Use of Relational Storage and a Semantic Model to Generate Objects: The PENGUIN Project

Gio Wiederhold, Thierry Barsalou,
Byung Suk Lee, Niki Siambela, Walter Sujansky
Stanford University, Stanford CA 94305

1. Introduction

Relational technology provides a basis for reliable sharing of large bodies of information. Object approaches provide conceptual locality of information to focus users attention. Application systems, similarily, tend to be effective if the data space is well focused and constrained. Future decision-support systems will require the combined use of database and intelligent applications.

2. The PENGUIN project

In the PENGUIN project, we demonstrate the hypothesis that relational storage, used to support an object-oriented approach can serve as a unifying framework for developing systems which share information [Wied:86].

Objects offer a consistent framework for users and their applications. Permanently storing information in the form of complex objects, however, can seriously inhibit sharing and flexibility, since an object presents one particular view of the world. A desirable compromise is to exploit existing database technology by defining an object-based layer on top of a relational database management system [BarW:90]. This approach does not require storing objects explicitly in the database, but rather calls for generating and manipulating temporary object instances by binding data from base relations to predefined object templates. PENGUIN introduces a tripartite architecture, where the object layer mediates between a relational-database layer and a knowledge-based layer [BarW:90]. This presentation presents an overview of our work on the database and object layers.

Views and Objects

There are many similarities between views and object concepts. Both are intended to provide a better level of abstraction to the user, although the database user is seen to manipulate sets of objects in a non-procedural notation while the objects are manipulated procedurally and iteratively [DiDa:86].

The *collection* of tuples of a view is defined by the query which generates the set, and described by the relation-schema associated with the view query. The set of objects is described by a programmer using a class prototype declaration. Actual instances are created when the user executes the program and fills in slots of object instances matching the objects class. Object-oriented languages may have a `ForAll` primitive to rapidly generate collections of objects of some class. Objects are made persistent in object databases by causing their entire contents to be stored. They can only be read back into the same program, or perhaps into programs with identical class definitions.

The description of the relation has to be available to the the relational user, since no implied operations can be kept in the relation-schema. There are proposals to store object defining procedures in relational schemas [StRo:85].

Both tuples and objects can be selected based on any of multiple criteria, and interrogated to yield data for processing. View update may be severely restricted due to ambiguities in base relation update [DaBe:82]. Object update can be restricted by having only limited access functions, but otherwise no constraints are imposed on the programmer, although consistency problems can easily arise among users sharing objects.

Basic, of course, is the difference in persistence. Databases, and hence views over them, persist between program invocations. Objects must be explicitly written to files to gain persistence. Related to persistence is the critical issue to be addressed, namely the multiplicity of views that can be derived from a base relation.

View and Object Similarities

Better abstraction to the user,

DB: Database non-procedural

OB: Objects manipulated procedurally and iteratively

Collection of

DB: tuples defined by query

OB: objects defined by the user's generation

Description

DB: schema associated with the view query

OB: local class prototype determines object instances

Operations

DB: inherited from prototype object

OB: no implied operations in relation-schema.

Update

DB: View update severely restricted due to ambiguities

OB: Object update can be restricted by functions

Same consistency problems among users sharing objects.

Sharability

DB: algebra permits selection of relevant information

OB: Objects are shared in the form they are stored

Figure 1. Views and Objects.

$\uparrow$

$\boxed{\textbf{Structural Model}}$

Figure 2. Penguin Architecture.

3. Management of Semantics

At the core of PENGUIN is a semantic data model called the structural model. The structural model augments the relational model by representing declaratively the knowledge about the constraints and dependencies among the relations in the database. Its primitives are relations, as determined by the normalization process, and connections to describe those relationships between the relations [Wied:83]. Three types of connections are defined that correspond to precise integrity constraints, define the permissible cardinality of the relationships, and encode the relationships' semantics (e.g., aggregation or generalization) [SmSm:77]. PENGUIN exploits the structural model's knowledge of the database in both the relational and the object layer.

Figure 3. Structural model of a ship database.

Structural connections describe the relationships among the entities, for example: `Ships` <u>`belong_to`</u> `Ship_classes`. Connections constrain the model, we need an object in `Ship_classes` to match every reference made by any of the `Ships`.

We distinguish four types of connections among relations [WiQi:87], three of them appear in Figure 3:

- O Ownership connections   Symbol: ——*
  linking lower entity instances which depend on the owner; the lower level entities disappear when the owner is removed. (e.g., `Ships` own their `Voyages`)
- R Reference connections   Symbol: >——
  linking from instances to higher level abstract referenced entities; the referenced entities are constrained to exist while referred to. (e.g., `Stops` referring to `Ports`)
- S Subset connections   Symbol: ——⊃
  linking from generalization to a subtype; the subtype instance is removed when the general instance is deleted. (e.g., `Docks` linked to `Passenger_docks`)
- I Identity connections   Symbol: ——=
  linking base data to derived data; the derived copy must, eventually, be recomputed, to assure consistency. (e.g., from `Cargoes.Volume` we can derive `Warehouse.Space`)

Relationships imply constraints and hence we associate with each connection type a rule stating how the database, and hence the programs that implement these connections, should behave.

Having a connection between two relations requires that there will be matching data attribute. An instance of a connection between the tuples exists when the actual data values match, for example, when the `Ships.country_of_registry` field value matches a specific `Countries.name` value.

The rule for the reference connection states that in this case there should be no `Ships.country_of_registry` field value which does not exist as a `Countries.name` value. Also, it is not legal to delete a `Countries` entry while there are still `Ships` listed as being registered there.

We use these concepts to provide a graphical representation of the database. A business database may have a hundred relations. In practice there are often dozens and perhaps hundreds of attributes in a single relation. There are typically two to three times as many connections as relations.

4. Sharability

The problem of sharing was first made explicit in VLSI design, where the objects defined in specific applications, as logic, layout, and timing analyses differed radically. Computer-assisted design of immunologic-research experiments served as PENGUIN's initial testbed. This application demonstrates the use of sharing information through view objects in a genetics laborarator, where immunologists, experiment designers, and operators of a cell-sorter must cooperate. This integration showed that the object layer can serve as a domain-independent, bidirectional communication channel between relational database systems and knowledge-based systems [Gian:90].

At the lowest level of abstraction, a standard relational database encodes the complex collection of information relevant to the domain. Structural semantics improves standard database operations in various ways. The structural model supports the integration of many user views into a comprehensive database model, thereby facilitating the design process.

Once the logical design is completed, a suite of tools is used to define, verify, and explore the relational schema. Data retrieval and manipulation is then performed through a cooperative user interface.

5. Components

The three main components of PENGUIN are the object generator, the object instantiator, and the object decomposer. The object generator maps relations into object templates through a mixed-initiative dialog with the user, where each template can be a complex combination of join and projection operations on the base relations. Object templates are then arranged into object networks, which collectively define the object schema. The process is knowledge-driven, using the semantics of the database structure.

Figure 4. Creating a Object Template from the Database Schema.

5.1 Object Template Generation

The first step in use of PENGUIN is View-object definition. This is being carried out interactively and assures the creation of correct object templates from the schema of a relational database, augmented with formal semantics in the form of a structural model. Each template is *anchored* on a *pivot* relation, as shown in Figure 4. The key of the pivot relation serves as the object identifier when objects are instantiated. The required SQL queries are generated automatically, and shipped to the database server when object instances are selected by the user.

5.2 Object Instantiation

In the object layer, PENGUIN generates and manipulates temporary object instances by binding data from base relations to predefined object templates When the program needs instances of the objects they can be selected either interactivly or by programming. Figure 5 shows an example, where pointing to an element in the object class and entering a predicate for selection causes the previously generated and stored SQL-query to be augmented with the predicate prior to execution.

Figure 5. Object Instantiation.

The object instantiator provides nonprocedural access to the actual object instances. Combining the database-access function (stored in the template), and a declarative query, PENGUIN automatically generates the relational query and transmits it to the DBMS, which in turn transmits back the set of matching relational tuples. Those tuples are then assembled into new object instances.

The interactive interface exploits simultaneously the browsing and direct-manipulation features of hypertext environments, the analytical-querying and concurrency-control features of relational DBMSs, and the declarative semantics introduced by the structural model to ease the man-machine interaction at different stages of a database session [BaCh:89].

5.3 Object Decomposition

Finally, results have to be made available to other participants by placing them into the relational database. The object decomposer implements this function; that is, it maps the object instances back to the base relations. This component is invoked when changes to some object instances need to be made persistent at the database level. At the time of template generation all possible ambiguities are enumerated and resolved, so that at this point the required transforamtions can occur expeditiously [BSKW:91]. Figure 6 shows the dialogue that takes place to resolve a specific set of update ambiguities.

Database consistency, as defined by the integrity rules of the structural model, is dynamically enforced by program modules which monitor transactions and intervene to prevent or correct illegal updates [WWHC:89].

The modules involved in instantiation (solid heavy line) and decomposition (dashed heavy line) are shown in Figure 7. The thin lines show the template generation and use flow.

Figure 6. Dialogue to assure unambigous update.

Figure 7. Dataflow among Penguin modules during execution.

Figure 8. Message passing among Penguin modules during execution.

A reimplementation of PENGUIN, to better support maintainability and portability is in process and should be completed by this summer. Not surprisingly, this implementation also uses object-oriented programming concepts. We focus on very large object modules, since we find that conventional structured programming approaches are not effective in large-scale programming tasks. Figure 8 indicates the role of the central message handler and the message flows corresponding to the dataflows shown in Figure 7.

6. Conclusion

We have demonstrated a novel and powerful technique, PENGUIN, to derive object-oriented data structures from a shared relational database. The objects are tailored to adapt themselves to the needs of workstation applications, while maintaining strict consistency with the underlying shared database model.

Our results indicate that PENGUIN's object-based architecture provides correct and effective access to and manipulation of complex units of information while preserving the advantages associated with persistent storage of data in relational format. Computer Science research issues that have been addressed achieve optimization within a server-workstation environment. We have results on minimizing the need for joins and outerjoins, and on obtaining minimal data transfer from relational database servers to the workstations using object representations [Lee:90].

Currently technology transfers to Civil Engineering, for the design of structures, and to the Center for Integrated Systems, to manage manufacturing processes, are in progress [LSBW:91.

Acknowledgements

This work was principally supported by the National Library of Medicine under Grant R01 LM04836, by industrial funding via Stanford CIFE Center in Civil Engineering, and by Digital Equipment Corporation under the Quantum project. Additional computer facilities were provided by the SUMEX-AIM resource under NIH Grant RR-00785. Much fundamental research derives from DARPA-sponsored work under contract N39-84-C-211.

References

[BaCh:89] Thierry Barsalou, R. Martin Chavez, and Gio Wiederhold: "Hypertext Interfaces for Decision-Support Systems: A Case Study"; Proc IFIP MEDINFO 89, Beijing and Singapore, Dec. 1989, pp. 126-130.

[BarW:89] T. Barsalou and G. Wiederhold: "Knowledge-directed Mediation Between Application Objects and Base Data"; Proceedings of the Working Conference on Data and Knowledge Base Integration, University of Keele, England, October 1989; in S.M. Deen (ed.) 'Data and Knowledge Base Integration' 1990, Pitman, London, UK.

[BarW:90] Barsalou, Thierry and Gio Wiederhold: "Complex Objects For Relational Databases"; Computer Aided Design, Vol. 22 No.8, Buttersworth, Great Britain, October 1990, pages 458-468.

[BSKW:91] Barsalou, Thierry, Niki Siambela, Arthur M. Keller, and Gio Wiederhold: "Updating Relational Databases Through Object-based Views"; *Proceedings of ACM-SIGMOD 91*, Boulder CO, May 1991.

[DaBe:82] Umesh Dayal and Phil A.Bernstein: "On the Correct Translation of Update Operations On the Relational View"; *ACM TODS*, vol.7 no.3, Sep.1983.

[DiDa:86] Klaus Dittrich and Umesh Dayal (Eds.): *Proceedings 1986 International Workshop on Object-Oriented Database Systems*; IEEE CS Order no.734, Sep.1986

[Gian:90] R.V. Giangrande: System Integration Technologies, Vol.1: Issues and Options; Transportation Systems Center, Cambridge MA, February 1990.

[Lee:90] Byung Suk Lee: *Efficiency in Instantiating Objects from Relational Databases Through Views*; Thesis. Report No. STAN-CS-90-1346, Dec. 1990.

[LSBW:91] Law, Kincho, Niki Siambela, Thierry Barsalou, and Gio Wiederhold: "An Architecture for Managing Design Objects in a Shareable Relational Framework"; accepted for the *International Journal of Systems Automation Research and Applications* (SARA), to appear 1991.

[SmSm:77] Smith, John and Smith, Diane C.P.: "Database Abstractions: Aggregation and Generalization"; *ACM TODS*; vol.2 no.2. June 1977, pp.105–133.

[StRo:85] Michael Stonebraker and Larry Rowe: "The Design of POSTGRES"; Tech.Report UC Berkeley, Nov.1985.

[Wied:83] Gio Wiederhold: *Database Design*; McGraw-Hill, 1983.

[Wied:86] Gio Wiederhold: "Views, Objects, and Databases"; *IEEE Computer*, Vol.19 No.12, pages 37-44, December 1986.

[WiQi:87] Gio Wiederhold and XiaoLei Qian: "Modeling Asynchrony in Distributed Databases"; *Third IEEE Computer Society Data Engineering Conference*, Los Angeles, Feb. 1987.

[WWHC:89] G.CM Wiederhold, M.G. Walker, W. Hasan, S. Chaudhuri, A. Swami, S.K. Cha, X-L. Qian, M. Winslett, L. DeMichiel, and P.K. Rathmann: "KSYS: An Architecture for Integrating Databases and Knowledge Bases"; in Amar Gupta (ed.), *Heterogenous Integrated Information Systems*, IEEE Press, 1989.