Abstract

Specification and design in relational databases largely use a domain-specific design approach (e.g., Entity-Relational Model) different from that used in software engineering. In software engineering, the use of UML (Unified Modeling Language) models, design patterns, and the Unified Process are the primary design and development methodology, especially in modern object-oriented software engineering. The application of generic software engineering design patterns to databases allows issues in logical design and in implementation of databases to be more easily connected with similar concerns in software engineering. It can leverage previous experience with software design, make knowledge from database discipline more immediately relevant with software engineering. In this research, we apply the generic software engineering design patterns to the design and implementation of relational databases.

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, 10], and design patterns [4, 6]. Since a database system is a large scale software program; the specification, design, and development of database system should follow the software development life cycle of any large scale software project. The goal of our research is to use 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 to the design and development of relational databases.

Generic software engineering design patterns can be categorized into categories exemplified by the GRASP (General Responsibility Assignment Software Patterns) [6] and GoF (Gang-of-Four) [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. This paper focuses on the application of GRASP patterns to relational databases. Our research on the applicability of GoF patterns to databases can be found in [7].

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 [11] 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.

4 APPLICATION TO RELATIONAL DATABASES

The GRASP patterns affect multiple levels of the relational database system. It includes data design, query design, DBMS logical table management, physical table management of FMS (File Management System), file implementation and distribution of tables in FMS, and query access planning. The patterns can also enforce policy and system constraints such as access and permissions, security, distribution, and integrity.

The description of the GRASP patterns as they apply to the design and development of relational databases is 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 [8]. If the reader is familiar with entity-relationship model, “class” can be replaced by “entity” in the following discussion.

4.1 GRASP – Creator

The Creator pattern states that B creates A if B contains A, B aggregates A, B has the initializing data for A, B records A, or B closely uses A [6]. This pattern can be applied to relational database as follows:

(1) If there is a creation relationship between two classes, or any of the relationships specified above, then these two classes can be modeled as an aggregate (a single class) or as a generalization / specialization relationship between B and A.

(2) In database implementation, the concept of Creator is related to locating and structuring access patterns and the responsibility for managing temporary tables.

This pattern is usually related to Low Coupling (eliminate extra relationship) and Polymorphism.

4.2 GRASP – Information Expert

The Information Expert pattern assigns a responsibility to the information expert – the class that has the information necessary to fulfill the responsibility [6]. This pattern can be applied to relational database as follows:

(1) Assign relevant attributes to the class that belong to (describes) that class or to the super-class of that class. This will likely create relational tables that conform to the 2nd or the 3rd normal form.

(2) Assign attributes related to a relationship to the association class, even if valid to place them with one of the participants.

(3) If a relationship has substantial attributes, think about creating a separate association table (even for a 1-to-1 or 1-to-many relationship).

(4) The concept of this pattern applies to writing queries based on an association table with the relationship as the primary focus – structure using foreign key field equality.

This pattern is usually related to Low Coupling and High Cohesion.

4.3 GRASP – Low Coupling

The Low Coupling pattern assigns responsibilities so that (unnecessary) coupling remains low. Use this principle to evaluate alternatives [6]. This pattern can be applied to relational database as follows:

(1) Coupling among classes is visible as relationships. Relationship table can be eliminated (Low Coupling) for 1-to-1 relationship or 1-to-many relationship during implementation. However, if the relationship has many associated attributes (an association class), then implementation of the relationship table is desirable (Information Expert).

(2) Where possible, localize constraints, preferring single table to multi-table, row to table, field to row.

This pattern is usually related to Information Expert and High Cohesion.

4.4 GRASP – High Cohesion

The High Cohesion pattern keeps objects focused, understandable, and manageable, and as a side-effect, supports Low Coupling [6]. This pattern can be applied to relational database as follows:

(1) In designing a class or a table, keep relevant information/attributes in a table (Information Expert). Eliminate unnecessary join operation during execution and relationships to other classes (Low Coupling).

(2) Design tables balanced by performance (reduce join operation improves performance) and normal forms (too many related attributes in one table tends to violate 2nd or 3rd normal form).

(3) If a class has multiple roles, consider separating relevant attributes into separate tables to support later decoupling of roles.

This pattern is usually related to Information Expert and Low Coupling.
4.5 GRASP – Controller

The *Controller* is the object beyond the UI (User Interface) layer that receives and coordinates (controls) a system operation [6]. This pattern can be applied to relational database as follows:

1. In database design, the virtual table behind the UI layer supports the creation of views to present relevant information for users’ needs.
2. A *Controller* is (often) an object between the UI layer and the domain layer. Likewise in database, the view is a virtual table between the UI layer and the domain layer of base tables.
3. In a distributed system, or a multi-disk system, there will be a *Controller* in the file implementation layer to translate logical table access to physical table access.

This pattern is usually related to *Pure Fabrication* and *Indirection*.

4.6 GRASP – Polymorphism

The *Polymorphism* pattern assigns responsibility to the classes for which the behavior varies [6]. This pattern can be applied to relational database as follows:

1. The concept of this pattern applies to attribute inheritance in data modeling.
2. The concept applies to “is-a” relationship between classes, the generalization / specialization, or the super-class / sub-class relationship.
3. *Polymorphism* allows constraints or other safeguards to characterize sub-classes (and affect update methods) such as value, unique, access, or row constraints; or security restrictions.
4. In physical database design, DBMS access routines call FMS access routines; the FMS has the responsibility for knowing the file type and layout and for selecting or modifying the access routine (*Polymorphism* in combination with the *Protected Variation* pattern).

This pattern is usually related to *Creator* and *Protected Variation*.

4.7 GRASP – Pure Fabrication

The *Pure Fabrication* pattern assigns a highly cohesive set of responsibilities to an artificial class that does not represent a domain object [6]. This pattern can be applied to relational database as follows:

1. The catalog of a database is a *Pure Fabrication*.
2. The concept applies to the keeping of archival and bookkeeping records in a data warehouse.
3. The creation of views for users’ convenience, or performance improvement (avoid join operation during execution).
4. The creation of a table with attributes that belong to many other domain-objects/base-tables for users’ requirements and performance reasons (e.g., in a distributed environment).
5. Detaching distribution, locking, security, and access rules into a table wrapper. Alternatively, these may be placed into the catalog – use *Controller* to access.

This pattern is usually related to *Controller*, *Low Coupling*, and *high Cohesion*.

4.8 GRASP – Indirection

The *Indirection* pattern assigns responsibility to avoid direct coupling [6]. This pattern can be applied to relational database as follows:

1. A level of *Indirection* (an additional table) is created to interface with other databases to achieve *Low Coupling*.
2. Directly related to creation of indices, especially to support set unions.

This pattern is usually related to *Pure Fabrication* and *Low Coupling*.

4.9 GRASP – Protected Variations

The *Protected Variations* pattern assigns responsibility to classes so that the variations or instability in these classes do not have an undesirable impact on other classes [6]. This pattern can be applied to relational database as follows:

1. The concept is ingrained in relational model. For example, data independence, standardized SQL.
2. Table and query realization (in combination with *Indirection*) for union types protects against independent changes in factors or problems arising from further sub-classing.
3. “Interface” classes used to store data from other databases.

This pattern is usually related to *Indirection* and *Polymorphism*.

5 CONCLUSION AND FUTURE RESEARCH

In this paper, we show many examples of applying the GRASP design patterns to the design and development of relational databases. As the examples show, it is a sound concept to use these generic software engineering design patterns for relational databases.

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 [7]. In future research, we will combine generic patterns
(such as the GoF and GRASP patterns) with the domain-specific patterns such as [9], to demonstrate the applicability in databases. We are in the process of identifying and organizing more examples. We will also extend design pattern applicability to data warehouses in our future research.

The use of UML and generic software engineering design patterns for relational database is only part of the goal to eventually have standard notation, standard patterns, and 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


