Object-Oriented & Object-Relational DBMSs

Module 9, Lecture 3

"You know my methods, Watson. Apply them."
~~ A.Conan Doyle, *The Memoirs of Sherlock Holmes*

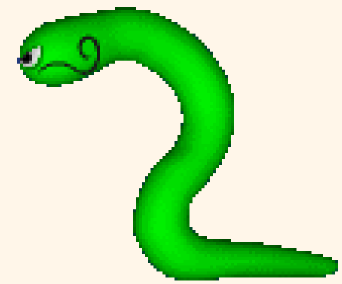
Motivation

- Relational model (70's): Clean and simple.
  - Great for administrative data.
  - Not as good for other kinds of data (e.g., multimedia, networks, CAD).
- Object-Oriented models (80's): Complicated, but some influential ideas.
  - Complex data types.
  - Object identity/references.
  - ADTs (encapsulation, behavior goes with data).
  - Inheritance.
- Idea: Build DBMS based on OO model.

Example App: Asset Management

- Old world: data ***models*** a business
- New world: data IS business
  - 1011010111010100010100111 = $$$$$!
  - Software vendors, entertainment industry, direct-mail marketing, etc.
  - This data is typically more complex in structure than administrative data.
- Emerging apps mix these two worlds.

An Asset Management Scenario


Herbert the Worm

- Dinky Entertainment Corp.
  - Assets: cartoon videos, stills, sounds
  - Herbert films show worldwide
  - Dinky licenses Herbert videos, stills, sounds for various purposes:
    - action figures
    - video games
    - product endorsements
  - DBMS must manage assets and business data

Why not a Standard RDBMS?

```
create table frames (frameno integer, image BLOB,
                              category integer)
```

- Binary Large Objects (BLOBs) can be stored and fetched.
- User-level code must provide all logic for BLOBs.
- Scenario: Client (Machine A) requests "thumbnail" images for all frames in DBMS (Machine B).
  – Inefficient, too hard to express queries.

Solution 1: Object-Oriented DBMS

- Idea: Take an OO language like C++, add persistence & collections.

```
class frame {
  int frameno;
  jpeg *image;
  int category;
}
persistent set <frame *> frames;
foreach (frame *f, frames)
  return f->image->thumbnail();
```

- Shut down the program. Start it up again. Persistent vars (e.g. frames) retain values!

OODBMS, cont.

- New collection types:
  - Type constructors: set<>, bag<>, list<>
  - Iterators to loop through collection types.
- Gives a rudimentary "query language".
  - How to do selection? projection?
  - "join" set<emp *>emps, set<dept *>depts?
  - Can have pointers in this data model, with efficient pointer-based joins.
  - What RDBMS feature is missing here?

OODBMS applications

- OODBMSs good for:
  - complex data
  - fixed set of manipulations (no ad-hoc queries)
  - special-purpose applications written by hackers
- Problems:
  - no query support
  - application bugs trash persistent data
  - security problems: no protection w/in a page!
  - schema evolution very difficult
  - some argue it's back to the network data model
- A modest success in the marketplace

Solution 2: Object-Relational

- Idea: Add OO features to the type system of SQL. I.e. "plain old SQL", but...
  - columns can be of new types (ADTs)
  - user-defined methods on ADTs
  - columns can be of complex types
  - reference types and "deref"
  - inheritance and collection inheritance
  - old SQL schemas **still work!** (backwards compatibility)
- Relational vendors all moving this way (SQL3). Big business!

An Example ORDBMS Schema

ADTs

```
create table frames (frameno integer, image jpeg,
   category integer);
create table categories (cid integer, name text,
   lease_price float, comments text);
create type theater_t row (tno integer, name
   text, address text, phone integer)
create table theaters theater_t;
create table nowshowing (film integer, theater
   ref(theater_t), start date, end date);
create table films (filmno integer, title text,
   stars setof(text), director text, budget
   float);
create table countries (name text, boundary
   polygon, population integer, language text)
```

complex types

reference types

Complex Types

- User can use type constructors to generate new types:
  - setof(foo)
  - arrayof(foo)
  - listof(foo)
  - row (n1 t1, ..., nk tk)
- Can be nested:
  - setof(arrayof(int))

ADTs: User-Defined Atomic Types

- Built-in SQL types (int, float, text, etc.) are limited.
  - Even these types have simple ***methods*** associated with them (math, LIKE, etc.)
- ORDBMS: User can define new atomic types (& methods) if a type cannot be naturally defined in terms of the built-in types:

```
create type jpeg (internallength = variable,
  input = jpeg_in, output = jpeg_out);
```

- Need input & output methods for types.
  - e.g., Convert from text to internal type and back.

Reference Types & Deref.

- In most ORDBMS, every object has an OID.
- So, can "point" to objects -- reference types!
  - ref(theater_t)
- Don't confuse reference and complex types!
  - **`mytheater row(tno integer, name text, address text, phone integer)`**
  - **`theater ref(theater_t)`**
- Both look same at output, but are *different!!*
  - Deletion, update, "sharing"
  - Similar to "by value" vs. "by reference" in PL

Dinkey Schema Revisited

```
create table frames (frameno integer, image jpeg,
  category integer); -- images from films
create table categories (cid integer, name text,
  lease_price float, comments text); -- pricing
create type theater_t tuple(tno integer, name
  text, address text, phone integer)
create table theaters theater_t; -- theaters
create table films (filmno integer, title text,
  stars setof(text), director text, budget
  float); -- Dinkey films
create table nowshowing (film integer, theater
  ref(theater_t), start date, end date);
create table countries (name text, boundary
  polygon, population integer, language text)
```

An Example Query in SQL-3

- Clog cereal wants to license an image of Herbert in front of a sunrise:

```
select F.frameno, thumbnail(F.image),
       C.lease_price
  from frames F, categories C
 where F.category = C.cid
   and Sunrise(F.image)
   and Herbert(F.image);
```

  - The thumbnail method produces a small image.
  - The Sunrise method returns T iff there’s a sunrise in the picture.
  - The Herbert method returns T iff Herbert’s in pic.

Another SQL-3 Example

- Find theaters showing Herbert films within 100 km of Andorra:

```
select N.theater->name, N.theater->address, F.name
  from nowshowing N, frames F, countries C
 where N.film = F.filmno
   and Radius(N.theater->location, 100) || C.boundary
   and C.name = 'Andorra'
   and F.stars ∋ 'Herbert the Worm'
```

  – theater attribute of nowshowing: ref to an object in another table. Use -> as shorthand for deref(theater).name
  – Set-valued attributes get compared using set methods.

Example 2, cont.

```
select N.theater->name, n.theater->address, F.name
  from nowshowing N, frames F, countries C
 where N.film = F.filmno
   and Radius(N.theater->location, 100) || C.boundary
   and C.name = 'Andorra'
   and F.stars ∋ 'Herbert the Worm'
```

- join of N and C is complicated!
  - Radius returns a circle of radius 100 centered at location
  - || operator tests circle,polygon for spatial overlap

New features in SQL-3 DML

- Built-in ops for complex types
  - e.g. the typical set methods, array indexing, etc.
  - dot notation for tuple types
- Operators for reference types
  - deref(foo)
  - shorthand for deref(foo).bar: foo->bar.
- User-defined methods for ADTs.
- Syntax has not been completely decided yet

Path Expressions

- Can have nested row types (Emp.spouse.name)
- Can have ref types and row types combined
  - nested dots & arrows. (Emp->Dept->Mgr.name)
- Generally, called path expressions
  - Describe a "path" to the data
- Path-expression queries can often be rewritten as joins. Why is that a good idea?

```
select E->Dept->Mgr.name
  from emp E;
```

```
select M.name
  from emp E, Dept D, Emp M
 where E.Dept = D.oid
   and D.Mgr = M.oid;
```

- What about Emp.children.hobbies?

User-Defined Methods

- New ADTs will need methods to manipulate them:
  - e.g., for jpeg images: thumbnail, crop, rotate, smooth, etc.
  - Expert user writes these methods in a language like C and compiles them.
  - Methods must be registered with ORDBMS, which then dynamically links the functions into server.

```
create function thumbnail(jpeg) returns jpeg
    as external name '/a/b/c/Dinkey.o'
```

Inheritance

- As in C++, useful to "specialize" types:

```
create type theatercafe_t under
                theater_t (menu text);
```

- Methods on theater_t also apply to its subtypes.
  – Can redefine some of these methods.
  – Can define additional methods.

Inheritance

- "Collection hierarchies": Inheritance on tables
  - `create table student_emp under emp (gpa float);`
  - Queries on emp also return tuples from student_emp (unless you say "emp only")
- "Type extents":
  - All objects of a given type can be selected from a single view (e.g., select * from theater_t)

Modifications to support ORDBMS

- Parsing
  - Type-checking for methods pretty complex.
- Query Rewriting
  - Often useful to turn path exprs into joins!
  - Collection hierarchies → Unions
- Optimization
  - New algebra operators needed for complex types.
    - Must know how to integrate them into optimization.
  - WHERE clause exprs can be expensive!
    - Selection pushdown may be a bad idea.

Modifications (Contd.)

- Execution
  - New algebra operators for complex types.
  - OID generation & reference handling.
  - Dynamic linking.
  - Support "untrusted" methods.
  - Support objects bigger than 1 page.
  - Method caching: much like grouping.
    - f(x) for each x is like AVG(major) for each major.

Modifications (Contd.)

- Access Methods
  - Indexes on methods, not just columns.
  - Indexes over collection hierarchies.
  - Need indexes for new WHERE clause exprs (not just <, >, =)!
    - GiST can help here.
- Data Layout
  - Clustering of nested objects.
  - Chunking of arrays.

Stonebraker's Application Matrix

	No Query	Query
Complex Data	OODBMS	ORDBMS
Simple Data	File System	RDBMS

Thesis: Most applications will move to the upper right.

OO/OR-DBMS Summary

- Traditional SQL is too limited for new apps.
- OODBMS: Persistent OO programming.
  - Difficult to use, no query language.
- ORDBMS: Best (?) of both worlds:
  - Catching on in industry and applications.
  - Pretty easy for SQL folks to pick up.
  - Still has growing pains (SQL-3 standard still a moving target).

Summary (Contd.)

- ORDBMS offers many new features.
  - But not clear how to use them!
  - Schema design techniques not well understood
  - Query processing techniques still in research phase.
    - A moving target for OR DBA's!
- Prediction: You will use an ORDBMS in the future.