УДК 004.9:378:796

ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ В ПРАКТИКЕ ВЫСШЕЙ ШКОЛЫ

[1]Ржавин В.В., [1]Обломов И.А., [2]Фадеева К.Н.

[1]ФГБОУ ВО «Чувашский государственный университет им. И.Н. Ульянова», Чебоксары, e-mail: grzhavv@gmail.com, ra4yes@rambler.ru;

[2]ФГБОУ ВО «Чувашский государственный педагогический университет им. И.Я. Яковлева», Чебоксары, e-mail: fadeevakn@mail.ru

Актуальность данной статьи обусловлена необходимостью анализа ошибочных решений при проектировании баз данных на концептуальном и логическом этапе. Проектирование схем реляционных баз данных является одним из наиболее важных и ответственных этапов в процессе разработки информационных систем, систем автоматизированного управления и других компьютерных систем, информационная поддержка которых основана на использовании баз данных. В статье рассмотрен личный опыт проектирования реляционных баз данных в практике высшей школы и сделан анализ типичных ошибок студентов. Проектирование реляционных баз данных часто вызывает серьезные затруднения у начинающих проектировщиков. На основании опыта зарубежных и отечественных авторов была разработана система шаблонов, охватывающая различные аспекты проектирования и включающая в себя имя, краткое описание решаемой шаблоном проблемы, типичные ошибки, решение проблемы. Внедрение описанных шаблонов в учебный процесс показало их эффективность. Разработка шаблонов проектирования реляционных баз данных и дальнейшее их применение при реализации курсового проекта позволяет обучающимся высшей школы избежать типичных ошибок, а преподавателям ускорить проверку студенческих работ и избавиться от излишних разъяснений.

Ключевые слова: реляционные базы данных, проектирование баз данных, паттерны проектирования, проблемы и ошибки проектирования баз данных

USING RELATIONAL DATABASE DESIGN TEMPLATES IN THE PRACTICE OF HIGHER EDUCATION

[1]Rzhavin V.V., [1]Oblomov I.A., [2]Fadeeva K.N.

[1]I. Ulyanov Chuvash State University, Cheboksary, e-mail: grzhavv@gmail.com, ra4yes@rambler.ru;

[2]I. Yakovlev Chuvash State Pedagogical University, Cheboksary, e-mail: fadeevakn@mail.ru

The relevance of this article is due to the need to analyze erroneous decisions when designing databases at the conceptual and logical stage. The design of relational database schemas is one of the most important and responsible stages in the development of information systems, automated control systems and other computer systems, the information support of which is based on the use of databases. The article examines the personal experience of designing relational databases in the practice of higher education and analyzes typical mistakes of students. Relational database design often causes serious difficulties for novice designers. Based on the experience of foreign and domestic authors, a template system was developed that covers various aspects of design, and includes a name, a brief description of the problem solved by the template, typical errors, and a solution to the problem. The implementation of the described templates in the educational process has shown their effectiveness. The development of relational database design templates and their further application in the implementation of the course project allows higher school students to avoid typical mistakes, and teachers to speed up the verification of student papers and get rid of unnecessary explanations.

Keywords: relational databases, database design, design patterns, problems and errors in database design

Благодаря информатизации образования преподавателям предоставляются возможности для внедрения в образовательный процесс новых методик, которые направлены на качественную организацию самостоятельной работы студентов при курсовом проектировании, в рамках которого необходимо разработать информационную систему [1]. В основе проектирования баз данных лежит требование адекватности базы данных предметной области. Адекватность предполагает, что модель данных без искажений передает взаимосвязи информации в той части реального мира, которая подлежит информатизации. Причем совершенно ясно, что речь может идти только об определенном подмножестве как информации, так и информационных связей, которые необходимы для решения поставленной задачи [2].

Цель исследования – описать созданные нами по аналогии с шаблонами при объектно-ориентированном проектировании программ шаблоны проектирования реляционных баз данных. Ни в коей мере не претендуя на полноту, они объединяют в себе как личный, так и общественный опыт проектирования реляционных баз данных, а также анализ типичных ошибок студентов в ходе выполнения курсового проекта и возникающих при этом проблем.

Актуальность настоящего исследования обусловлена фиксацией ошибочных реше-

ний в ходе проверки студенческих работ, которые повторялись от студента к студенту, причем как на этапе концептуального, так и логического проектирования. Это послужило основной причиной для формирования набора готовых схем-решений, которые помогли бы как студенту, так и преподавателю, первому – избежать типичных ошибок, второму – ускорить проверку студенческих работ и избавиться от излишних разъяснений.

Материалы и методы исследования

Разработке подходов и алгоритмов проектирования реляционных баз данных, а также всем аспектам данных процессов посвящено достаточно большое количество работ. Основные теоретические положения данного направления были заложены в 1970–1980-е гг. Наиболее известными авторами данной тематики являются Т. Конноллли, К. Бегг, Б. Карвин, Э. Кодд, К. Дейт, Д. Мейер, Х. Дарвен, В.В. Бойко, Д.В. Гмарь и др. [2]. В работе использована совокупность концепций, методов и приемов, основанных на математическом аппарате реляционной алгебры, направленных на получение качественной схемы реляционной базы данных.

Результаты исследования и их обсуждение

В результате обобщения опыта зарубежных и отечественных авторов [3–5] была разработана система шаблонов, охватывающая различные аспекты проектирования:

1. Шаблон «Суперкласс – Подкласс».
2. Шаблон «Объект–Атрибут–Значение».
3. Шаблон «Полиморфные ассоциации».
4. Шаблоны 1:M:
   a. Шаблон «Заказ»;
   b. Шаблон «Экземпляр»;
   c. Шаблон «Группа»;
   d. Шаблон «Работа».
5. Шаблоны истории изменений объектов:
   a. Шаблон «График работы»;
   b. Шаблон «Прейскурант».
6. Шаблоны реализации древовидных структур:
   a. Шаблон «Ссылка на предка»;
   b. Шаблон «Транзитивное замыкание»;
   c. Шаблон «Каталог».
7. Шаблоны реализации справочников.
8. Шаблон «Константа».

Приведенные шаблоны разработки строятся по одной схеме и включают в себя:

− имя;

− краткое описание решаемой шаблоном проблемы;

− типичная ошибка;

− решение проблемы.

Ниже при описании шаблонов говорится об ошибках проектирования. Необходимо уточнить, что понимается под ошибками. Это не фатальная ошибка, которая исключает возможность использования баз данных. Чаще всего это решение, которое ведет к неэффективной работе по обработке данных, то есть затрудняет обработку данных и создает проблемы. Поэтому иногда говорят не об ошибках, а об антипаттернах, антишаблонах. То есть о шаблонах, которые используют, но их не всегда можно рекомендовать [6].

Для иллюстрации разработанных патернов ниже рассматриваются шаблоны «Суперкласс – Подкласс» и «Объект – Атрибут – Значение».

Шаблон «Суперкласс – подкласс»

Описание проблемы. В обычной таблице все строки представляют экземпляры сходных объектов. Разные же наборы атрибутов представляют разные типы объектов, так что они принадлежат разным таблицам. Тем не менее в современных моделях программирования разные типы объектов могут быть связаны друг с другом путем расширения одного и того же базового типа. В объектно-ориентированном проектировании эти объекты считаются экземплярами одного базового типа, а также экземплярами их соответствующих подтипов. Чтобы упростить сравнения и расчеты по нескольким объектам, желательно было бы хранить такие объекты разных типов в виде строк в одной таблице БД. Также необходимо хранить их индивидуальные атрибуты, отсутствующие в базовом типе.

Рассмотрим следующий пример. Пусть есть система отслеживания программных ошибок (Bug Tracking System – BTS). При тестировании программного продукта тестировщик записывает в базу данных этой системы свои комментарии [7]. Комментарии могут касаться проблем (Issue) двух типов:

1. Bug – баг, ошибка, т.е. расхождение между фактическим и ожидаемым результатом.

2. Feature Request – запрос об улучшении, запрос новой функциональности.

Каждая тема для обсуждения (*Bug* или *Feature Request*) может иметь несколько комментариев от одного или разных заинтересованных лиц (авторов) [8].

Данные комментарии имеют как общие атрибуты, так и индивидуальные. На рис. 1 приведена диаграмма классов, где класс **Проблема** содержит общие атрибуты, наследуемые классами **Ошибка** и **Функция**. Как видно из рисунка, подклассы содержат и собственные атрибуты.

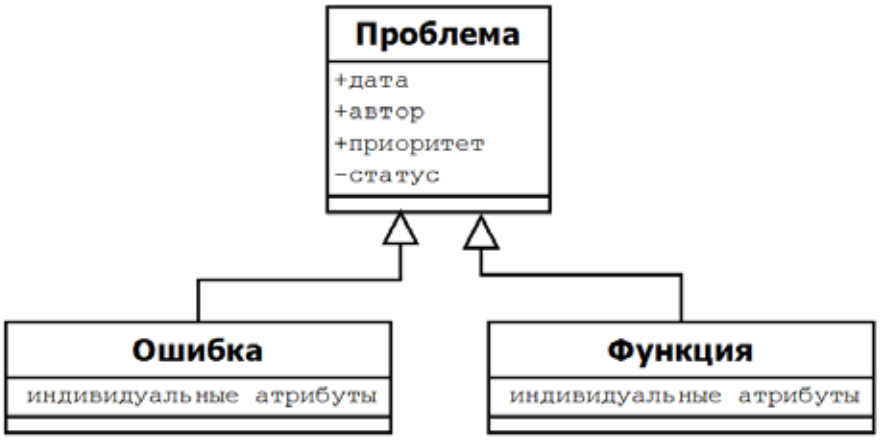

Рис. 1. Суперкласс и подклассы

Как отобразить структуру классов приложения в структуру баз данных? Проблема проектирования БД состоит в том, что к одной записи «Проблема» надо привязать записи с отличающимися атрибутами.

Типичная ошибка. Решение «в лоб» предполагает размещение в одной таблице всех атрибутов с использованием NULL значений. Как показано ниже, это не лучшее решение. Также можно использовать шаблон EAV (Сущность – Атрибут – Значение) [6].

Решение. Решение данной задачи с использованием классического подхода (в отличие от EAV) позволит моделировать данные более легко и с большей гарантией целостности данных.

Решение состоит в моделировании подтипов. Существует несколько способов хранения таких данных:

1. Наследование одиночной таблицы.
2. Наследование конкретной таблицы.
3. Наследование таблицы классов.
4. Слабоструктурированные данные.

Большинство решений работает лучше всего тогда, когда существует конечное число подтипов и известен атрибут каждого подтипа. Какое решение будет оптимальным для применения, зависит от того, как предполагается запрашивать данные, поэтому решение о структуре следует принимать по каждому конкретному случаю.

Наследование одиночной таблицы

Это самая простая структура, обеспечивающая хранение всех связанных атрибутов в одной таблице с отдельными столбцами для каждого атрибута, существующего в каком-либо типе. Один атрибут отводится для определения подтипа заданной строки. В примере этому атрибуту присвоено имя Тип проблемы (рис. 2).

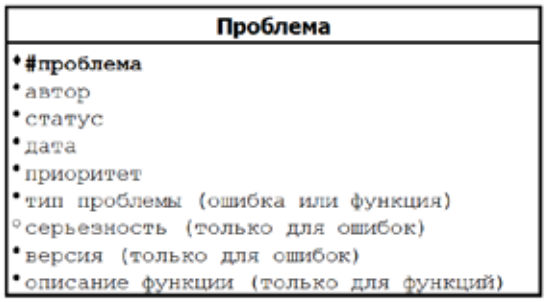

Рис. 2. Атрибуты таблицы «Проблема»

Некоторые атрибуты являются общими для всех подтипов. Многие атрибуты зависят от подтипов, и этим столбцам должны присваиваться значения NULL во всех строках, хранящих объекты, к которым не применяется этот атрибут. При данном способе хранения в приложении придётся вручную отслеживать, какие атрибуты применимы для каждого подтипа. Обратите внимание на столбец дискриминатора «тип проблемы», который содержит значение, определяющее, какому классу принадлежит каждая запись.

Наследование конкретной таблицы

Другое решение заключается в создании отдельной таблицы для каждого подтипа. Общая таблица не создается. При этом во всех таблицах содержатся одни и те же атрибуты, которые являются общими для базового типа, а также соответствующий атрибут для подтипа.

Преимущество данного подхода по сравнению с подходом *Наследование одиночной таблицы* заключается в том, что индивидуальные атрибуты разнесены по разным таблицам, и потому отсутствует необходимость в дополнительном атрибуте для определения подтипа в каждой из та-

блиц. Однако при этом трудно отличить общие атрибуты от атрибутов, характерных для подтипов. К тому же, если добавить новый атрибут в набор общих атрибутов, необходимо изменить все таблицы подтипов.

Когда требуется найти все объекты независимо от их типов, задача усложняется, если каждый подтип хранится в отдельной таблице [9]. Для этого придется объединить в запросе таблицы, отфильтровав только общие атрибуты.

Структура *Наследование конкретной таблицы* оптимальна в том случае, когда редко возникает необходимость в запросе одновременно всех типов.

Наследование таблицы классов

Третье решение имитирует наследование примерно так, как если бы таблицы были объектно-ориентированными классами. Сначала создается одна таблица для базового типа, содержащая атрибуты, общие для всех подтипов. Затем для каждого подтипа создается еще одна таблица с первичным ключом, который служит также в качестве внешнего ключа для базовой таблицы. Такой подход известен под именем ***Суперкласс – Подкласс***.

Шаблон«Объект–Атрибут–Значение»

Вокруг этого шаблона сломано немало копий. Некоторые авторы (Б. Карвин, Т. Коннолли, К. Бегг) относят его даже к антишаблону, и для этого есть серьезные основания.

Описание проблемы. Использование этого шаблона касается двух проблем:

1. Разный состав атрибутов для объектов одной сущности (выше эта проблема обозначена как «Суперкласс – подкласс»).

2. Изменяющийся во времени состав атрибутов (динамические атрибуты).

Если первая проблема имеет классическое решение в виде шаблона «Суперкласс – подкласс», то вторая требует иного подхода.

При разработке программных систем часто стремятся добиться расширяемости. Такое программное обеспечение должно адаптироваться к будущим потребностям с минимальным перепрограммированием или вовсе без дополнительных трудозатрат.

Каждый объект имеет свой набор атрибутов. Количество таких атрибутов не всегда устойчиво и может меняться со временем. Достаточно часто заранее бывает трудно определить, какие из атрибутов будут использоваться в проектируемой базе данных. Добавление новых атрибутов влечет за собой изменение структуры базы данных, что приведет к необходимости вводить изменения в транзакции, приложения, формы, отчеты. В этих условиях иногда допустимо использовать шаблон «Объект – Атрибут – Значение».

Типичная ошибка. Применение самого шаблона в данном случае неоднозначно, поэтому правильней говорить об оценке игнорирования указанных выше двух проблем и размещении всех атрибутов в одной таблице со всеми вытекающими отсюда последствиями.

Решение. Создание дополнительной таблицы, где атрибуты хранятся в виде строк. Каждая строка в такой таблице атрибутов содержит три столбца: ***Entity (***Объект), ***Attribute*** (Атрибут), ***Value*** (Значение).

Данная структура называется «Entity – Attribute – Value», или сокращенно EAV. EAV также известна как *вертикальная модель базы данных* и *открытая схема.*

Получается универсальная структура, позволяющая описать и сохранять объекты с отличающимися схемами атрибутов. Однако за универсальность придется платить усложнением обработки.

Добавление таблицы АТРИБУТ позволяет получить такие преимущества: для поддержки новых атрибутов число столбцов не увеличивается; исключается хаос столбцов, содержащих NULL в столбцах, где этот атрибут неприменим.

Ниже приведена таблица (АТРИБУТ) с описанием ошибки, идентифицируемой по значению 1234 её первичного ключа.

Объект	Атрибут	Значение
1234	Тип проблемы	Ошибка
1234	Дата	12.11.15
1234	Статус	Новая
1234	Приоритет	Высокий
1234	Краткое описание	Сохранение не работает
1234	Автор	Сомов М.
1234	Серьёзность ошибки	Потеря функциональности
1234	Номер версии тестируемой программы	1.2

Все атрибуты объекта (в данном случае объекта 1234), как общие, так и частные, описаны в единой таблице. Как видно, структура получилась достаточно простой. Однако простота структуры БД не компенсирует усложнение работы с ней.

Когда атрибут объекта становится столбцом в таблице, мы описываем его имя, тип данных, размер, формат и прочее. Атрибут становится элементом метаданных. А это значит – контроль над ним со стороны системы управления базами данных. В модели EAV мы теряем этот контроль, то есть возможность декларативной поддержки кор-

ректности данных. Кроме того, значением атрибута будет являться только строковый тип, а это значит, что придётся преобразовывать типы. Помимо этого, у нас могут появиться повторы для названий атрибутов.

В модели EAV невозможно создание обязательных атрибутов. В обыкновенной структуре базы данных для создания обязательных атрибутов достаточно установить обязательный столбец, объявив его как NOT NULL. В EAV-структуре это сделать невозможно, потому что каждому атрибуту соответствует не столбец, а строка в таблице. Тогда для проверки существования значения обязательного атрибута необходима проверка существования строки с именем этого атрибута со значением в столбце *Значение*. Средства SQL такое ограничение не поддерживают. Поэтому необходимо написать приложение для принудительного ввода ограничения.

При использовании данного шаблона придется пожертвовать слишком многими функциями, которые признаны сильными сторонами реляционной парадигмы. Поэтому использование EAV ограничено и допустимо для поддержки динамических атрибутов в некоторых программах.

Что дает применение данного шаблона? На первый взгляд, кажется, что мы здесь скорее проиграли, чем выиграли. Действительно, вместо одной таблицы мы получили три. Но выигрыш станет заметен, как только мы захотим изменить состав атрибутов объекта. Например, добавить цвет фона, количество скоростей и пр. Действительно, при стандартном подходе нам придется менять структуру данных, в предлагаемом же варианте следует лишь добавить записи в таблицу Параметр. Это и есть главное достоинство такого шаблона.

EAV находит применение благодаря высокой масштабируемости, которую не способна дать обычная нормализованная структура базы данных. Разработчики могут добавлять новые атрибуты к любой сущности (товару, категории, покупателю, заказу и пр.) без каких-либо модификаций структуры базы данных [10].

В целом же применение EAV в реляционной базе данных не всегда можно оправдать. Если есть необходимость в управлении нереляционными данными, возможным решением будет использование нереляционной технологии (MongoDB, Redis). Недостатки подхода EAV в реляционной базе данных характерны и для альтернативных подходов [4]. Когда метаданные изменчивы, трудно формулировать простые запросы. Приложения тратят немало времени на раскрытие структуры данных и их адаптацию под обнаруженные структуры.

Чтобы принять грамотное решение об использовании данного шаблона, необходимо четко представлять все преимущества и недостатки данного подхода.

Заключение

Разработанные шаблоны покрывают почти все потребности студентов при курсовом проектировании. Ограничения на объем статьи не позволяют описать все шаблоны, но мы полагаем, что смогли дать представление о возможностях шаблонов в проектировании реляционных баз данных. Опыт применения таких шаблонов показал эффективность их использования в учебной практике.

Список литературы

1. Кара-Ушанов В.Ю. SQL – язык реляционных баз данных: учебное пособие. Екатеринбург: Изд-во Урал. ун-та, 2016. 156 с.

2. Баранчиков А.И. Методы и модели синтеза информационных структур хранения на основе результатов извлечения закономерностей в актуальных данных предметных областей: дис. … канд. техн. наук: 05.13.17 Теоретические основы информатики. Москва, 2014. 395 с.

3. Эмбер С.В., Садаладж П.Д. Рефакторинг баз данных: эволюционное проектирование / пер. с англ. М.: ООО «И.Д. Вильямс», 2016. 672 с.

4. Гмарь Д.В., Игнатова Ю.А., Цуранов Э.В., Шахгельдян К.И. Методы работы с вертикальной моделью данных // Информационные технологии и вычислительные системы. 2015. № 2. С. 1–28.

5. Карвин Б. Программирование баз данных SQL. Типичные ошибки и их устранение / Пер. с англ.: учебное пособие. М.: Изд. Рид Групп, 2012. 336 с.

6. Ржавин В.В. Шаблоны проектирования реляционных баз данных // Состояние и перспективы развития ИТ-образования: сб. докл. и науч. ст. Всерос. науч.-практ. конф. Чебоксары: Изд-во Чуваш. ун-та, 2019. С. 217–221.

7. Коннолли Т., Бегг К. Базы данных: проектирование, реализация, сопровождение. Теория и практика. 3-е изд.: пер. с англ.: учебное пособие. М.: Изд. Дом «Вильямс», 2017. 1440 с.

8. Ржавин В.В. Шаблон проектирования «Суперкласс-подкласс» реляционных баз данных. Информатика и вычислительная техника: сб. науч. тр. Чебоксары: Изд-во Чуваш. ун-та, 2021. С. 214–220.

9. Структура наследования – ~Разное .:: CodingRUS ::. программирование по-русски на Delphi, C++, PHP, Prolog, GPSS. URL: http://codingrus.ru/readarticle.php?article_id=5191 (дата обращения: 12.10.2022).

10. Андреева К.Ю. Построение баз данных на основе EAV-технологий // Актуальные направления научных исследований XXI века: теория и практика. 2015. Т. 3. № 5–2 (16–2). С. 154–157. DOI: 10.12737/15994.