The ObjectStore Database System

Charles Lamb
Gordon Landis
Jack Orenstein
Dan Weinreb

(1991)

Goals

- Uniform programmatic interface to both persistent and transient data
- Object access speed for persistent data equal to (in-memory) pointer dereferencing to transient data

Close integration with Programming Language

- Choose C++: popular language in targeted applications (CAx, GIS)
- Adding persistence to C++
- Persistence is not part of the type of an object

Motivations

- *Ease of learning*
  - no need for a new type or new object definition
- *No translation code*
  - Between persistent data representation and transient data representation
  - Solve the ‘Impedance mismatch’ : persistent data is treated like transient data
- *Expressive power*
  - general purpose language (as opposed to SQL)

Motivations

- *Reusability*:
  - same code can operate on persistent or transient data
- *Ease of conversion*
  - data operations are syntactically the same for persistent and transient data
- *Type checking*
  - same static type-checking from C++ works for persistent data.

Motivations

- *Temporal/Spatial locality*
  - take advantage of common access patterns
- *Fine interleaving*
  - low overhead to allow frequent, small database operations
- *Performance*
  - do it all with good performance compared to RDBMSs

C++ extension to access persistent data

- Keyword: **persistent**
  - Used when declaring variables
- Keyword: **db**
  - Used when object being created should be allocated in database *db*.
- A few other keywords
  - **inverse_member**, **indexable**
  - for defining how objects in the DB relate.

```
main()
{
  database *db = database::open(“/company/records”);

  persistent<db> department* engineering_department;

  transaction::begin();

  employee *emp = new(db) employee(“Fred”);
  engineering_department->add_employee(emp);
  emp->salary = 1000;

  transaction::commit();
}
```

Discussion

Do you think it is a good idea to tie Object store to a popular programming language?

- If no, give your reason and a specific example.
- If yes, why? Given that there are other popular Object-oriented languages today such as Eiffel, C#, Java and Smalltalk, would you still go with C++? In addition to popularity, what are the other criteria needed to choose such an Object-oriented programming language?

ObjectStore supports

- Library of collection types
- Bidirectional relationships
- Access to persistent data inside transactions
- Optimizing query facility
- Version facility for collaborative work

Collections

- Similar to arrays in PL or tables in RDBMS
- Variety of behaviors:
  - Ordered collections (lists)
  - Collections with or without duplicates (bags or sets)
- Allow performance tuning
  - developers specify access patterns
  - an appropriate data structure is chosen transparently

Relationships

- Pairs of inverse pointers which are maintained by the system.
- One-to-one, one-to-many, and many-to-many relationships are supported.
- Syntactically, relationships are C++ data members
- Updates cause its inverse member to be updated.

Accessing persistent data

- Overhead is a major concern.
- Once objects have been retrieved, subsequent references should be as fast as an ordinary pointer dereference.
- Similar goals as a virtual memory system

  -- use VM system in OS for solution:

  - Set flags so that accessing a non-fetched persistent object causes page fault.
  - Upon fault, retrieve object.
  - Subsequent access is a normal pointer dereference

Associative Queries

- More closely integrated with the host language than SQL
- Any collections can be queried
- Special syntax: `[: predicate :]`

```
employees [: salary >= 10000 :]
```

- Queries may be nested to form more complex queries

Queries

- ObjectStore also uses indexes and a query optimizer
- BUT indexes are more complex
  - fields directly contained in objects
  - paths through objects and collections
- Index maintenance is more of a problem (embedded collections)

Query optimizations

Some RDBMS query optimization techniques don’t work or make sense

- Collections are not known by name
- Queries over a single top-level collection
- Join optimization is less of a problem
  - paths can be viewed as precomputed joins
  - join optimization now index selection issue
  - “true joins” are rare

Discussion

Would you rather use a relational database, or Object Store? More pointedly: for each of the following, list applications you would use with them and why:

- object store

- C++ and a relational dbms

Conclusion

- ObjectStore provides
  - Ease of use
  - Expressive power
  - Tight integration with host environment
  - High performance due to VM mapping architecture
- Performance experiments show caching and virtual memory-mapping architecture work.
- Small case study shows productivity benefits

Of Objects and Databases: A Decade of Turmoil

Carey, M.J.; DeWitt, D.J.
(1996)

Objects and Databases. Areas of research

- Extended relational database systems.
- Persistent programming languages.
- Object-oriented database systems.
- Database system toolkits/components.

Extended relational database systems

- Allow the addition of new, user-defined abstract data types (ADTs).
    - ADTs are implemented in an external language.
    - After being registered with the database, ADT’s functions can be used in queries.
- Projects:
    - Ingres
    - Postgres
        - Query optimizers with ADT’s properties and functions awareness.
        - Support for storing and querying complex data types.

Persistent Programming Languages

- Add data persistence and atomic program execution to traditional object-oriented programming languages.
- Problems addressed:
  - Impedance mismatch
  - Orthogonality
  - Persistence models
  - Binding and namespace management for persistent roots
  - Type systems and type safety
  - Alternative implementation techniques for supporting transparent navigation, maintenance, and garbage collection of persistent data structures

Object-Oriented Database Systems

- Combination of all of the features of a modern database system with those of an object-oriented programming language
- Focus on:
  - Reducing or eliminating ‘Impedance Mismatch’
  - Supporting querying, indexing and navigation
  - Addressing version management needs of engineering apps
- Projects:
  - Gemstone (Smalltalk)
  - Vbase (CLU-like language)
  - Orion (CLOS)

Database system toolkits/components

- Provide a DBMS that can be extended at almost any level
- Use mostly kernel facilities plus additional tools that help building domain-appropriate DBMS.
- Projects:

  EXODUS.

    - Storage manager for objects
    - Persistent Programming Language (E)
    - Query optimizer generator

  Starburst.

    - Part extended relational DBMS, part component–based DBMS
    - Clean architectural model that facilitates storage and indexing extensions
    - Rule-based extensible query subsystem

1996: What has happened since 1986?

- System toolkits & persistent programming languages
  - In spite of some interesting results these were a failure from a commercial point of view.
- OO database systems
  - Many results from the academic point of view. Not expanded commercially as expected by its developers.
- Language-specific object wrappers for relational databases
  - New approach that appears to be important for building OO, client side apps.
- Extended relational DBMS
  - Renamed as Object-Relational DBMS. Appears to be settling in terms of providing objects for enterprise DB apps.

The Database Toolkit approach problem

- Require a lot of expertise
- Inflexible, awkward or incomplete
- Not worthwhile to start from scratch despite toolkits to ease the process since OO-DBMS and OR-DBMS provide enough extensibility

Why did EXODUS fail?

- Its storage manager’s Client/Server architecture interfered with users’ implementation of their own object servers.
- E programming language
  - Too high-level for skilled database implementors
  - Too low-level for application-oriented programmers
- The query optimizer was inefficient and hard to use

Was all that bad after all?

- Interesting research by-products relevant to OO-DBMS and OR-DBMS

Persistent Programming Language

- No commercial implementation
- Still active as a research area in academia.
- Work transferred to OO-DBMS in areas
  - Navigational programming interfaces
  - Persistence models
  - Pointer Swizzling schemes
  - Garbage collection schemes for persistent data

What went wrong with OO-DBMS?

- No complete agreement on standards
- Tight coupling between an OO-DBMS and its application programming language
- OO-DBMS products lagging behind RDBMS (e.g. no view facilities!)
- Low availability of application development tools
- Difficult schema evolution
- Not adapted to prevalent computing environment of thin client/fat servers

Discussion

Given the problems stated with each of the four areas

- Extended relational database systems
    - Ingres, Postgres
- Persistent programming languages
    - JADE
- Object-oriented database systems
    - Objectstore
- Database system toolkits/components
    - EXODUS, Starburst

Which one would you still choose to research? Why? How would you overcome its issues?

What is OR-DBMS?

- Subsume RDBMS
  - starts from the relational model and its query language SQL and builds from there
  - Top level: collection of named relations BUT objects in the relations are as rich as can be supported by OO-db
- Supports object features
  - ADTs - extend set of built in types to new data types: text, image, audio, video, etc.
  - Row Types - direct extensions of type systems for tuples: rows in table can have object-like properties (named types & functions/methods)
- SQL extensions for object queries
  - Path expressions
  - Support for nested sets

2006: a fully integrated solution

Object relational servers will provide:

- Scalability and robustness
- Support for OO ADTs
  - Inheritance among ADTs
  - ADT implementation in various programming languages
- Full OO support for row types
- Methods and queries will be run on cached data on servers or clients depending on which method is faster

OO-dbms will remain:

- Niche solutions for areas such as engineering design, telecom…

2006: Research Challenges

- Server functionality and performance
- Client integration
- Parallelization
- Legacy data sources
- Standards

Discussion

- Was their vision for 2006 correct? In what ways?
- How is the reality different from their predictions? Why?
- Predict the future: What do you expect from OO-DBMS and OR-DBMS in 2016?

What are Object Oriented Client Wrappers?

- Gaining favour in commercial world
- Support the development of object-oriented, client side applications working against legacy databases
- Language specific
- Act as proxies for data in the underlying database allowing more natural interaction with data for programming tools.
- Tools to aid in the definition and construction of objects from the underlying db and maintain correspondences between programming objects and database data through key-to-OID
- Very weak querying side