Relational Roots

Rachel Pottinger

Goals of the day:

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

Administrative notes (me hat)

- Sample grades have been posted for the optional reading responses
  - Pretty much, if you got a 2, you need more analysis to get a 3 (note that a 2 is still worth 90%)
- I need someone to discussion leading for next Wednesday. If interested see me after class. Note that I grade more leniently on the first several people.

Overview: Two papers (presenter hat)

- E.F. Codd. A Relational Model of Data for Large Shared Data Banks. CACM 13(6), 1970, pp. 377-387
- Donald D. Chamberlin, Morton M. Astrahan, Mike W. Blasgen, Jim Gray, W. Frank King III, Bruce G. Lindsay, Raymond A. Lorie, James W. Mehl, Thomas G. Price, Gianfranco R. Putzolu, Patricia G. Selinger, Mario Schkolnick, Donald R. Slutz, Irving L. Traiger, Bradford W. Wade, Robert A. Yost: A History and Evaluation of System R. CACM 24(10): 632-646 (1981).

Codd paper: (presenter hat)

- *The* paper that introduced relational databases – a real paradigm shift
- Interesting from at least three perspectives:
  - More detailed overview than what I gave ;)
  - Describing the new system – and it's comparison with prior work
  - What was retained, and what changed

The Key Idea: Physical Data Independence

- As stated in overview, not previously true
- Seen through some of the examples:
  - "… existing systems … require … data … stored in at least one total ordering … associated with the hardware-determined ordering of addresses"
  - "can application[s]… remain invariant as indices come and go"? (not always obvious)

Secondary idea: Removing Access Path Dependency

- Previous work had more complicated data structures
  - If in hierarchical model, need to decide on the hierarchy
  - Three problems (at least): (more detail on each coming)
    - Design problem
    - Access path problem
    - Failure when a change in structure is necessary

Design problem

- In the relational model, everything's just a relation
- Don't need to a priori decide how things are related
- Worth noting two exceptions to this:
  - Normalizing
  - Foreign keys and other constraints

Access path problem

- An access path is the way that we actually access the data, i.e., the bits on the disk
- In Codd's relational model, this is just the relations or indices on them
- In previous data models querying required knowing the indexes
- Access was also restricted by the hierarchy of the data

Failure when a change in structure is necessary

- When the structure is changed, this means all applications are obsolete
- This is still somewhat true even with physical data independence
  - Still have to redo all queries (they're just a lot shorter now)

Previous approach discussion (discussion hat)

Pick a group and discuss:

- Design problem (everything's just a relation): Given that there's still normalizing and other constraints, how much of a win is it to not have to tackle these issues immediately?
- Access path (access to data): How much of the access path problem is solved by gaining physical data independence? I.e., would having a hierarchy hurt? If so, how much?
- Structure change: Was it possible that the success of relational databases killed off any interest in making tree-structured data easier to work with?

Relational view:

- A *mathematical* relation
  - Sets rather than bags
- Table only viewed as a vehicle for exposition
- Could have multiple attributes with same name (domain)
  - Necessitates more complicated "relationships" in model

Discussion of overall view of relational model: (discussion hat)

- "The term relation is used here in its accepted mathematical sense". Bags are necessary for some apps, but "set" semantics is often used by those trying to make a first stab at a topic. Does this make sense?

Normal form (presenter hat)

- Key idea: not quite the current notion of normal form – goal is to rid "non-simple domains" – implied hierarchy
- Now:
  - No longer have the same notion of simple domains; just have simple foreign keys
  - This kind of normalization comes for free with ER → Relational translation
  - Lots of other normal forms considered in 70s

Operators

Goal for *designers* not *users*

- Permutation: permute the order of the columns, (for performance?). (discussion hat) Why would this be relevant? Is it just a holdover from mathematics?
- Projection: same as today
- Join: same as today. (discussion hat) today we usually describe as a cross product followed by selection. He describes it straight out. Why?
- Composition & Restriction: basically combinations of projection and join

Key point: some things he got, some things he didn't.

Summary of Codd's paper

- The introduction of relational databases
- Total paradigm change
- Still using not only concepts but terms
- Some things he got wrong (chiefly query language)
- Worth noting, it's in CACM

Ending discussions (discussion hat)

- As noted by many on WebCT, there is a lot that's left to the reader. There is no implementation or evaluation. Thus it would be rejected from almost every conference today. What does this say about our metrics today? What does this say about chances for paradigm change?

System R

- Basically started where Codd's paper left off
- Major research system that pioneered relational databases including:
  - SQL (not covered)
  - Query optimization (up next)
  - Done at IBM San Jose (now Almaden)
- Was one of two first real database systems
  - Other was Ingres from Berkeley
- Many other papers gave deep evaluation; this is just a summary

A brief over view of their goals

- High level interface
- Support different uses, e.g., pre-programmed queries, reports, and ad-hoc
- Allow changing database (e.g., tables and views) without stopping system
- Allow many users
- Recovery
- Allow different views (query and updates)
- Achieve speed of previous systems

Discussion hat: how impressive was this?

Three phases of the project

- Phase zero: prototype
- Phase one: re-design
- Phase two: evaluation in usage

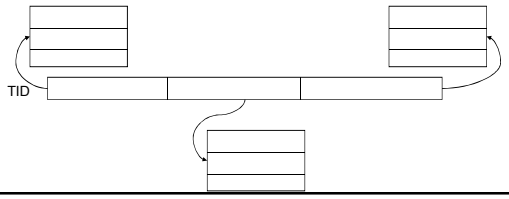
Phase zero

- Always planned to throw one away

  Discussion hat: Now common systems maxim, what are pluses and minuses?
- Space of problem:
  - Only single user
  - No joins!

Underlying system XRM

- Don't worry about the details
- Main points
  - assumed unique (separate) tuple id
  - No data actually in base tuple (inversions)
  - Worked horribly!



Hardest part: optimizing queries

- Much more on this next week
- One key point: original cost model was # of tuples fetched. Discovered not main factor.

Phase One: multiple users

- First up, ditch the storage system (XRM) move to a new one (RSS)
- Have a locking sub-system that "ensures that each data value is accessed, by only one user at a time". (Discussion hat) Does this allow enough concurrency? Is it restrictive enough? Goal, talk things over
- Allowed querying from both PL/I and Cobol

Compilation

- Includes parsing and checking validity – nowadays, never talked about
- They followed their previous work and changed the cost model to minimize I/Os
- More on optimization next week

Join Methods (still used today)

- Nested loops:
  - Scan over a qualifying row in table A. For each row, fetch matching row of table B
  - Greatly speeded if index on table B
- Sort-Merge
  - Sort table A. Sort Table B. Merge using matching values
  - Key advantage: when you're done, it's sorted
- More on these when we get to evaluation

Join Discussion

- Only appreciated joins after user study. Why is this a surprise, especially because Codd's paper listed them

Security model

- Very limited

Recovery & locking

- Media failure discussion
  - Nowadays usually handled by RAID. We won't go into this
- Locking: same notion as today, though exact lock types are different
- They locked predicates, not the same as our locking today

Phase Two (evaluation)

- Generally good
- Interesting to look at what is implied for prior systems:
- "several user sites reported that they were able to install the system, design and load a database, and put some application programs within a matter of days" Discussion: would this fly now? Why or why not?

Discussion:

- "User sites also reported that it was possible to tune the system performance after data was loaded by creating and dropping indexes [sic] without impacting end users or application programs" Hard to imagine this is a surprise. What does it mean about impact of this work?

SQL

- SQL generally successful
  - Major point, since not part of Codd's model
- One advantage cited: only need one language for different contexts – applications, ad hoc, and declaring views
  - Huge, huge win
- SQL creep begins
  - "exists"
  - "like"

Discussion:

- Users requested ability to submit "statement repeatedly for different data values without re-invoking the optimizer". Interesting and odd request. Do you think it's still relevant today? Why or why not?

Security lessons

- Wanted a "group" of users. This is now standard practice
- Note that they comment that they get rid of the notion of "shadow pages" and just use a log. This is what is typically done.

Summary of System R

- One (of 2) first real implementations of relational model
- Great methodology, huge amount of progress
- Many things they got right
- A few things they got wrong

Overall discussion:

- Did the papers expose any big changes between the Codd invention and the System R discussion? If so, what?

Meta comments (me hat)

- Discussion
  - Don't leave it until the very end of a paper, but can be batched or not
  - Can be related to both papers
- Didn't discuss all details – even left some big chunks out
  - I'll give you a list of things to be sure not to skip