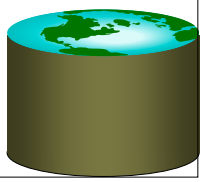
Object-Oriented & Object-Relational DBMS

R & G (& H) Chapter 23

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



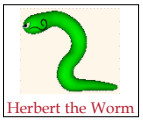
Motivation

- **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**
  - 10110101110101000101001111 = $$$$$!
  - software vendors, entertainment industry, direct-mail marketing, etc...
  - this data is typically more complex than administrative data
- **Emerging apps mix these two worlds.**

An Asset Management Scenario


Herbert the Worm

- **Dinkey Entertainment Corp.**
  - assets: cartoon videos, stills, sounds
  - Herbert films show worldwide
  - Dinkey licenses Herbert videos, stills, sounds for various purposes
    - action figures
    - video games
    - product endorsements
  - database 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**
- **Performance**
  - Scenario: client (Machine A) requests "thumbnail" images for all frames in DBMS (Machine B)
  - Should move *code to data*, don't move data to code!
- **Inefficient, too hard to express queries.**

"Object-Relational" Databases

- **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 (SQL:1999).**
- **Postgres group invented a lot of this stuff at Berkeley**
  - And had it working in the early 90's!
  - Unfortunately, it defined its own syntax before the standard
    - And now is not standard-compliant
    - Most of this stuff can be done in Postgres with analogous syntax
    - And Postgres has more extra goodies here than SQL:99!

Some History

- **In the 1980's and 90's, DB researchers recognized benefits of objects. Two research thrusts:**
  - OODBMS: extend C++ with transactionally persistent objects
  - ORDBMS: extend Relational DBs with object features
- **Postgres was a Berkeley research project, defined ORDBMSs. Postgres "beat" OODBMSs.**
  - Was commercialized as Illustra
  - Informix (a relational vendor) bought Illustra and integrated the ORDBMS features into Informix' core server
  - Oracle and IBM were forced to compete with Informix
  - The OODBMS companies never caught on
  - SQL:1999 standard included many features invented in the Postquel language
- **The Postgres research project went "open source" in 95**
  - Some Berkeley folks converted from Postquel to an extended SQL
  - The open source community took the code and ran with it
- **IBM bought Informix a couple years ago**
  - Hence sells 2 of the 3 leading ORDBMS implementations!

An Example SQL:1999 Schema

ADTs

reference types

complex types

```
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 as row (tno integer, name text,
   address text, phone integer) ref is system generated;
create table theaters of theater_t ref is tid system
   generated;
create table nowshowing (film integer, theater
   ref(theater_t) scope theaters, start date, end date);
create table films (filmno integer, title text, stars
   varchar(25) array[10], director text, budget float);
create table countries (name text, boundary polygon,
   population integer, language text)
```

Complex Types

- **use type constructors to generate new types**
  - row (n1 t1, ..., nk tk)
  - *base* array [i]
- **can be nested:**
  - row(filmno integer, stars varchar(25) array [10])
- **Other obvious extensions:**
  - listof(base)
  - setof(base)
  - bagof(base)
  - Not in the SQL:1999 standard. Postgres supports setof, Informix (commercialized Postgres) supports setof, bagof, listof.

ADTs: User-Defined Atomic Types

- **Built-in SQL types (int, float, text, etc.) limited**
  - have simple *methods* as well (math, LIKE, etc.)
- **ORDBMS: can define new types (& methods)**

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

- **Not naturally composed of built-in types**
  - new *atomic* types
- **Need input & output methods for types**
  - convert from text to internal type and back
  - we'll see how to do method definition soon...

Reference Types & Deref.

- **In ORDBMS, objects can be given object IDs (OIDs)**
  - Unique across time and space
  - `create table theaters of theater_t ref is tid system generated;`
  - Some systems do this for all rows of all tables
- **So, can "point" to objects -- reference types!**
  - ref(theater_t) scope theaters
- **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 as row(tno integer, name text,
   address text, phone integer) ref is system generated;
create table theaters of theater_t ref is tid system
   generated; -- theaters
create table films (filmno integer, title text, stars
   varchar(25) array[10], director text, budget float);
   -- Dinkey films
create table nowshowing (film integer, theater
   ref(theater_t) scope theaters, start date, end date);
create table countries (name text, boundary polygon,
   population integer, language text)
```

An Example Queries in SQL-99

- **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 is_Sunrise(F.image)
   and is_Herbert(F.image);
```

  - The thumbnail method produces a small image
  - The is_Sunrise method returns T iff there's a sunrise in the pic
  - The is_Herbert method returns T iff Herbert's in pic

Another SQL-99 Example

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

```
select N.theater->name, N.theater->address, F.title
  from nowshowing N, frames F, countries C
 where N.film = F.filmno
   and Overlaps(Radius(N.theater->location, 100),
                C.boundary)
   and C.name = 'Andorra'
   and `Herbert the Worm' = F.stars[1]
```

  - theater attribute of nowshowing: ref to an object in another table. Use -> as shorthand for deref(theater).name
  - Array index as in C or Java

Example 2, cont.

```
select N.theater->name, N.theater->address, F.title
  from nowshowing N, frames F, countries C
 where N.film = F.filmno
   and Overlaps(Radius(N.theater->location, 100),
                C.boundary)
   and C.name = 'Andorra'
   and `Herbert the Worm' = F.stars[1]
```

- **join of N and C is complicated!**
  - Radius returns a circle of radius 100 centered at location
  - Overlaps compares a circle, polygon for spatial overlap

New features in SQL-99 DML

- **Built-in ops for complex types**
  - e.g. array indexing, dot notation for row types
- **Operators for reference types**
  - deref(foo)
  - shorthand for deref(foo).bar: foo->bar.
- **User-defined methods for ADTs**
- **Additional vendor-specific syntax**
  - For stuff like setof, bagof, listof…
  - E.g. typical set operators

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?**
  - Analogy to XML trees

User-Defined Methods

- **New ADTs will need methods to manipulate them**
  - e.g. for jpeg: thumbnail, crop, rotate, smooth, etc.
  - expert user writes these methods in a language like C, compiles them
  - register methods with ORDBMS:

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

  - Most ORDBMS bundle a JVM
  - C functions can be dynamically linked in

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
- **"Collection hierarchies": inheritance on tables**
  - `create table theater_cafes of type theater_t under theaters;`
  - queries on theaters also return tuples from theater_cafes (unless you say "theaters only")
- **"Type extents"**
  - all objects of a given type can be selected from a single view (e.g., select * from theater_t)
  - Not supported in SQL99

User Defined Aggregates

- **May want to define custom aggregates**
  - For standard types
    - E.g. RunnerUp instead of MAX
  - For new ADTs
    - E.g. ColorHistogram over jpegs
- **An aggregate is actually a triplet of 3 user-defined helper functions**
  - Initialize: generate a transition value
  - Advance: incorporate a new input value into the transition value
  - Finalize: convert transition value into an output value
- **Note that the DBMS need not understand the types of the running state, nor the behavior of the functions!**

Modifications to support all this?

- **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!
    - select pushdown may be a bad idea

More modifications

- **Execution**
  - new algebra operators for complex types
  - OID generation & reference handling
  - JVMs and/or dynamic linking
  - support "untrusted" C 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, cont.

- **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.
    - http://gist.cs.berkeley.edu
    - GiST indexes implemented in Postgres, Informix
- **Data Layout**
  - clustering of nested objects
  - chunking of arrays

An Alternative: OODBMS

- **Persistent OO programming**
  - Imagine declaring a Java object to be "persistent"
  - Everything reachable from that object will also be persistent
  - You then write plain old Java code, and all changes to the persistent objects are stored in a database
  - When you run the program again, those persistent objects have the same values they used to have!
- **Solves the "impedance mismatch" between programming languages and query languages**
  - E.g. converting between Java and SQL types, handling rowsets, etc.
  - But this programming style doesn't support *declarative* queries
    - For this reason (??), OODBMSs haven't proven popular
- **OQL: A declarative language for OODBMSs**
  - Was only implemented by one vendor in France (Altair)
  - XQuery is the revenge of OQL!

Summary, cont.

- **ORDBMS offers many new features**
  - but not clear how to use them!
  - schema design techniques not well understood
    - No good logical design theory for non-1st-normal-form!
  - query processing techniques still in research phase
    - a moving target for OR DBA's!
  - XML is an alternative for complex object features
    - The equivalences between SQL's complex object support and its (future) XQuery integration are not well explored
    - This redundant functionality "happened to" SQL, don't expect it to make sense!