Software Engineering Design Patterns for Relational Databases

Cyril S. Ku
Dept. of Computer Science
William Paterson University
Wayne, New Jersey
USA

Thomas J. Marlowe
Dept. of Math & Comp. Sci.
Seton Hall University
South Orange, New Jersey
USA

Tatyana Budanskaya
Dept. of Comp. Sci.
NJIT
Newark, New Jersey
USA

Philip K. Kang
Dept. of Computer Science
William Paterson University
Wayne, New Jersey
USA

Abstract – The use of design patterns such as the GRASP (General Responsibility Assignment Software Principles) or GoF (Gang-of-Four) patterns in software engineering has been well-documented and widely used in software design and implementation. Research efforts have also been made to apply these generic software engineering design patterns to other design and implementation endeavors in computer science. One such effort is our research [8] to propose a strategy for the integration of the teaching of software engineering and databases, using an innovative curriculum for teaching database design and implementation based on the UML (Unified Modeling Language), the Unified Process, and design patterns. Our previous effort [6] has demonstrated the use of GRASP patterns for the development of relational databases. In this research, we further investigate GoF design patterns for the design and implementation of relational databases. The application of generic software engineering design patterns to databases allows issues in logical design and in implementation of databases to be more easily connect with similar concerns in software engineering.

Keywords: Database, Database Design, Design Pattern, Relational Database, Software Engineering

1 Introduction

The specification and design of relational databases traditionally use a domain-specific design approach based on ER (Entity-Relationship) [2] and EER (Extended ER) [3] data models and still other approaches for modeling dynamic (query) and systems aspects of a design. In contrast, modern object-oriented software engineering has largely standardized on the use of the Unified Process [5], UML [1, 11], and design patterns [4, 7]. Since a database system is a large-scale software application, the specification, design, and development of database management system should follow the software development life cycle of any large-scale software project. The goal of our research is to use the UML models, design patterns, and the Unified Process for the development life cycle of databases. Specifically in this paper, we discuss the application of generic software engineering design patterns (specifically, the GoF patterns) to the design and development of relational databases.

Generic software engineering design patterns can be classified into categories exemplified by the GRASP [7] and GoF [4] patterns. GRASP provides general guidelines for class design and responsibility assignment while the GoF patterns are building block patterns that provide design and code scaffolding.

2 Related Work


Instead of using domain-specific patterns, our research effort uses the generic software engineering design patterns (GRASP and GoF) for the design and development of databases. Our research on the applicability of GRASP patterns to relational databases can be found in [6]. Preliminary research of using GoF patterns for relational database development has been done by us [8]. This paper further investigates and summarizes the applicability of GoF design patterns on the design and implementation of relational databases and data warehouses.
3 Research Benefits

The potential benefits of this research are manifold. First of all, standardization of design and development of relational database systems with modern object-oriented software engineering methodology will provide better communication across disciplines and make connections with other software projects. This will unify and simplify communication with management and customer/user representatives in complex projects, as well as with developers of other application modules. Economically, the cost of learning and training is amortized across all types of software development efforts.

Recent work [12] suggests that the use of design patterns produces better and faster design in software engineering. This research should extend those benefits to databases, resulting in better and faster database design and development. The use of patterns supports reuse of knowledge, insights, and components, and development of design idioms, and the use of UML provides uniform descriptions of relational database and object-oriented database design and implementation. It will increase the portability of design between a relational approach and a hybrid (e.g., object-relational) approach.

As we mentioned in [8], the application of software engineering design patterns to databases allows issues in logical design and in implementation to be more easily connected with similar concerns. Pedagogically, a common and standardized approach to design will aid students’ acquisition of both knowledge and concepts in software engineering and databases.

4 Application of GoF Patterns to Relational Databases

The GoF patterns are categorized into Creational, Structural, and Behavioral patterns [4]. There are totally 23 GoF design patterns. In this paper, we identify 15 of these patterns that can be used directly for the design or implementation of relational database systems.

The descriptions of the GoF patterns as they apply to the design and/or development of relational databases are provided below. The names of the patterns are in italics. We use UML class diagrams in place of ER or EER models for database specification and design [9]. In the following discussion, “class” can be replaced by “table” in relational model. “Object” is the instance of “class”, “schema”, “entity-set”, or “table”.

Due to the limitation of space in this paper, the UML diagrams for the following patterns as they apply to the design and development of relational databases are not shown. Here is an example of the Façade pattern to give the reader a flavor of what it involves. Façade is widely used to hide the complexity of a subsystem behind a single class, which provides a single interface into the subsystems and coordinates the interactions among the components of the subsystems. The pattern can be represented by a template (or parameterized) collaboration (Figure 1). The dashed oval represents the collaboration that is the Façade pattern. The surrounding boxes represent the collaborators, which in this case show the pattern applies to the data mart as an interface to the data warehouse, the operational database, and other legacy system. The dependencies show that the collaboration depends on the collaborators. The label on a dependency tells which role the collaborator satisfies the pattern. The template collaboration shows only the participating roles and classes that fill those roles. The structure of the collaboration can be shown by a class diagram and the interaction involved in the pattern can be modeled with an interaction diagram such as the communication diagram or the sequence diagram.

![Figure 1: Collaboration of the Façade Pattern](image)

4.1 Creational – Builder

The Builder pattern separates the construction of a complex object from its representation so that the same construction process can create different representations [4]. This pattern can be applied to relational database as follows:
- The construction of aggregate or composite tables from the component tables.
• The creation of logical schemas (relational tables) from the internal representation in file management system.

4.2 Structural – Adapter

The Adapter pattern converts the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces [4]. This pattern can be applied to relational database as follows:
• Interface between table-oriented DBMS layer and file management layer.
• Interface between OLTP (On-Line Transaction Processing) system and OLAP (On-Line Analytical Processing) system.
• User interface to query language translation.
• Queries of federated databases with variation in data schemas – particularly when legacy data is involved.

4.3 Structural – Bridge

The Bridge pattern decouples an abstraction from its implementation so that the two can vary independently [4]. This pattern can be applied to relational database as follows:
• The data independence concept in relational database (logical schema and physical data structure independence).
• Multiple realizations of database schemas across distributed or federated databases.

4.4 Structural – Composite

The Composite pattern composes objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly [4]. This pattern can be applied to relational database as follows:
• Implementation of the aggregation or composition concept.
• Table traversal of complex queries (or access plan) with the Visitor pattern.

4.5 Structural – Decorator

The Decorator attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality [4]. This pattern can be applied to relational database as follows:
• Specific realizations of unions (super-class/sub-class relationships) can be seen as instances of Decorator.

4.6 Structural – Facade

The Façade pattern provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use [4]. This pattern can be applied to relational database in data warehousing architecture as follows:
• Interface between data warehouse/data mart and operational databases (see Figure 4.1).
• Interfaces among distributed database (not necessary relational) systems.

4.7 Structural – Proxy

The Proxy pattern provides a surrogate or placeholder for another object to control access to it [7]. This pattern can be applied to relational database as follows:
• The enforcement of various database constraints (access control or security).
• Efficient access plans for queries involving remote queries in distributed databases (see the Memento pattern).

4.8 Behavioral – Command

The Command pattern encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations [4]. This pattern can be applied to relational database as follows:
• Together with the Proxy pattern, enforce domain, key and referential integrity constraints by rejecting or modifying illegal updates.

4.9 Behavioral – Iterator

The Iterator pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation [4]. This pattern can be applied to relational database as follows:
• Table traversal for non-complex queries.

4.10 Behavioral – Mediator

The Mediator pattern defines an object that encapsulates how a set of objects interact. Mediator
promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently [4]. This pattern can be applied to relational database as follows:

- Complex deferred triggers between databases and data warehouse.

### 4.11 Behavioral – Memento

Without violating encapsulation, the Memento pattern captures and externalizes an object’s internal state so that the object can be restored to this state later [4]. This pattern can be applied to relational database as follows:

- Together with the Proxy pattern, allow access to old state to model transactions.
- Also with Proxy, model distributed queries and the needed locking for concurrent operations.

### 4.12 Behavioral – Observer

The Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically [4]. This pattern can be applied to relational database as follows:

- Database views (to filter appropriate, up-to-date entries and fields from the base tables, and/or to provide controlled access to data).
- Data warehouse or data marts as observer of operational databases.

### 4.13 Behavioral – State and Strategy

The State pattern allows an object to alter its behavior when its internal state changes. The object will appear to change its class and the Strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it [4]. These two patterns can be used together to apply to relational database as follows:

- Support fast query-time optimization of stored access plans, allowing for selection among multiple implementations for join, set operations, and temporary files, particularly in stored queries and views.
- Database reorganization – dynamic conversion between file structures.
- Implementation of “what-if” analysis (with the Command pattern).

### 4.14 Behavioral – Visitor

The Visitor pattern represents an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates [4]. This pattern can be applied to relational database as follows:

- With the Composite pattern, table traversal (access plan) of complex queries.

### 5 Conclusion and Future Research

In this paper, we show many examples of applying the GoF design patterns to the design and implementation of relational databases. As the examples show, many of the concepts can be explained, designed, and developed by the GoF software engineering patterns.

Preliminary efforts have used GoF patterns for relational database design and implementation from the pedagogical viewpoint of designing a comprehensive curriculum for software engineering and database courses [8]. In future research, we will combine generic software engineering patterns (the GoF and GRASP patterns) with the domain-specific patterns such as [10], to demonstrate the applicability in databases and data warehouses. We are in the process of identifying and organizing more examples.

The use of UML and generic software engineering design patterns for relational database is only part of the research goal to eventually have standard notation, standard patterns, and the use of the Unified Process for the development life cycle of databases and data warehouses.

### 6 Acknowledgements

This research was supported in part by a grant from the Center for Research, College of Science and Health, and in part by the ART (Assigned Released Time for Research) program, Office of the Provost, William Paterson University of New Jersey.

### 7 References


