different attributes have same meaning

Stonebraker & Hellerstein think truly semi-structured data is rare

Schema Evolution

- Alternate interpretation of "schema later"
  - Relational DBMSs (schema first) have heavy-weight mechanisms for changing a schema
    - E.g., ALTER TABLE
  - Open Question: Can we make it easier to modify and evolve schemas?

XML as a Data Model (early 2000s)

- Define schemas using DTDs or XMLSchema
- Data model is complex:
  - Records can be hierarchical (IMS)
  - Records can reference any other record (CODASYL)
  - Records may have set-based attributes (R++)
  - Several modes of inheritance (Semantic)
  - "Union" attributes may be one of several data types (e.g. an int or a string)
    - Complex to index / query
  - All in all, a major KISS violation (?)
(Prediction) 16: Semi-structured data is probably a niche market

(Prediction) 17: XQuery is essentially Object-Relational SQL

(Prediction) 18: XML will not solve semantic heterogeneity

XML Comments
- XML seems here to stay as a document / message format
- Extreme Solution: Replace relational model with XML data model, native implementation
- Various Hybrid Solutions, too
  - E.g., XML data type in relational DBMS

Summary
- 1960s/1970s: Hierarchical (IMS)
- 1970s: Network/Graph (CODASYL)
- 1970s/1980s: Relational
- 1970s: Entity-Relational
- 1970s/1980s: Semantic (SDM/GEM)
- 1980s/1990s: OO & OR
- Late 1990s – present: Semi-structured & XML