Object-Relational DBMS

Wei Hong, Ph.D.
Intel Research, Berkeley
whong@intel-research.net

"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**

Object-Oriented Databases

- **Initial Idea: make (C++) objects persistent**
  - Good for "pointer chasing" type of apps (e.g., CAD, CAM), niche market
  - Big paradigm shift from relational databases
  - Players: Objectivity, Object Design, Versant, etc.
- **Evolution: towards Object-Relational**
  - Added limited SQL support
  - Embracing Java and XML

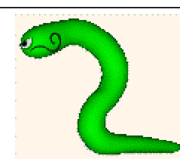
"Object-Relational" Databases

- **Idea: add OO features to the type system of SQL, i.e. "plain old SQL", but...**
  - columns can be of new atomic types (ADTs)
  - columns and rows can be of complex types
  - user-defined methods on new types
  - object identity, reference types and "deref"
  - type inheritance
  - old SQL schemas **still work!** (backwards compatibility)
- **Evolution:**
  - All major relational vendors have evolved their RDBMS into ORDBMS.
  - SQL-99 is the current standard, but not nearly as well adopted as SQL-92.
- **Postgres:**
  - one of the first ORDBMS prototypes, turned into Illustra, then Informix, now IBM.
  - PostgreSQL: an open-source ORDBMS at your finger tips!

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 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): collection of bits that can be stored and fetched like a file**
- **App code must provide logic to interpret the bits, e.g., colors of an image**
- **Hard for code sharing**
- **Poor Performance**
  - Scenario: client (Machine A) requests images for all frames in DBMS (Machine B)

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 of type theater_t;
create table nowshowing (film integer, theater
  ref(theater_t), start date, end date);
create table films (filmno integer, title text,
  stars set(text), director text, budget float);
create table countries (name text, boundary
  polygon, population integer, language text)
```

structured types

ADTs: User-Defined Atomic Types

- **Basic SQL types (int, varchar, etc.): builtin atomic types**
  - builtin *methods*, e.g., math, comparison, etc.
- **ORDBMS: can define new types (& methods)**

```
create type jpeg (internallength = variable,
  input = jpeg_in, output = jpeg_out);
Create type point (internallength = 16, input =
  point_in, output = point_out);
```

- **Not naturally composed of built-in types**
  - new *atomic* types
- **Required parameters for new ADT**
  - Internallength
  - Input/output: convert from/to string
- **Optional Parameters**
  - Alignment
  - Send/receive: convert to/from wire format
  - Etc.

User-Defined Methods

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

```
create function Herbert(jpeg) returns boolean
as external name '/a/b/Dinkey.so' language C;
create function thumbnail(jpeg) returns jpeg
as external name '/a/b/Dinkey.so' language C
trusted not variant;
```

- **Elements of a user-defined function**
  - Name, argument types and return type
  - Implementation and language
  - Attributes, e.g., trusted, iscachable, handles_null, etc.

User-Defined Methods, cont

- **C Functions**
  - ORDBMS dynamically links functions into server at run time
  - Must use specific ORDBMS server programming API
    - Access to run-time states, e.g., argument types
    - Access to system resources, e.g., memory
    - Access to database: query interface
  - High performance, but
    - Tricky to write: thread safety, resource management, exception handling, interrupts, etc.
    - Security concerns
  - Tend to be built by DBMS developers themselves: DataBlades, DataCartridges, Extenders, etc.
- **SQL Functions**

```
create function ConvertCurrency(float, text) returns float
   as ‘select $1 * exchange_ratio from CurrencyExchange
   where country_name = $2’ language SQL;
```

- **Other languages: JAVA, PERL, TCL, proprietary stored procedure languages (e.g., PLSQL)**

User-defined Operators

- **Shorthand for function calls: x = y is equivalent to Equal(x, y)**
- **Some systems let you modify the operator-to-function bindings, e.g.,**

```
create operator || (procedure = overlap)
```

- **Attributes for the optimizer**
  - Commutator
  - Negator
  - Selectivity estimator
  - Hashable, sortable?

User-defined Aggregates

- **Aggregates beyond min, max, sum, avg, count, e.g., ThirdLargest**
- **Aggregates on new types, e.g., polygon**
- **Aggregation framework: state init, state transition, finalize**

```
create aggregate name (BASETYPE =
  input_data_type, SFUNC = sfunc, STYPE =
  state_type, [, FINALFUNC = ffunc] [,
  INITCOND = initial_condition]);
```

- **Avg: state is count and sum initialized to 0, state transition is increment count, add to sum, finalize by dividing sum with count.**

Distinct Types

- **Clone an existing type and all its methods, overload methods**
- **Example:**

```
create distinct type Price as float; -- simply
  for strong typing
create distinct type BerkeleyTime as Time;
create function IsLate(BerkeleyTime) returns
  boolean as 'select curtime() > $1 + '10
  minutes'' language SQL;
```

- **Don't develop a brand new type unless you have to!**

Structured Types

- **use type constructors to generate new types**
- **Collection types**
  - set(T): multiset
  - array(T), T[][]
  - list(T)
- **Row types (composite type)**
  - row (Col_1 T_1, ..., Col_k T_k)
  - Named row type, e.g., theater_t
- **Reference Types**
  - Ref(T)
- **All first-class types!**

Collection Types

- ***IN* operator: elem *IN* collection**
- **Collection type expressions can be used in *FROM* clause (table expressions)**

```
create function Theaters(date) returns
  SET(theater_t) …;
select name from Theaters(curdate()) where
  address like '%Berkeley%';
```

- **Subqueries are of SET type**

```
HerbertFight((select * from frames where
  Herbert(images)))
```

- **Array and List: ordered, access elements by index**

Row Type

- **Dot opeartor: theater.address**
- **Nested dot notation: theater.address.zipcode**
- **Ambiguity with schema.table.column**
- **Backward compatibility is higher priority**

Reference Types

- **In most ORDBMS, every object has an OID**
- **So, can "point" to objects -- reference types!**
  - ref(theater_t)
- **Don't confuse reference and row types!**
  - `mytheater row(tno integer, name text, address text, phone integer)`
  - `theater ref(theater_t)`
- **"by value" v.s. "by reference"**
- **Deref: deref(theater) returns a theater_t row, theater->name is shorthand for deref(theater).name**
- **Referential integrity**
  - ORDBMS may not enforce it!

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 of type theater_t;
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)
```

More Example Queries

- **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 pic
  - The Herbert method returns T iff Herbert's in pic

Another 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 `Herbert the Worm’ IN F.stars
```

  - 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 ‘Herbert the Worm’ IN F.stars
```

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

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?**

Inheritance

- **As in C++, useful to "specialize" types:**
  - `create type theatercafe_t under theater (menu set(row(item text, price Price)));`
  - methods on theater_t also apply to its subtypes
- **"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)

Popular Extensions to ORDBMS

- **Spatial (come with Postgres)**
  - Point, polygon, circle, etc.
  - Overlap, contain, etc.
  - Spatial index, e.g., R-tree
- **Multimedia**
  - Text, image, video
  - Text search, image/video manipulation and search
- **TimeSeries**
  - Timestamped arrays of row types
  - Clip, merge, moving averages, etc.

Summary, cont.

- **Tips on how to use Object-Relation features**
  - Think beyond alpha-numeric types
  - Push data logic into DBMS
  - Leverage existing or prepackaged types and methods, e.g., DataBlades, Cartridges, Extenders.
  - Modify behavior through distinct types or inheritance
  - Complex types are a double-edged sword. Use caution!
- **Watch out for XML data models (XML Schema, XQuery)!!**