
OODBMS and ORDBMS

16 May 2010
Lecture 9b

Topics for Today

- Introduction to Object-Database Systems
- Motivating example
 - Dinky
- Source RG 23.1-23.3

Introduction

- So far we have seen relational databases
- They can do custom types
 - Restrictions on atomic types
 - Constraints
 - Documentation

Review: Domains, Distinct Types

- CREATE DOMAIN ratingval INTEGER DEFAULT 1
 - CHECK (VALUE >= 1 AND VALUE <= 10)
- CREATE TABLE (sid int primary key, rating ratingval)
- CREATE TYPE ratingtype AS INTEGER

Objects?

- In this unit we'll introduce custom types much more like objects in classical programming languages
 - Collections
 - Methods
 - References
- We'll talk about Benefits and Drawbacks
 - What do we want to gain?
 - What do we stand to lose?

What do we want?

- To enrich the types that the database can store
- To get around the awkwardness of the relational model when dealing with collections
- To define types which can have custom behavior

What might we lose?

- Normalization?
- Speed?
- Interoperability?
- Standardization?

Approaches

- The book take a generic approach based on **SQL:1999**
 - The latest version is SQL:2008
 - There aren't any commercially available DBMS' which support it fully
- We'll do targilim on MS SQL Server
 - They have a slightly different approach than the book
 - A bit more, a bit less
 - This is not meant as an endorsement of MS SQL – it's just what we have available

Why Objects?

- Databases aren't just for simple data anymore
 - Web sites use them
 - Online stores keep graphics for their catalogs
 - An online music store keeps MP3s for search and download
- What do we do with things that are not integers, floats, or datetimes?
 - Binary Large Object (BLOB)
 - Character Large Object (CLOB)
- These are kept off disk and are not interpreted by the database at all
 - The application must figure out what to do with them

Objects and First Class Citizens

- Object-Database Systems break those limitations to make objects into first class citizens of the database
 - They can do anything that an atomic type can do
 - The user is able to define custom behaviors for them
- We can do WHERE, Order By on them
- We can build and tear them down on the fly

OODBMS and ORDBMS

- There are two major approaches
- Object Oriented Database Management Systems
 - Pure objects
 - No more tables, just store objects
 - Everything is done with custom code
- Object-Relational Database Management Systems
 - Relations + Objects in specific fields
 - We keep the tables, but allow custom types to appear in the columns
 - Everything that is not an object behaves normally
- ORDBMS has won out – Oracle, MS, IBM all support it

What we'll cover

- The book presents two kinds of examples:
 - Ones based on SQL:1999 – we'll work on those closer
 - Fictitious examples which should appear in the next releases of SQL
- Another approach is using the JDBC driver
 - Interested? Read Chapter 6

So Far

- Introduction to Object-Database Systems
- Motivating example
 - Dinky

Motivating Example

- A multimedia based database
- It stores the usual numbers, strings, dates. In addition times also is used to catalog large quantities of media.
 - Media can include images, sounds, music, videos, and movies
- Database is used to manage the various media that are sold or licensed.
- Goal is to allow queries which can make use of structure in media objects it stores, not just store and retrieve.

New Data Types

Three goals:

1. There is a need to define **user-defines types** with user defined behavior, which is akin to writing C# classes with methods representing behaviors performed on or by object.
2. Use of **inheritance** to allow for commonality between types.
 - Makes sense in the multimedia case
 - In other circumstances inheritance doesn't make sense
3. We need to introduce the notion of **pointer** - **object ids** in the book's parlance, to make storage of large data objects more reasonable
 - Akin to references which allow to more efficient store large objects without putting them in memory

BLOB

- Method of storage of large objects in the relational model is called *Binary Large Objects* (BLOB)
 - BLOB is not interpreted by DBMS, just stores and retrieved
 - The requesting application is responsible for interpreting the contents
 - BLOBs can't be filtered, just selected based on criteria
- Advantages of BLOBs:
 - Can be efficiently stores and retrieved, but aren't first class citizens
- Disadvantages
 - Can't be filtered
 - Don't have any meaning, so there is no metadata which DBMS might be able to use

Inheritance?

- Inheritance and structured types are simply not supported at all in the relational model
- We want to get this functionality out of the ORDBMS and we will study an approach for doing so in the next few lectures

Sample Schema

1. CREATE TABLE Frames (*frameno* integer, *image* jpeg_image, *category* integer);
2. CREATE TABLE Categories (*cid* integer, *name* text, *lease_price* float, *comments* text);
3. CREATE TYPE theater_t AS ROW(*tno* integer, *name* text, *address* text, *phone* text) REF IS SYSTEM GENERATED;
4. CREATE TABLE Theaters OF theater_t REF is tid SYSTEM GENERATED;
5. CREATE TABLE Nowshowing (*film* integer, *theater* REF(theater_t) SCOPE Theaters, *start* date, *end* date);
6. CREATE TABLE Films (*filmno* integer, *title* text, *stars* VARCHAR(25) ARRAY [10], *director* text, *budget* float);
7. CREATE TABLE Countries (*name* text, *boundary* polygon, *population* integer, *language* text);

Figure 9.1: SQL:1999 DDL Statements for Dinky Schema

Manipulating the New Data

- Once data items are worked out, we need to think how to use them
- With the custom data types we can define custom user functions to manipulate the data
- The database should be able to run the functions on the objects in the columns and return values based on them

Query Example 1

```
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)
```

Query Example 1

- The semantics of the query is very similar to the semantics of a regular query:
 - (a) The `Select` chooses columns from the table
 - (b) The function `thumbnail` performs an operation on the value in a single column (like `+3`)
 - (c) The `Where` examines each row and makes a decision based on the row
 - (d) The functions `is_sunrise` and `is_herbert` examine each row to make a decision

Comparison

- Another option is to implement functions of comparison between values in the rows
- This also requires some functions to perform the comparisons on the custom types

Query Example 2

```
SELECT N.theater->name, N.theater->address, F.title
FROM Nowshowing N, Films F, Countries C
WHERE N.film = F.filmno
      AND C.name = 'Andorra'
      AND overlaps(C.boundary, radius(N.theater->address, 100))
      AND 'Herbert the Worm' = F.Stars[1]
```

What we have so far -

- One caveat from the previous example is that by making an array type we are losing First Normal Form
- To summarize, the above example has shown us examples of the three basic attributes of object-type databases:
 - User-Defined Methods For the objects to examine and convert them
 - Operators for Structured Types For structures which are stored locally in each table
 - Operators for Reference Types For pointers to other tables (like foreign keys)
- We'll talk about how you can implement all of these functions in the next few sections

Conclusion

- Introduction to Object-Database Systems
- Motivating example
 - Dinky