Comparative Performance Measures of Relational and Object-Oriented Databases Using High Energy Physics Data

Superconducting Super Collider Laboratory
Disclaimer Notice

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Superconducting Super Collider Laboratory is an equal opportunity employer.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
Comparative Performance Measures of Relational and Object-Oriented Databases Using High Energy Physics Data

J. Marstaller

Superconducting Super Collider Laboratory†
2550 Beckleymeade Ave.
Dallas, TX 75237, USA

December 1993

†Operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract No. DE-AC35-89ER40486.
COMPARATIVE PERFORMANCE MEASURES OF RELATIONAL AND OBJECT-ORIENTED DATABASES USING HIGH ENERGY PHYSICS DATA

J. Marstaller
Superconducting Super Collider
2550 Beckleymeade Avenue
Dallas, TX 75237-3997

ABSTRACT

The major experiments at the SSC are expected to produce up to 1 Petabyte of data per year. The use of database techniques can significantly reduce the time it takes to access data. The goal of this project was to test which underlying data model, the relational or the object-oriented, would be better suited for archival and accessing high energy data. We describe the relational and the object-oriented data model and their implementation in commercial database management systems. To determine scalability we tested both implementations for 10-MB and 100-MB databases using storage and timing criteria.

1. Introduction

Historically, analysis of High Energy Physics (HEP) event data from detectors has been done using a serial, tape-based approach. Database computing offers several advantages to the traditional approach in physics analysis: 1) It may not be necessary to read an entire event into the computer, when only a few quantities are needed for a given analysis, thus reducing the I/O factor considerably. 2) Databases allow simultaneous access of the data. 3) Analysis of the data is immediate and interactive. 4) Data can be selected via simple queries using the standard query languages.

The basis for the project described here is within the framework of the Petabyte Access and Storage project (PASS). The extent to which HEP can benefit from database technology is to be evaluated. A HEP database, using existing data from the Collider Detector at Fermilab (CDF), was designed to make initial performance measurements of a relational database management system (RDBMS) versus an object-oriented database management system (OODBMS). Approximately 1 GB of CDF data was taken and divided into 100 datasets to be stored on disk. A database requires the data stored to be put into some kind of underlying data model, which provides in turn some form of stable structures. The goal is to determine the data model that produces better results.

2. Conceptual Design

When designing a database, the first step is to collect user requirements. These requirements can be put into a high-level construct that provides a preliminary view of how the user perceives the database. This high-level model, the conceptual model, describes how the user, i.e., a
physicist, conceives the domain area (HEP data). The conceptual data model was used as a means to produce a logical data model specific to the relational and the object-oriented implementation. A semantic data model, the entity-relationship (ER) model, was chosen to illustrate the conceptual design. An entity-relationship diagram is a graphical technique that depicts the enterprise being modeled as a collection of sets, relationships between the sets, and attributes associated with both entity and relationship sets. The most common extension to the ER diagram includes the abstraction concepts of aggregation and generalization. Aggregation abstracts a collection in such a way that objects (entities) are viewed as a single higher-level object. Generalization is a form of abstraction which captures the commonalties of objects (entities), while at the same time ignoring the differences. The extended entity-relationship model, (EER) model, is fairly simple to construct and has a reasonably unique interpretation. This model produced a diagram (Figure 1) which served as a means of communication among users and designers. The conceptual design ensured that no loss of information occurred, i.e., all entities mentioned in the data analysis tool were captured.

![Figure 1. Conceptual Design.](image)

3. Relational Design

The next step in designing the data model is to create a logical data model. The logical model has constructs that are easy for the users to follow and yet still avoids physical details of implementation. It results, however, in a direct computer implementation, in this case in an implementation in the relational database management system.

The EER model also depicts the logical data model. Figure 2 shows the partial model. The conceptual data model was used as a means to produce a logical data model specific to the relational and object-oriented implementation. The nature of the objects has an inherent hierarchical structure based upon level of details. To store this data in a traditional relational database, links need to be established between the objects to represent this hierarchy. Each run and event is given a unique identifier. Each child relation (such as “Tack,” “Vertex,” etc.) is augmented to include the primary key of its parent. In other words, any relation in the hierarchy must include the primary-key of its parent relation. This compares to data from computer-aided
design application. This hierarchy was modeled using weak entities. Weak entities depend on the existence of other entities for their identification. The CASE tool we used, ERDRAW, uses non-directed edges, labeled ID, to show the dependency. The entity “Track” was modeled to reflect the CDF data. The entity “Track” is the generalization of two homogenous entities “Defaulttrack” and “Otherversiontrack.” The homogenous entities “Defaulttrack” and “Otherversiontrack” only contain the common attributes, whereas the generalization “Track” has additional attributes. ERDRAW provides the directed edges labeled “ISA” for illustration.

Figure 2. Part One of the logical design for RDBMS using ERDRAW notation.

4. Object-Oriented Design

The ultimate goal of the object-oriented data model design is to produce a system that consists of a collection of objects. The focal point of the design task is to determine and specify those classes that govern the structure and functionality of these objects. As in the relational design method, the key is to identify the abstractions that occur in the problem domain.

Objects are concepts, abstractions, or things with crisp boundaries within the problem domain (HEP). An object class describes a group of objects with similar properties (attributes), common behavior (methods), common relationships to other objects and common semantics. A class can be viewed as a template, which specifies the intended purpose of the objects or as an abstraction mechanism. Once the object classes were established, we identified and recognized relationships that exist between the classes within the application domain (HEP). The main goal
is to recognize patterns for aggregation, generalization, specialization or inheritance on both the object and the class level. This object-oriented logical data model was produced using OMTool. OMTool follows the basic conventions for object model notation as described in Rumbaugh. Figure 3 shows the logical design for the OODBMS. The class “Event” is on top of the hierarchy and depending on the event the corresponding instantiations of the other classes can be found, i.e., “Hit,” “Track,” etc., have a relationship to “Event” so each of these classes is part of an “Event.” The class “Particle” is modeled as an abstract class (one that has no direct instantiations). The subclasses of “Particle,” the different types of particles, inherit all attributes from their superclass.

Figure 3. Logical Design for the OODBMS using OMTool notation.
5. Testing Criteria

For the purpose of this project, we wanted to measure how well the underlying data model reflects and enhances the performance of the system. Our test measures response time, i.e., the real time elapsed, from the point a program calls the database system with a particular query, until the results of the query, if any, have been placed into the program's variables.

There are two broad categories of performance measures: time and storage. We wanted to determine the storage efficiency of the database management systems; i.e., determine what fraction of a database is actual source data and what is database management overhead. We had a 10-MB and a 100-MB database and checked the size of the database dumpfiles on disk.

We looked at only three time measurements: 1) the time it takes to create a database, 2) the time it takes to archive a database, and 3) the time it takes to complete the queries. All tests are run on disk resident databases.

6. Results

The goal of our test was to model a specific environment, the high energy physics analysis environment, and the aspects of how the two database systems are performing in this environment; therefore, a lot of time was spent in modeling the HEP data and reflecting this environment in the relational and object-oriented data model. All measurements are summarized in Figure 4.

<table>
<thead>
<tr>
<th></th>
<th>RDBMS</th>
<th>OODBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage:</td>
<td>8.4 MB</td>
<td>84 MB</td>
</tr>
<tr>
<td>Time:</td>
<td>3:47.0</td>
<td>28:18.0</td>
</tr>
<tr>
<td>Populate</td>
<td>7.2</td>
<td>1:37.6</td>
</tr>
<tr>
<td>Dump db</td>
<td>27.8</td>
<td>5:33.4</td>
</tr>
<tr>
<td>Load db</td>
<td>24.4</td>
<td>36.0</td>
</tr>
<tr>
<td>Query:vfit</td>
<td>24.4</td>
<td>36.0</td>
</tr>
</tbody>
</table>

Figure 4. Time Measurements in min:sec. 1/10 sec.

Storage:

To measure the overhead of each database management system, we determined the size of the files to be archived. The relational database dumpfile is larger because it stores a self-contained database that includes all of its pertaining information. The object-oriented database schema is not part of the size of the database file size recorded. For both database systems, the schema remains static throughout analysis. The schema must, however, exist prior to loading a database file into an existing healthy database. For the OODBMS, the schema files are of insignificant size, so we did not archive them with the database file. The OODBMS database file is approximately 25% smaller than the RDBMS database file.
**Time:**

1. The first time measurement was to see how long it takes to create and fill (populate) a database for both database systems and both sizes (10 MB and 100 MB). The RDBMS showed much more consistent results in creating and loading both database sizes. When loading a relational database, only the actual data is loaded, whereas when loading an object-oriented database the pointers, or the paths, have to be loaded also. This reflects the time increase for the OODBMS. For 10 MB the OODBMS performed approximately 25% better than the RDBMS, however, for 100 MB the performance deteriorates and becomes approximately 12% worse than the RDBMS.

2. Archiving the database files was the next time measurement recorded. For both DBMS, we used vendor-provided utilities to archive the datafiles. Archiving included loading a database file from disk and dumping a database file for storage. Both measurements are important when 1 GB of data is divided into separate datasets so that archiving can eventually be done in a distributed, parallel fashion. The RDBMS performed consistently better than the OODBMS. To retrieve a database file, the RDBMS requires approximately 18% less time for the 10 MB and 24% less for 100 MB. To store 10 MB for the RDBMS it requires approximately 31% and for 100 MB 38% less time than for the OODBMS.

3. For the query used as a test measurent, we selected a “track refitting” query. There are other physics query results, but the “refitting” query reflects a standard traversal over three quite large tables, five levels deep. The RDBMS scales up better, without optimization from the designer (25%). In the OODBMS, where the user has to take care of all query optimization, using such features as “path of,” which will store a predefined indexed path for a query, the performance can be increased dramatically by implementing such changes.

Both relational and object-oriented databases have their own inherent problems for the physics research community. Relational databases provide a simple way of extracting data with SQL, but all analyses must be done outside the database. In other words, for the algorithms there still needs to be a layer of application programs on top of the database. SQL does not have sufficient capabilities, such as recursion. This causes problems, such as the cost of writing application programs, the granularity of the data present in the programs and the impedance mismatch, however, this approach seems to be rather attractive to the physicists since most of the algorithms are already implemented in FORTRAN. Object-oriented databases, on the other hand, are less mature than relational systems and are more difficult to fine tune for performance.

The grand picture is, of course, to develop a system that is very flexible, modular, scalable and distributed. It should, essentially, take data and process it into information, preferably visual information, so that the user can extract facts and make meaningful conclusions from that. Use of databases is essential for these projects, however, the design of such a database is not yet fully understood. The next-generation database application has little in common with current business data processing. Since a need for much larger, more complex datasets exists, new capabilities in areas such as type extensions, complex objects, rule processing and archival storage are required.
This project was not designed to be “another” benchmark in the area of database comparison, therefore, traditional database benchmarks, such as “insert” were not measured. We wanted to show the performance characteristics of the database management systems in the application domain, i.e., high energy physics applications. The benchmarks designed for generic use and overall performance were applicable, but less desirable than designing specific tests.

7. Acknowledgments

This paper has been produced within the framework of the PASS project\(^1\) at the SSCL. The author has participated in this project, designing and implementing the databases and creating the performance measures. She would like to thank Alain Gauthier for developing the “track refitting” query incorporating existing FORTRAN code. She would like to express special thanks to Ed May and Ute Nixdorf from the SSCL who were crucial to the success of this exercise.

8. References