The ObjectStore Database System

Charles Lamb, Gordon Landis, Jack A. Orenstein, Daniel Weinreb

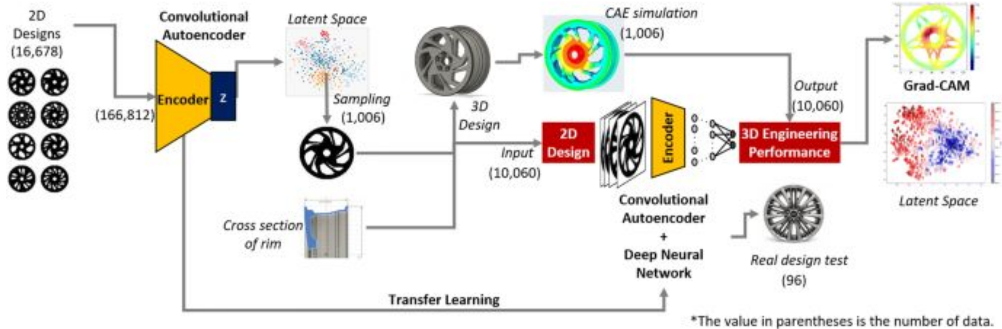
Presenter: Kaiyun
Discussion Lead: Chris

Slides adapted from Clint Morgan, modified by Rachel Pottinger

Motivation

Commercial DBMS targeting large datasets with intrinsic structure

- Provide a uniform programmatic interface to both persistent and transient data
- Shrink the impedance mismatch
  - Discrepancy between application code and database code
  - Object–relational mapping



The 2 Data Types

Persistent data

Affects the global monitoring environment.

Do not update the structure in-place, but yield a new updated structure.

Always preserves the previous version of itself when it is modified

Transient data

Only affects the current monitor session

Why Choose C++

- Ease of learning
- No translation code
    - persistent data is treated like transient data.
- Expressive power
    - general purpose language (as apposed to SQL)
- Reusability
- Conversion
- Type checking
- Temporal/Spatial locality
    - the next user of a data portion will be the same as the previous user.
- Fine interleaving
    - 'fetching an object'/dereferencing a pointer as fast as possible

Discussion 1 (Group of 2)

Consider the listed motivations of choosing C++, which features do you think are the most/least important in a query language? Please rank these features in order of importance.

Ease of learning	Ease of conversion
No translation code	Type checking
Expressive power	Temporal/Spatial locality
Reusability	Fine interleaving

ObjectStore add features

Three programming interfaces:

- C library interface
- C++ library interface
- Extened C++ interface for query & relationship facilities

Differ from C++ and most DBSMS:

- A collection facility (sets, lists, and so on)
- A way to express bidirectional relationships
- Support for groupware based on versioned data

Collections

- Object class library
- Abstract structures which resemble tables in relational structures
- A variety of behaviors
    - Ordered collections (lists), collections w/o duplicates (bags/sets)
- Performance-tuning facilities:
    - Replacing data structure
    - Policy
- Relationship facility:
    - A pair of inverse pointers guarantes referential integrity
    - One-to-one, one-to-many, many-to-many relationships

Queries

- Query syntax [: :] - C++ extension
- An expression operates one or more collections and produces a collection/reference to an object
- Any collection can be queried
- Current form: only semi-joins, not full joins

Accessing to Persistent Data

Guaranteed to be transaction consistent and recoverable in the event of system failure

- All-or-none update semantics

Once the target object has been retrieved from the database, subsequent references should be just as fast as dereferencing an ordinary pointer in the language.

1. No access to pages when accessing a non-fetched persistent object
2. Upon fault, retrieve page into client's cache
3. Subsequent access is a normal pointer dereference

Query optimizations?

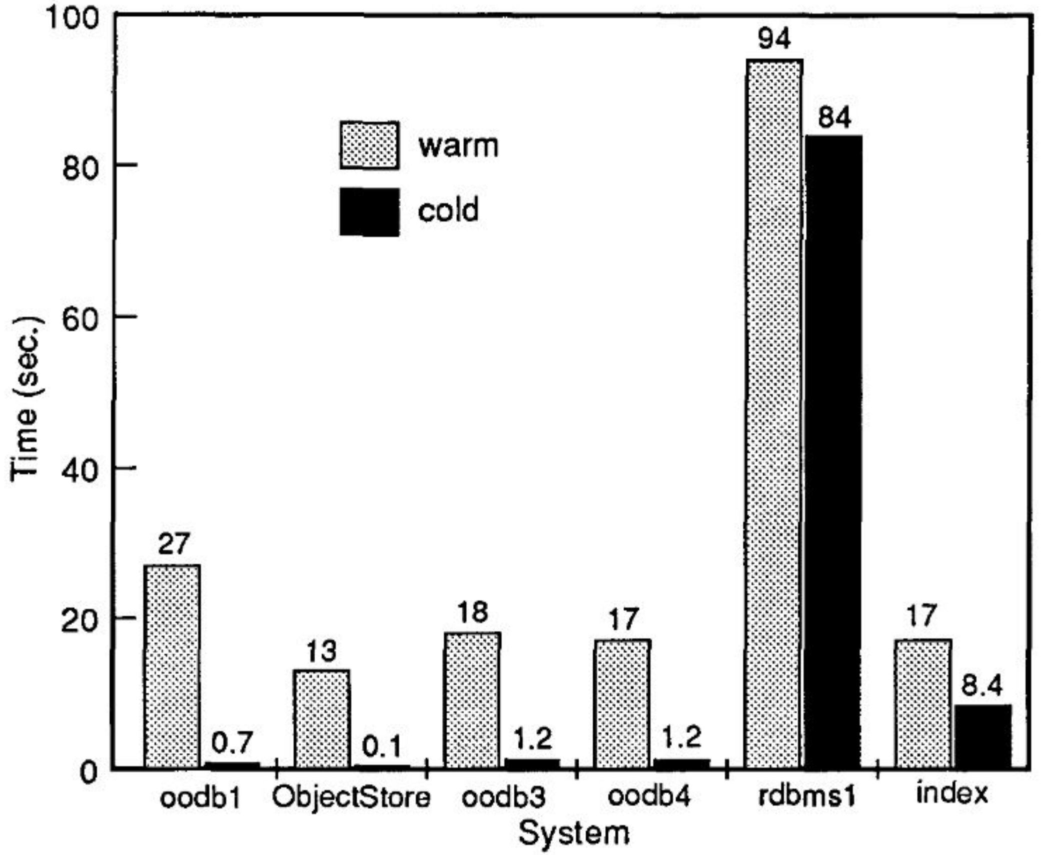
- Collections are not known by name
  - multiple strategies, with the final selection left until the moment the collection being queried is known
- Join optimization is less of a problem
  - paths can be viewed as precomputed joins
- Optimization - index selection
- Index maintenance is more of a problem
  - declare potential index keys
  - series of single-step indexes

Discussion 2 (Group of 4)

ObjectStore is closely integrated into the programming language, which is opposite to the separation of programming language and database in relational database management systems.

Do you think it is a beneficial innovation, or does it complicate development(good/bad idea)? Would you prefer such DBMS or the one that is independent, but is less powerful (e.g. SQL)? What are some features that SQL does not have, but you think are necessary to have?

Traversal Test



Conclusion

Designed for use in applications that perform complex manipulations on large databases of objects with intricate structure.

- Ease of use
- Expressive power
- Reusable code base
- Tight integration with the host environment

Of Objects and Databases: A Decade of Turmoil

M.J. Carey, D.J DeWitt

VLDB 10 year award for "Object and File Management in the EXODUS Extensible Database System" in VLDB 1986.

4 Areas of Research

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

Extended relational database systems

- Allow for new, user-defined abstract data types (ADT)
- ADTs are implemented in an external language.
- After being registered with the database
  - can be use as built-in type
  - functions can be used in queries

Persistent Programming Languages

- Add data persistency and atomic program execution to OO programming languages.
- Benefit from losing impedance mismatch
- Problems addressed:
  - Orthogonality
  - Persistence models.
  - Binding and namespace management for persistent roots.
  - Type systems and type safety.
  - Alternative implementation techniques for supporting transparent navigation, maintenance, & garbage collection of persistent data structures.

Object-oriented Database Systems

- Major motivation: remove impedance mismatch
- Focus on support for queries, indexing & navigation
- Addressing the version management needs of engineering applications
- Early years: No agreement on the details of
  - data model
  - query model/language
  - version management features

Database System Toolkits/Components

- No type of DMBS can meet requirement on next-generation applications
- DBMS extendable at almost any level
    - using mostly kernel facilities plus additional tools that help building domain-appropriate DBMS.
- EXODUS
    - Storage manager for objects
    - E: a persistent Prog. Language.
    - Query optimizer generator

What happened in 1996

System toolkits & persistent programming languages

Some interesting results, failure from commercial scene

OO database systems

Not org. commercial expectations

NEW: Language-specific object wrappers for relational databases

Important for building OO, client side apps

NEW: Object-Relational DBMS

Renamed from extended RDBS

Emerge as winner for providing objects for enterprise DB apps.

Discussion 3 (Group of 2)

After the initial explosion of objects in the database field, many interesting approaches and results occurred but were not commercially viable and therefore died off. What if an area/tech/approach lacks commercial value but still has academic/engineering value? Can you give some examples of database research that may not be commercially viable but provide critical value?

Failure of Exodus

Require a lot of expertise users

End up in being inflexible, awkward or incomplete.

OO and O-Relational database systems provide enough extensibility

- not worthy to start from scratch

- Companies use EXODUS to implement their own object servers
  - The storage client/server architecture - unwanted level of indirection
- E programming language:
  - Database implementors: want control over low-level details
  - Application-oriented programmers: too low-level
- Inefficient query optimizer

OODBMS MUST Support

1. Complex objects.
2. Object identity.
3. Encapsulation.
4. Inheritance and substitutability.
5. Late binding.
6. Computationally complete methods.
7. Extensible type system.
8. Persistence.
9. Secondary storage management.
10. Concurrency control.
11. Recovery.
12. Ad hoc queries.

OODBMS May Support

- Multiple inheritance
- Type checking (static vs. dynamic up to you)
- Distribution (client/server)
- Long xacts
- Version management

Optional

- Programming paradigm.
- Exact details of the type system.
- Degree of fanciness of the type system.
- Degree of uniformity of the object model.

Problems with OODB

- Impossible to gain complete agreements of standards.
- OODBMS behind RDBMS
    - No view facility
    - Robustness, scalability & fault-tolerance
- Painful schema evolution
- Tight coupling OODBMS & single application programming Language
- Low availability of application development tools/user env
    - Few end-user tool avaliable

Discussion 4 (Group of 3)

The inability to agree on standards has been cited as one of the main reasons for the failure of object-oriented database systems. Do you know any other areas that suffer from a lack of standardization? How can researchers agree on a standard in areas with multiple approaches? Who should set up standards?

ORDBMSs Must Support

- Provide support for richer object structures and rules
- Subsume second generation (i.e., relational) DBMSs
- Be open to other subsystems

- A rich type system, inheritance, functions & encapsulation, optional unique ids, and rules/triggers;
- A highlevel query-based interface, stored & virtual collections, updatable views, and separation of data model and performance features;
- Accessibility from multiple languages, layered persistence-oriented language bindings, SQL support, and a query-shipping client/server interfaoe.

ORDBMS

- Start with relational model and its query language
- Provide support for objects:
  - ADTs
  - Row types
  - Multi-valued attributes

Fully Integrated Solution in 2006

Fully support for OO ADTs (inheriency, adaptbility), OO row types

Support for middle-tier and desktops applications

> Provide a development environment where the same object model will describe the DB in all levels, both for querying and navigational programming.

Methods (and queries will be run on cached data on servers or clients depending on where’s faster.

Leftover challenges for 2006

- Server functionality and performance
  - query-processing, path indices, extensible access ADTs
- Client integration
  - querying over the cache + database
- Parallelization
  - parallelize object-relatrional queries
- Legacy data sources
  - make all the data available through a common query interface
- Standards
  - third-party data type of ADTs has own defining & interface

Discussion 5 (Group of 4)

This paper makes some predictions for the future, based on current applications/approaches, which areas do you think are headed in the right/wrong direction? It is interesting to note that these predictions are about commercial database products, they avoid predicting future research, what do you think is the reason? Predict anything in the database area for the next 5 years (products, trends ......).

Conclusion

- Look back 1986
- Massive change in 1996
  - lasting impact on the shape of the highly integrated, client/server, object-relational database solutions
- Look forward 2006
  - object-relational database systems will begin taking over the enterprise

… And where we are