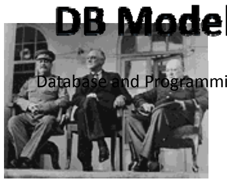
Retrospect on DB Models & Languages



CPS 296.1
Database and Programming Languages: Crossing the Chasm

Jun Yang
Duke University
January 19, 2010

† Thanks to contents/ideas borrowed from
Hellerstein (http://redbook.cs.berkeley.edu/redbook3/lecs.html)
and LeFevre (http://www.eecs.umich.edu/~klefevre/eecs584/Handouts/Stonebraker-Hellerstein.pdf)

Image from http://www.bccc.edu/887421129114724/blank/browse.asp?A=383&BMDRN=2000&BCOB=0&C=52764

Announcements

- Sign up (by email) to lead discussions by Wednesday
  - Student-led discussions start next week!
- Read the two papers on making DBMS object-relational for Thursday
  - Review for "Inclusion of New Types…" due by 9am on Thursday
  - Submit your review on the Blackboard class forum by replying to my post
  - Again, see course website for instructions on reviewing and submission, as well as tips on reading research papers

2

On leading discussions

- Three types of discussion: research papers, survey papers, or tutorials of systems/platforms
- As leaders, you must finish reading/researching in advance
- I will meet you to talk about the lecture
  - By default, during office hours on the day of the lecture before the one you are leading
    - Thursday lecture → meet on Tuesday; Tuesday lecture → meet on Thursday of the preceding week
- Besides providing summary, critique, and answering questions, strive to generate discussion from class
  - A good way is to ask "facilitating" questions

3

Overview

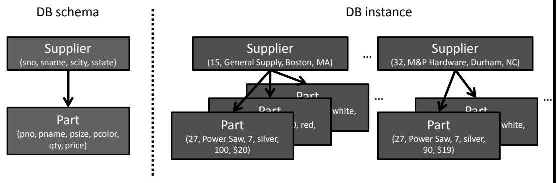
- Stonebraker & Hellerstein. "What Goes Around Comes Around." In *Readings in Database Systems* (aka "the red book"), 4th ed., 2005
  - A retrospective survey of DBMS data models and query languages
  - Lessons to learn from past experience
  - And why XML is doomed

➲What do *you* think?

4

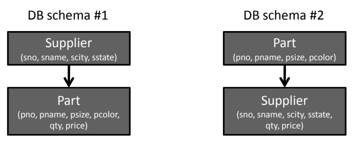
Hierarchical: IMS (1968)

- Organize record types in hierarchies
  - Each non-root type has a single parent type
  - Each record of a non-root type has one parent of the parent type
    - Corollary: each record has a unique HSK (Hierarchical Sequence Key)
- Model simplicity facilitates simple language & implementation


DB schema
Supplier
(sno, sname, scity, sstate)
Part
(pno, pname, psize, pcolor, qty, price)
DB instance
Supplier
(15, General Supply, Boston, MA)
…
Supplier
(32, M&P Hardware, Durham, NC)
Part
white,
Part
), red,
Part
(27, Power Saw, 7, silver, 100, $20)
…
Part
white,
Part
(27, Power Saw, 7, silver, 90, $19)
…

5

Issues with model


DB schema #1
Supplier
(sno, sname, scity, sstate)
Part
(pno, pname, psize, pcolor, qty, price)
DB schema #2
Part
(pno, pname, psize, pcolor)
Supplier
(sno, sname, scity, sstate, qty, price)

- Information is repeated
  - Schema #1: parts info repeated across suppliers
  - Schema #2: supplier info repeated across parts
- Existence depends on parent data
  - Schema #1: what if nobody supplies a part?
  - Schema #2: what if a supplier doesn't supply anything?

6

Issues with language (DL/1)

- Conceptually, records are laid out in HSK order: depth-first, left-to-right → get ingrained in language constructs
- Record-at-a-time language
- Programmer writes an algorithm for solving each query, e.g.:

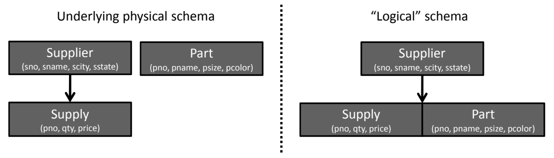
```
▪ get unique Supplier with sno = 15
  until failure do
       get next within parent with pcolor = red
```

Why is this bad?

- Different underlying storage formats (sequential/B-tree/hash) → different restrictions on commands
  - Heavy coupling between storage and client apps
- Different data characteristics → different optimal algorithms
  - Optimization is performed by programmer and DB designer
- Not declarative, poor physical data independence

7

Hacking the model


Underlying physical schema
Supplier
(sno, sname, scity, sstate)
Part
(pno, pname, psize, pcolor)
Supply
(pno, qty, price)
"Logical" schema
Supplier
(sno, sname, scity, sstate)
Supply
(pno, qty, price)
Part
(pno, pname, psize, pcolor)

- Store data in two physical databases
  - No redundancy
- IMS grafts together the two to present a logical view to programmers
  - But lots of restrictions and complexity
  - No complete transparency

8

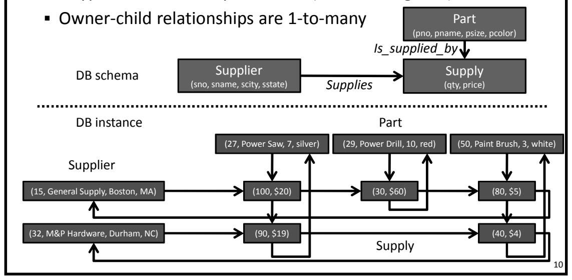
Lessons

- Lesson 1: physical/logical data independence is good
  - Δdata $\gg$ Δapp
  - Changes to physical/logical representations of data should not require expensive changes to apps
- Lesson 2: tree-structured data models are restrictive
  - Force navigation one way
  - Need extensions/hacks to be general
- Lesson 3: logical reorganization of tree-structured data is hard
- Lesson 4: record-at-a-time interface forces programmer to do manual query optimization

9

Graph/network: CODASYL (1969)

- A directed graph where nodes are records types and arcs are "sets" (relationships)
  - A type can have multiple owners (via incoming arcs)
  - Owner-child relationships are 1-to-many


Part
(pno, pname, psize, pcolor)
Is_supplied_by
DB schema
Supplier
(sno, sname, scity, sstate)
Supplies
Supply
(qty, price)
DB instance
Part
(27, Power Saw, 7, silver)
(29, Power Drill, 10, red)
(50, Paint Brush, 3, white)
Supplier
(15, General Supply, Boston, MA)
(100, $20)
(30, $60)
(80, $5)
(32, M&P Hardware, Durham, NC)
(90, $19)
(40, $4)
Supply

10

CODASYL programming

- Bachmann (Turing Award 1973): program by navigation

```
▪ get unique Supplier with sno = 15
  until failure do
       get next Supply in Supplies
       get owner Part through Is_supplied_by
       check pcolor = red
```

- Alternatively, start navigating from Parts

11

Improvements & limitations

- Model is powerful itself to avoid redundancy and dependency on owners' existence
- Arcs are just binary, though n-ary relationships can be simulated
- Language is still record-at-a-time
- Programming over graphs is harder than over trees
- Less logical data independence than IMS
- Still no physical data independence

12

Lessons

- Lesson 5: graphs are more flexible than trees but more complex
- Lesson 6: loading and recovering graphs is harder than trees

Relational (1970)

- Started with the 1970 proposal by Codd (Turing Award 1981)
  - Motivated by heavy maintenance required with IMS applications
- Data stored in flat tables—no nesting
- High-level, set-oriented language
- Underlying physical storage is completely up to vendors
- Example schema and query

  Supplier(sno, sname, scity, sstate)
  Part(pno, pname, psize, pcolor)
  Supplies(sno, pno, qty, price)

```
SELECT * FROM Supplier, Supplies, Part
WHERE Supplier.sno = 15 AND Part.pcolor = 'red'
AND Supplier.sno = Supplies.sno AND Supplies.pno = Part.pno;
```

The “Great Debate”

- Ideological battle throughout the 1970’s
  - Codd et al. advocating relational
  - Bachman et al. advocating CODASYL (graph/network)

- *Relational languages too hard*
- *Implementing relational model efficiently too difficult*
- *CODASYL able to simulate relational*

- *CODASYL too complex*
- *Too much dependence on data layout*
- *Record-at-time too hard to optimize*
- *Relational better for complex relationships*

Image by *NY Times*, http://graphics8.nytimes.com/images/2007/10/31/us/31debate.xlarge3.jpg

How was it resolved?

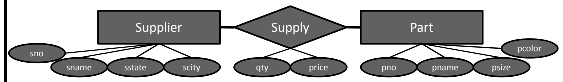
- Both parties adopted many of each other’s policies while pretending to remain at oppose sites of the ideological spectrum
- IBM advocated the relational model, and won in the marketplace due to its dominant position in the microcomputers industry

Lessons

- Lesson 7: set-at-time languages offer better physical data independence
  - Up to the DBMS to optimize physical structure based on data/workload characteristics
- 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
  - ➲Are there exceptions?

E/R model

- Schema expressed in diagrams with “entity” sets connected by “relationship” sets



- Never caught on as a physical/implementation model, but very successful for modeling and DB design
  - Automatic mapping to relational schema possible
- Lesson 11: “relationships” are easier to understand than “functional dependencies”

Relational++/semantic (early 1980's)

- Pure relational seemed inadequate for many apps
- Add features to data model
  - Set-valued attributes
  - Record/tuple references (and the "cascaded dot" notation)
    - "Inverse" references
  - Inheritance, single or multiple
  - Etc.
- Not enough pain at the time—most features could be simulated in relational with some programming and no loss of performance
- Lesson 12: without large performance/functionality advantages, new constructs will go nowhere

19

Object-oriented DB (mid-1980's)

- OO had become the standard programming paradigm
- Impedance mismatch makes writing DB-backed apps difficult
  - Need to translate between DB and PL objects introduces both inconvenience and inefficiency
- Some apps, e.g., CAD (Computer-Aided Design), really want persistent complex objects

20

Starting with PL

- Extend PL (e.g., C++) with DB features to support persistence
  - Data model comes directly from PL, including all its OO features
  -
```
Persistent Part p;
Persistent int i;
i = i+1;
```

Problems
- Absence of leverage: loading/unloading code is gone; so what?
- No standards: different OODBMS were incompatible
- Painful upgrade: all programs have to be relinked
- No PL Esperando: huge chore to add persistence to all PLs
- Unsuccessful in the bigger market of business data processing
  - Query language is lacking or an after-thought
    - Back to CODASYL: programmers wrote algorithms; no optimization
  - Running DB and PL in the same address space raises security concerns

21

O2's different plan

- A French company built on research at INRIA
- A carefully designed OO data model
  - Closer to the semantic data model than to C++
- A high-level, declarative language (OQL)
  - Embedded into the host language
- Could have worked, but "as goes the United States goes the rest of the world"
  - Move to attack US market came too late

Image from http://sctnowonthecampus.files.wordpress.com/2009/07/uncle-sam.jpg 22

Lessons

- Lesson 13: new systems will not sell to users unless they are in "major pain"
- Lesson 14: persistent PL requires the support of the PL community

23

Object-relational (mid-1980's)

- Motivated by the need for new, richer data types (e.g., GIS)
- Extend DB instead of PL
  - User-defined data types (e.g., box)
  - User-defined operators (e.g., box-intersects-box)
  - User-defined functions (e.g., implementing box-intersects-box)
  - User-defined access methods (e.g., R-tree indexing)
- Basic "outer" data type is relation, with extensible data types in the fields
- Relational theory applies to outer operations

24

ORDBMS roots

- Postgres
  - Showed how to build a DBMS engine so new types, functions, etc. could be plugged in
    - ➲ More on this on Thursday
- Sybase
  - Showed that stored procedures were also a good idea for coding application logic (not just operators)
    - Good for both performance and software development (keeping business logic in one place)
- Extensibility and stored procedures have now made it into the SQL standards and most commercial DBMS

Lessons

- Lesson 15: OR has two major benefits: putting code into DB and user-defined access methods
- Lesson 16: widespread adoption requires standards or an elephant pushing hard

Semi-structured data (~2000)

- Conventional DBMS are schema-first
  - Schema defined at DB creation time; difficult to evolve

How about schema-later?

- Application classes
  - Structured data $\rightarrow$ schema-first
  - Structured data with text fields (e.g., forms) $\rightarrow$ schema-first
  - Semi-structured data (e.g., ads, resumes) $\rightarrow$ schema-last
  - Free text $\rightarrow$ schema-not-at-all
- But even ads and resumes are moving to become more structured!
- Semantic heterogeneity remains difficulty to tackle, and schema-first will make matter worse

Semi-structured data example

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

- Semantic heterogeneity
  - Different sets of attributes
  - Same attributes have different meanings/formats
  - Different attributes have same meaning

XML model & language

XML Schema

- Hierarchical data (like IMS)
- References (like CODASYL)
- Set-valued attributes, inheritance, “union” types
- All in all, a major KISS (Keep It Simple, Stupid) violation

XQuery

- Declarative
- But many features are more difficult to work out, e.g.:
  - View support
  - ➲Query optimization, authorization…

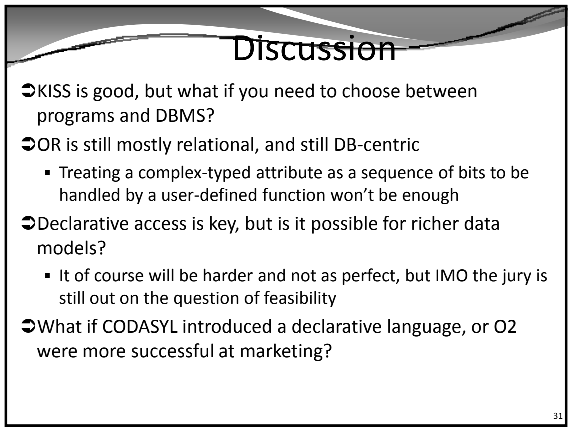
Image from http://whatapic.blogspot.com/2008/05/baby-kiss-pig.html

Lessons & predictions

- Lesson 17: schema-last is probably a niche market
- Lesson 18: XQuery is pretty much OR SQL with a different syntax
- Lesson 19: XML will not solve the semantic heterogeneity either inside or outside the enterprise

Predictions

- XML will become a standard data exchange format
- ORDBMS will be better at handling the next “big thing”
- Elephants will add XML to their ORDBMS
  - ➲True, but IMO it required extensibility beyond what the standard OR features had to offer



Discussion

- ➲KISS is good, but what if you need to choose between programs and DBMS?
- ➲OR is still mostly relational, and still DB-centric
  - Treating a complex-typed attribute as a sequence of bits to be handled by a user-defined function won't be enough
- ➲Declarative access is key, but is it possible for richer data models?
  - It of course will be harder and not as perfect, but IMO the jury is still out on the question of feasibility
- ➲What if CODASYL introduced a declarative language, or O2 were more successful at marketing?

31