Complex Data and Object-Oriented Databases

• Topics
  • The object-oriented database model (JDO)
  • The object-relational model
  • Implementation challenges

• Learning objectives
  • Explain what an object-oriented data model is.
  • Describe data management problems for which relational databases are inadequate and for which object-oriented or object-relational are a better fit.
Overview

- OODBMS intended as relational database killer.
- Many a good company went bankrupt trying to put the relational vendors out of business.
- Not as formal a framework as there is in other models (although they tried).
- Has achieved use through extensions to SQL standard (RDBMS became ORDBMS -- object-relational database management systems).
Key Concepts in an Object Model

- Complex data types: Video, Images, Audio
- More built-in types
- Ability to extend with user-defined types
- Inheritance
- Object identification
- Pointers and swizzling
OODBMS

• Rationale
  • People are used to developing in OO languages.
  • Rather than add objects to SQL, why not add persistence to languages?
  • There were a bunch of “persistent foo” implementations: persistent C++, persistent small-talk, etc.
  • DO (which we’ll talk about in a minute) is an example of a standard that adds persistence to Java.

• Beyond Persistent-Foo
  • OO Languages don’t necessarily have a query interface.
  • Can certainly program queries, but part of the value of a DBMS is its ability to capture queries.
  • The Object Database Management Group (ODMG) introduced the ODMG data model and the OQL (Object Query Language) and ODL (Object Data Language)
ODMG Data Model

• The Basics
  • Database is a collection of objects.
  • Objects have unique OIDs.
  • Database contains collections of objects with similar properties.
  • Such a collection is called a class.
  • Properties of a class are specified using ODL.

• OQL
  • Designed to have SQL-like syntax.
  • Looks a lot like SQL:99
  • Retains basic “SELECT blah FROM blah WHERE” structure.
  • See book for examples.
ODL

- Attributes are either atomic or structured.
- Support set (unordered), bag (duplicates OK), list, array, and struct types
- Relationships have a type that is either a reference to an object or a collection of references.
- Relationships capture relationship both within and among classes.
- Methods can be applied to objects in a class.
- Interfaces define a class.
JDO: Java Data Objects

- Standard interface-based (API) way of adding persistence to Java.
- Like all things Java, this was a Sun-led standard.
- Two goals:
  - Transparent, java-centric view of persistence
  - Enable pluggable implementations of data stores into application servers
- Data is stored in a file system or “enterprise information systems” (i.e., Database Management Systems)
- Heterogeneity is fundamental: data can come from one or more storage instances. JDO operates on data of all sorts:
  - Row from a database
  - Object from an object database
  - Result of a general query
- Entire persistent state must be encapsulated in a Java object.
JDO Details

- Two standard interfaces:
  - `javax.jdo.PersistenceManager`
  - `javax.jdo.JDOHelper`

- The Persistence Manager provides
  - Query management
  - Transaction management
  - Life cycle management (buffering and sending to stable storage)

- `JDOHelper` provides
  - Bootstrap (acquire instance of `PersistenceManagerFactory`)
  - Find out what classes are persistent-capable
Using JDO

• JDO instances can be either transient or persistent.
• All persistent data must be in a JDO instance.
• All user classes and some system classes are persistent capable.
• First class objects are directly stored in a persistent store; have a unique JDO ID.
• Second class objects do not have a JDO ID and are stored only as a result of a referencing first class object being stored.
• Subclasses can be persistent even if superclass is not.
JDO Persistent Capable Classes

- Must have a no-arg constructor
- Must implement the PersistCapable interface
- PersistCapable allows application to find out the state of an object (transient, dirty, etc).
- JDO instances can be declared transactional (optimistic or locked)
- Callbacks allow application to regain control when some JDO action occurs (e.g., when a JDO instance is about to be written to disk).
  - jdoPostLoad (after read from disk)
  - jdoPreStore (before write to disk)
  - jdoPreDelete (before object is deleted)
  - jdoPreClear (restore default values)
Query in JDO

- Java API for query and a query language (JDOQL)
- Once you have one object, you can navigate from there. Three ways to get the first object:
  - Have or construct an Object ID
  - Via iteration
  - Construct and use a query object (JDOQL)
- JDOQL
  - Query language neutral (e.g., must support SQL and others).
  - Allow for optimization of query language.
  - Support multi-tier architecture (i.e., you do not know exactly where the query will be performed).
  - Large query results (i.e., may not fit into memory).
  - Support compiled queries.
  - Supports deletion via query.
The Object-Relational Model

- The relational response to the object-oriented data model was that there was no need to reinvent the wheel.
- Instead, simply add what we need to the relational model.
- What do we need?
  - New types: non-atomic columns, arrays, collections
  - New types: user-defined types (to handle things like audio/video, etc.)
  - Generalized extensibility
SQL Arrays

• One-dimensional typed arrays:
  Attr-name type ARRAY[size]
  • where type is a regular SQL type

• Examples:
  months varchar(8) ARRAY[12]
  ndays integer ARRAY[12]

• Operators on arrays
  • size of the array: CARDINALITY(ndays)
  • array access: ndays[i]
SQL Structured Types

• Call a structured type (struct) a row:
  • ROW(attr<sub>1</sub> type<sub>1</sub>, ..., attr<sub>n</sub> type<sub>n</sub>)
  
  CREATE TYPE month_t AS
  
  ROW(name varchar(8), ndays int)

  REF IS mid SYSTEM GENERATED;

• In SQL 99, ids can be SYSTEM or USER Generated OR derived

• Example:
  
  CREATE TABLE calendar
  
  (year integer, months month_t ARRAY[12]);

• Operators
  
  • Field access: calendar.year, calendar.months[4]

  • An attribute can be a structured type OR a reference to a structured type: months REF(month_t), in which case we perform field access using ->: calendar->months[4]
SQL Collection Types

- Not part of SQL:1999
- listof(type) (ordered)
  - operators: head, tail, append, prepend
- setof(type) no duplicates
- bagof(type) set w/dups
  - Operators: count, min, avg, subset
Abstract Data Types (ADTs)

• Allow users to define their own types
• Allow users to define methods that act upon their own types
• These types are opaque (abstract) from the point of view of the DBMS, but the DBMS will manipulate them using the methods provided by the user.
• Methods written in some language (e.g., Java) allowed by the DBMS on a given platform.
• Vendors sometimes sell packages of ADTs
  • Informix (IBM) DataBlades (based on Postgres)
  • Oracle Data Cartridges
  • IBM DB2 Extenders
• Example:
  • Consider a type for an image.
  • Methods might include: crop, rotate, compress, thumbnail
ADT Implementation Challenges

- Types/rows may be arbitrarily large.
- May vary tremendously in size.
- How do you store these things efficiently?
- How do you index abstract types?
- How do you optimize queries when you don’t fundamentally understand the underlying types?
- What if user-defined methods have bugs?
Big Items: Blobs

• Blobs (Binary Large OBjects) are the classic approach.
• Internal representation: cram them somewhere in the database (i.e., BDB big objects)
• External representation: store them elsewhere and store reference to them.
  • In files
  • In long contiguous pages allocated directly on disk
• Treated as opaque objects.
OO Approach to big items

- Stored externally.
- Treated as abstract data type (ADT).
- In addition to storing data, must have way to access, index, manipulate these large items.
- Provides fundamentally different functionality from blobs
Structured Items

• Objects contain pointers
  • How do you represent pointers on disk?
  • Do you use the same representation in memory?

• Objects contain or are collections
  • How do you represent collections?
  • What if collections are sorted?
  • What if they are arrays which implies a contiguous layout/allocation
  • How do you handle changes in the size of the collection (both growing and shrinking)?
How do you index ADTs?

• If the data type is completely abstract to the database, then you can’t index it, can you?
  • Allow applications to provide indexing techniques (more on this later).
• Even if the database knows about the type, do conventional indexes work?
• Hash tables provide what kind of lookups?
• B+-trees provide what kind of lookups?
• So, what do you do about other kinds of lookup:
  • Image similarity
  • Geospatial proximity (think Google earth)
  • N-dimensional data
  • Sub-spaces of N-dimensional data
  • Query-by-humming
  • Full text search
Extensible Database Systems

• Fundamentally this means letting users add code to your database!
• Movement in the late 80’s/early 90’s
• Pioneered by Postgres
• Commercialized by Miro/Illustra (DataBlades)
• Key idea: let applications download methods, optimizer functions, indexing techniques into your data manager.
• Problems?
Safe Extensions

- Interpreted instead of compiled
- Restricted to safe languages
- Post-processed to check safety
- Implemented as separate process (in separate address space)
- This was the early extensible system work and dovetails nicely with the emergence of safe languages and PCC
Performance Considerations

• Pointers
• Caching
• Concurrency
• Cost estimation and the Optimizer
Points

- Would like to use real pointers in memory, but must use OIDs on disk. How do you translate?
- Swizziling: The act of translating from disk addressing to memory addressing.
- Disk addressing typically in terms of OID.
- Memory addressing in terms of points.
- Approaches
  - Swizzle on access (keep table of OID-> memory mappings)
  - Use virtual memory to trigger swizzle.
  - Swizzle on eviction.
Caching

• All DBMS systems rely on caching to provide reasonable performance.
• What is the unit of caching?
  • Objects
  • Pages
  • Containers (large data segments containing potentially many objects)
• Granularity of caching dictates how much of the database resides on clients versus servers.
• Sometimes also useful to cache query results.
Concurrency

- Probably want granularity for concurrency to equal granularity for caching.
- Concurrency granularity may be critical from a throughput point of view.
Cost Estimation: Optimization

- How does the query processor account for ADTs?
  - Register indexes and relevant statistics
  - Provide callbacks and request user-defined functions
    provide data about distributions, counts, etc.