Extensibility, OR-Style

CPS 296.1

Database and Programming Languages: Crossing the Chasm

Jun Yang
Duke University
January 21, 2010

† Thanks to contents/ideas borrowed from Hellerstein (http://redbook.cs.berkeley.edu/redbook3/lecs.html)

Image from http://www.codeproject.com/KB/cs/MEFIntro.aspx?msg=2835480

Announcements

- You will hear from me via email tonight regarding discussion leader assignments
- For next Tuesday
  - 2 papers on roots and history of OODBMS
  - 1 review required
- For next Thursday
  - 1 paper about the experience of making a persistent PL
  - Review required

Overview

- Stonebraker & Kemnitz. "The Postgres Next-Generation Database Management System." *CACM*, 1991
  - 1986-1994
  - Overview of one of the first DBMS supporting OO & extensibility
  - Many radical ideas
    - ➲ Some now standard, some yet to come of age
- Stonebraker. "Inclusion of New Type in Relational Data Base Systems." *ICDE* 1986
  - What it *really* takes to add a new type
    - ➲ Much more than adding just a declaration!

➲Just how far can we push the Postgres-style extensibility?

Motivation

- "Pure" relational systems was too painful to use for non-administrative data-intensive apps in the early 1980's
  - CAD/CAM, CASE, GIS, etc.

➲Ideas/hypotheses

- Relational "outer shell" + inheritance + collection-typed, reference-typed, and UDT (User-Defined Types) attributes suffice for modeling?
- UDF (User-Defined Functions)/operators suffice for language?
- Views: code as virtual data
- Fast path to DBMS internals for performance
- Rules system to make databases "active"
- No-overwrite storage + time travel

Type system

- Table → class
  - Tuple → instance; tuple id → oid
  - Simple resolution of multiple inheritance
- Base types, e.g., *dname=c12*, *floorspace=polygon*
  - UDTs (e.g., *polygon*) can be added—more in the second paper
- Array of base types, e.g., *float[12]*

➲Should be a "type constructor," but is limited here to base types

- Set of references, e.g., *coworkers=EMP*, *hobbies=set*
  - 0 or more pointers (oids) instead of embedded instances
  - "set" allows instances of any class
    - ➲ Not precise enough?

➲Overall, not really arbitrarily nested types

Functions

- C functions
  - Convenient, but opaque, e.g.:
    - *overpaid(EMP) = /* check to see if annual salary > 150K */*

  ➲Why is opaqueness bad?
    - DBMS doesn't know how to optimize (e.g., use index on salary)
- POSTQUEL functions
  - POSTQUEL was the query language used by Postgres
  - Can be optimized as part of the query
- Operators
  - Written in C, but with properties and additional metadata that DBMS can exploit in query processing and optimization
    - More in the second paper

Example POSTQUEL

```
define function neighbors (DEPT) returns DEPT as
retrieve (DEPT.all) where DEPT.floor = $.floor

retrieve (DEPT.name)
where neighbors(DEPT).name = "shoe"
```

➲ "=" and "." can operate on sets

➲ Table/class name is heavily overloaded!

- As type declarations (in functions or create statements), it denotes a set of 0 or more references to instances
- In queries, it denotes an instance variable ranging over the class extent (collection of all its instances)?
  - But not quite; DEPT and DEPT* are different!
- Explicitly declare instance variables to avoid confusion, e.g.:

```
retrieve (DEPT.dname)
where DEPT.floor NOT-IN
  {D.floor from D in DEPT where D.dname != DEPT.dname}
```

7

Recursive queries

```
parent(older, younger)

retrieve* into answer
(parent.older) from a in answer
where parent.younger = "John" or parent.younger = a.older
```

Base case — Recursion step

- Fixed-point semantics
  - Start with an empty answer
  - Evaluate over current answer; make result the new answer
  - Repeat until answer no longer changes

➲ More on this when we talk about Datalog

Image from http://www.filemagazine.com/thecollection/archives/2008/10/fixedpoint_theo.html

8

Discussion on model/language

- Postgres became PostgreSQL
  - SQL has replaced (POST)QUEL (elephants won)
- Array of complex types is finally possible as of v8.3
  - Integration with query language is cool
    - ANY, ALL
    - Unnest : explode_array (UDF)
    - Nest: array_accum (User-Defined Aggregate)
      - Each UDA is specified by 3 functions init, transition, final
    - Need recursion to support truly arbitrary nesting
  - Integration with storage/query optimization remains weak
    - Each array is stored as a chunk of bits, apparently with no shredding or additional indexing

References

- http://www.postgresql.org/about/featurematrix
- http://merlinmoncure.blogspot.com/2007/09/this-will-be-first-in-what-hopefully-be.html
- http://merlinmoncure.blogspot.com/2007/09/one-of-my-favorite-problems-in.html
- http://developer.postgresql.org/pgdocs/postgres/xaggr.html

9

Interaction with host PL

- Fast path: allow app code to call DBMS internal modules
  - Still in separate address spaces though
- One interesting motivation
  - PL cache wants to assign OID before writing objects to DB
- Performance advantage if you know what you are doing

➲ Price to pay for performance?

- Safety
- Data independence

➲ Can you think of a more restrictive alternative?

- Allow client to specify execution plans + limited set of stored procedures

10

Rules

- Event-condition-action rules
  - Events include retrieval and modifications
  - Powerful but messy
- Example: 2 ways to force Joe to earn the same salary as Fred
  - Materialize Joe's salary; when updating Fred's, also update Joe's
    - "Forward chaining" by executing actions
  - Virtualize Joe's salary; when getting it, get Fred's instead
    - "Backward chaining" by rewriting queries
  - And what if there are multiple Freds?

➲ Not terribly high-level or declarative

- Programmers specify how, not what
- Programmers need to choose based on data characteristics and desired semantics

11

Discussion on rules

So how is it done in SQL now?

- Assertions: ideal, but nobody does it because efficient implementation is too hard
  - create assertion joe_and_fred_earn_same as check not exist (select * from EMP e1, EMP e2 where e1.name = 'Joe' and e2.name = 'Fred' and e1.salary <> e2.salary)
- Views: defined as queries over base tables
  - Virtual/materialized decision is orthogonal and starting to be automated by DBMS
  - Updating through views is still tricky
    - Oracle allows customization by INSTEAD OF triggers
- Triggers: just on modification events
  - Different controls (e.g., timing, batching)

12

No-overwrite storage

- Just write a new version of the updated record
  - A "vacuum" process moves old data to a historical database
  - "Time travel" is possible
- Can do without data logging
  - Undo info is already in old versions
  - But must flush updates when committing
    - Stable memory required for performance

➲Since then

- WAL (undo/redo) strikes back after Informix acquisition
- PostgreSQL added redo logging

➲New argument for no-overwrite today?

References

- http://redbook.cs.berkeley.edu/redbook3/lec17.html
- http://oreilly.com/catalog/unixbr/chapter/ch14.html
- http://wiki.postgresql.org/wiki/PostgreSQL_for_Oracle_DBAs
- http://www.postgresql.org/docs/8.0/static/wal.html

13

Performance comparison

- Why was the OODB being compared so fast?
  - Same address space; tight integration with PL
- Fast path performance vs. no fast path
  - Price of physical data independence
- Generic B-trees are slower
  - Price of extensibility

14

Adding new types

- Main point: adding a new type entails more than just declaring it!
  - How can you store/access data of this type efficiently?
  - How can you optimize queries containing functions/operators involving this type?
  - How can you support transaction semantics (concurrency control, recovery) for data of this type?

15

Defining a new type

- Content
  - Specify the amount of space for storage and code for conversion from/to strings for input/output
- Operators
  - For each operator, specify token, operand types, result type, precedence, and implementation code
- Code safety issue
  - Unprotected: same address space as server; fast but risky
  - Protected: different address space: safer but slow
  - Use protected for debugging, unprotected for production

16

Making an AM generic

- AM = access methods, e.g., B-tree
- E.g., what does a B-tree assume about the type it handles?
  - Basically, a totally ordered domain
- In general, each AM needs a template specifying:
  - What ops (signatures) it expects
    - E.g., B-tree requires <=, and <, >, =, >= are optional
  - What properties it expects
    - E.g., totally ordered domain
    - Only as guidance to developers who want to use this AM
    - Difficult to enforce

17

Leveraging a generic AM

- To leverage an AM, a new type need to implement ops required by the AM template
  - E.g.: need a B-tree to store boxes by order of their areas?
    - Implement area-eq(box1, box2), area-gt(box1, box2), etc.

What else?

- Each op provides cardinality/page count estimation formulae
  - Why are these estimates so important?
  - Interpretable by DBMS, and based on
    - Statistics kept by AM: # of tuples (N), # of disk pages, # of (unique) index keys (Ituples), max & min key value (high-key, low-key), etc.
    - Run-time parameter: the constant value in TABLE.ATTR OP value
  - E.g., area-eq(box1, box2): N/Ituples
  - E.g., area-lt(box1, value): (value–low-key)/(high-key–low-ley)*N

➲Limitations of these formulae?

18

Adding a new AM

- E.g., R-tree is needed for high-dimensional indexing
- First, specify the template
  - E.g., R-tree requires contains(T1, T2) and union(T1, T2)
    - ➲ Enough for R-tree?
- Second, implement AM methods (which call the required ops)
  - open/close
  - insert/delete/replace, build (why not repeatedly insert?)
  - get-unique(descriptor, tuple-id)
  - get-first(descriptor, OP, value)
    get-next(descriptor, OP, value, tuple-id)
    - For returning all records satisfying TABLE.ATTR OP value

19

Don't forget transactions

- For logging/recovery
  - Physical logging (of bits on pages) requires no additional work
  - Logical logging requires implementing REDO and UNDO for built-in events (insert/delete/replace records, etc.)
    - Also possible to add AM-specific events
- For concurrency control
  - AM code can call a standard scheduler when it reads and writes
  - More concurrency possible by calling lower-level lock/unlock
    - E.g.: top-down access of a tree-based index allows unlocking the parent after locking the child (and knowing the parent won't be changed later)
      - Default 2-phase locking would allow no concurrent index accesses

20

Query proc/opt

For each new op, specify:

- How to compute selectivities of predicates
  TABLE.ATTR OP value and TABLE1.ATTR1 OP TABLE2.ATTR2
  - Use AM estimation if index is available, e.g.: (1/Ituple) else 1/10
- Whether built-in join methods are applicable
  - Merge-joinable?
  - Hash-joinable?
  - Nested-loop join always possible
  - "Iterative substitution" (indexed nested-loop) possible with index

➲ Limitation?

- Only enables existing query processing algorithms for new ops; what about new algorithms?
- And how to make new algorithms reusable for old and new ops?

21

Discussion

➲ORDBMS is surprisingly accommodating

➲… up to a point; then model/language will turn ugly and performance will suffer

- Most indexing/processing/optimization techniques revolve around tables with (still quite) opaque cells

➲Instead of fitting everything in RDBMS, time has come to take good DBMS ideas and apply them to vertical markets?

- Physical data independence, I/O-efficient algorithms, cost-based optimization, etc.
- Read Stonebraker's "one size fits all" papers!

Image from http://bitsandpieces.us/wp-content/uploads/2009/03/imagesone-20size.jpg

22