УДК 004.65
DOI: http://dx.doi.org/10.21686/1818-4243-2024-6-67-21

С.М. Щербаков, К.Х. Калугян, А.В. Абрамова, В.Ю. Гречкина

Ростовский государственный экономический университет (РИНХ), г. Ростов-на-Дону, Россия

Антипаттерны в проектировании баз данных

Цель работы *является адаптация метода антипаттернов для повышения эффективности обучения студентов направлений «Прикладная информатика», «Информационные системы и технологии», «Программная инженерия» принципам и техникам проектирования реляционных баз данных в рамках соответствующих дисциплин и практик.*

Обучение проектированию баз данных является важной составляющей формирования специалиста-разработчика программного обеспечения. Совокупность принимаемых при проектировании решений должна опираться на общепринятые правила, принципы и рекомендации, а также на имеющийся опыт создания и эксплуатации информационных систем, трудновоспроизводимый в условия аудитории вуза.

Материалы и методы. *Повысить эффективность обучения студентов проектированию баз данных позволит подход, ориентированный на концепцию антипаттернов – неудачных распространенных проектных решений. Понятие антипаттерна получило широкое распространение вслед за широким использованием понятия паттерна, то есть распространенного типового решения, вписанного в контекст решаемой проблемы. Как понятие паттерна, так и понятие антипаттерна первоначально были ориентированы на объектно-ориентированное программное обеспечение, но в дальнейшем получили признание во всех областях информационных технологий.*

В работе предложена схема описания антипаттерна, включающая название, краткую характеристику, возможные причины появления, список недостатков применения при построении реляционных баз данных, концептуальная модель в виде ER-диаграммы (диаграмма сущность-связь), пример соответствующей структуры базы данных, способы ухода от антипаттерна, а также возможные исключения, когда применение именно такого проектного решения при создании базы данных может быть оправдано.

Результаты исследования. *На основании учебной, научной и технической литературы и периодики, а также на основе анализа имеющейся практики проектирования информационных систем составлен и описан каталог распространенных антипаттернов проектирования реляционных баз данных. Примеры антипаттернов: ««Винегрет» имен», «Таблицы-дубли», «Ложная абстракция», «Неатомарные реквизиты», «Столбцы-дубли», «Изменение данных в рабочем порядке (временные поля)», «Параллельные связи», ««Божественная» таблица», «Спекулятивное изменение метаданных» и др.*

При этом часть паттернов позволила проиллюстрировать общие принципы проектирования реляционных баз данных, например, такой как «Никакое изощренное программирование не способно исправить фатальные недостатки архитектуры». Показано, что имеющиеся неудачные решения зачастую обусловлены определенными предпосылками, например, необходимостью выполнить hotfix, то есть «временное» решение, или преждевременной попыткой повысить производительность системы, либо желанием получить некоторый отчет одним запросом, избежав лишних соединений и т.д. Помимо технических аспектов показана актуальность корректной и единообразной системы наименований, отсутствие которой затрудняет сопровождение и развитие системы и ведет к лишним затратам.

Предложенный подход был апробирован при обучении студентов дисциплинам «Базы данных», «Проектирование баз данных», «Архитектура информационных систем». Прошедшие обучение студенты улучшили понимание принципов и практик проектирования структуры баз данных.

Заключение. *Применение предложенного каталога антпаттернов в учебном процессе вуза позволяет ускорить обучение молодых специалистов приемам проектирования структуры реляционных баз данных, избежать распространенных ошибок при создании информационных систем, а также повысить общую культуру проектирования.*

Ключевые слова: *база данных, СУБД, SQL, проектирование, антипаттерн, учебный процесс.*

Sergey M. Shcherbakov, Karine K. Kalugyan, Anna V. Abramova, Vera Y. Grechkina

Rostov State University of Economics (RINH), Rostov-on-Don, Russia

Antipatterns in Database Design

The purpose of the study *is to adapt the antipattern method to improve the effectiveness of teaching students in the fields of "Applied Informatics", "Information Systems and Technologies", "Software Engineering" to the principles and techniques of relational database design within the framework of relevant disciplines and practices. Training in database design is an important component of the formation of a specialist software developer. The totality of decisions taken in the design process should be based on generally accepted rules, principles and recommendations, as well as on the existing experience in creating and operating information systems, which is difficult to reproduce in the conditions of the university audience.*

Materials and methods. *An approach focused on the concept of antipatterns – unsuccessful common design solutions - will allow increasing the effectiveness of students' training in database design. The antipattern concept has become widespread following the widespread use of the concept of a pattern, that is, a common standard solution inscribed in the context of the problem being solved. Both the concept of a pattern and the concept of an antipattern were initially focused on object-oriented software, but later gained recognition in all areas of information technology.*

The paper proposes a scheme for describing the antipattern, including the name, a brief description, possible causes of occurrence, a list of disadvantages of application in the construction of relational databases, a conceptual model in the form of an ER-diagram (entity-relationship diagram), an example of the appropriate database structure, ways to avoid antipatterns, as well as possible exceptions when using just such a design decision when creating a database may be justified.

Research results. *Based on educational, scientific and technical literature and periodicals, as well as on the basis of an analysis of the existing practice of designing information systems, a catalog of common antipatterns for designing relational databases has been compiled and described. Examples of antipatterns: "Jumble of names", "Duplicate tables", "False abstraction", "Non-atomic attributes", "Duplicate columns", "Changing data in due course (temporary*

fields)”, “Parallel connections”, ““Divine” table”, “Speculative metadata modification”, etc.
At the same time, some of the patterns made it possible to illustrate the general principles of relational database design, for example, such as “No sophisticated programming is capable of correcting fatal architectural flaws”. It is shown that the existing unsuccessful solutions are often due to certain prerequisites, for example, the need to perform a hotfix that is, a “temporary” solution, or a premature attempt to improve system performance, or the desire to get some report with one request, avoiding unnecessary connections, etc. In addition to the technical aspects, the relevance of a correct and uniform naming system is shown, the absence of which makes it difficult to maintain and develop the system and leads to unnecessary costs.
The proposed approach was tested when teaching students the disciplines “Databases”, “Database design”, “Architecture of information systems”. The students who completed the training improved their understanding of the principles and practices of database structure design.
***Conclusion.** The use of the proposed catalog of antipatterns in the educational process of the university makes it possible to accelerate the training of young specialists in the techniques of designing the structure of relational databases, avoid common mistakes when creating information systems, and improve the overall design culture.*

***Keywords**: database, DBMS, SQL, design, antipattern, learning process.*

Введение

База данных является ядром любой информационной системы. В настоящее время, несмотря на прогресс NoSQL баз данных, таких как MongoDB или Cassandra, реляционные базы данных по-прежнему доминируют в задачах создания информационных систем. Развитие ORM-фреймворков создает дополнительный слой объектов программного обеспечения, но никак не отменяет необходимости проектирования структуры базы данных, которую эти объекты представляют.

Задача проектирования базы данных не является простой и предполагает комбинирование теоретических знаний, практических навыков и профессионального опыта [1]. При этом ошибки проектирования, будучи совершаемыми на ранних этапах построения информационной системы, могут вести к значительным техническим сложностям и расходам в дальнейшем.

В отечественной [2-3] и зарубежной [4] литературе значительное внимание уделяется проблемам обучения проектированию баз данных. При этом выделяется необходимость комбинировать теоретическое обучение и практические задания, разбор примеров, выполнение самостоятельных заданий, применение приближенных к практике кейсов. Одним из важных элементов учебного процесса (наряду с теорией и самостоятельным «набиванием шишек») является и ознакомление с зафиксированным опытом индустрии.

В сфере информационных технологий широко распространены концепции паттерна и антипаттерна. Идея паттерна (типового проверенного решения, вписанного в контекст решаемой проблемы) была выдвинута в сфере архитектуры и дизайна [5]. Далее усилиями «Банды четырех» [6] концепция паттернов была перенесена в сферу информационных технологий, точнее в область проектирования объектно-ориентированного программного обеспечения.

Антипаттерн – это также распространенное решение, но решение неудачное, влекущее некоторый негативный эффект [7]. Не стоит путать антипаттерн с ошибкой, разработанная с применением антипаттерна система будет работать, но будет иметь проблемы с обслуживанием, развитием, масштабированием.

Хорошим примером антипаттерна может служить выключатель, расположенный позади открывающегося дверного полотна. В конечном счете при должной сноровке можно добраться до выключателя и включить свет.

В дальнейшем успех концепций паттерна и антипаттерна обусловил их перенос и на смежные области, включая управление проектами, дизайн интерфейсов и т.д. [8–10].

Материалы и методы

Антипаттерны проектирования в базах данных (БД) – проблемные шаблоны, которые могут возникать в процессе разработки БД и приводить к негативным последствиям для их функциональности, производительности и обслуживания. В отличие от паттернов проектирования, которые представляют собой устоявшиеся решения для типичных проблем при проектировании, антипаттерны выступают как распространенные неудачные решения, которые могут возникнуть в процессе разработки БД.

Суть антипаттернов в проектировании состоит в возможности их определения и предотвращения, а также в улучшении понимания проблем, которые могут возникнуть в процессе создания БД. Осознание и применение таких проблемных шаблонов позволяет разработчикам и администраторам избежать типичных ошибок, повысить производительность и обеспечить более надежное функционирование БД.

Обратим внимание, что цель составления каталога антипаттернов не заключается исключительно в недопущении применения неудачных практик, но также в внедрении в учебный процесс и общих принципов проектирования архитектуры, а также формирования культуры проектирования.

В работе мы проведем обзор известных антипаттернов в проектировании баз данных, а также рассмотрим предложе-

ния по их предотвращению и устранению. Источниками антипаттернов выступали [11-14]: техническая литература (отметим прежде всего книги Б. Карвина [15] и М. Блаха [16]), научная и профессиональная периодика, профессиональные форумы, а также личный опыт авторов в участии в проектах разработки баз данных, их экспертизы и оценивания.

Еще один аспект, над которым молодому инженеру стоит задуматься – распространенность антипаттернов. Если их часто применяют в реальных проектах, значит есть некоторая причина, толкающая разработчиков к этим решениям. Это может быть кажущаяся простота реализации или желание получить быстрое решение, которое в дальнейшем возможно будет улучшено. Иногда антипаттерны выступают в качестве временного hot-фикса, позволяющего сохранить работоспособность системы. В ряде случаев имеется стремление обеспечить выполнение какого-то распространенного запроса, а иногда – желание «сэкономить» число таблиц и т.д. Также возможна еще одна причина – устоявшиеся неудачная культура проектирования в той или иной фирме или подразделении.

При описании антипаттернов будем использовать следующую структуру (некоторые компоненты могу отсутствовать):

- названия антипаттерна (если антипаттерн ранее был описан в литературе, сохраняется оригинальное название);
- общая схема;
- пример антипаттерна;
- возможная причина появления;
- негативные последствия;
- способы решения, ухода от антипаттерна;
- возможные исключения, когда применение антипаттерна оправдано.

Обратим внимание, что описанные нами и другими авторами антипаттерны не стоит воспринимать догматично. Так, Б. Карвин относит решение Entity-Attribute-Value к антипаттернам [15], в то время как М. Блаха – к паттернам [16]. Конечное слово остается за инженером, осуществляющим проектирование.

Результаты исследования

1. Антипаттерны, связанные с недостатками структуры БД и проектирования таблиц

1.1. Неатомарные реквизиты.

Когда реквизит одной сущности разбит на несколько отдельных значений, это затрудняет выполнение запросов, обновление и поддержание целостности данных (рис. 1). Представим себе, что нам потребуется выбрать сотрудников, владеющих английским языком. Во-первых, мы получим неэффективный sql-запрос с полнотекстовым поиском, во-вторых, при заполнении таблиц мы не можем проконтролировать однообразие написания названий.

В этой ситуации следует пересмотреть структуру данных и объединить такие атрибуты в одно поле для обеспечения атомарности (рис. 3).

В целом, антипаттерн подводит нас к важному принципу: *все, что помещается в поле таблицы выводится из-под действия механизмов реляционной СУБД, что и лишает преимуществ ее использования.*

1.2. Столбцы-дубли.

Дублирование данных в разных столбцах приводит к избыточности и риску противоречий, затрудняет выполнение сложных запросов (рис. 2). SQL-запрос будет изобиловать связками OR. Также мы не сможем помешать пользователю внести два одинаковых значения в одну запись. Наконец, вполне вероятно появление необходимости очередного значения, которое не поместится в существующие столбцы таблицы.

Часто такое решение возникает как ответ на необходимость создать срочную «заплатку», «hotfix», которые, как

person	languages
Иванов И.И.	Англ., испанский, немецкий
Петров П.П.	Английский

Рис. 1. Антипаттерн «Неатомарные реквизиты»

Fig. 1. “Non-atomic attributes” antipattern

person	language	language2
Иванов И.И.	испанский	английский
Петров П.П.	английский	

Рис. 2. Антипаттерн «Столбцы-дубли»

Fig. 2. “Duplicate columns” antipattern

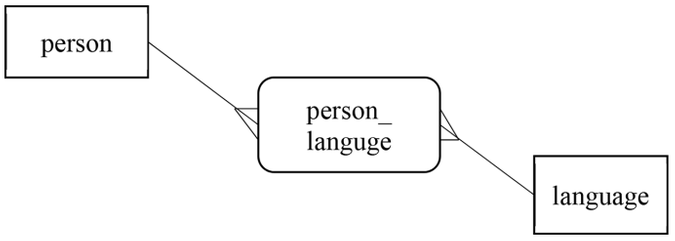

Рис. 3. Решение антипаттернов «Неатомарные реквизиты» и «Столбцы-дубли»

Fig. 3. Solution of the antipatterns “Non-atomic attributes” and “Duplicate columns”

клиент	город	товар	цена	количество
Иванов И.И.	Ростов	Стол	1000	12
Петров П.П.	Азов	Стол	1000	1
Петров П.П.	Азов	Стул	600	6

Рис. 4. Антипаттерн «Нарушение нормализации»
Fig. 4. "Violation of normalization" antipattern

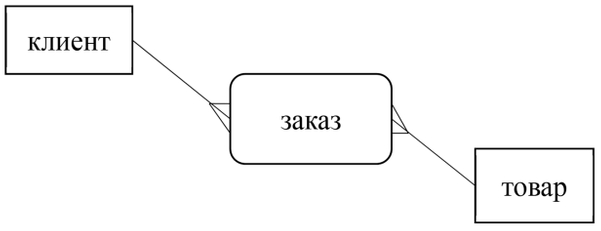

Рис. 5. Решение антипаттерна «Нарушение нормализации»
Fig. 5. Solution of the antipattern "Violation of normalization"

предполагаются, когда-нибудь в дальнейшем будут исправлены на более адекватное решение.

В таких случаях лучше пересмотреть структуру данных и создать связи между таблицами (рис. 3).

1.3.Нарушение нормализации.

Может привести к избыточности и потере целостности данных, трудностям при поддержке и изменении структуры таблиц [17] (рис. 4). Необходимо проанализировать структуру данных и привести ее к нормализованным формам (например, до третьей нормальной формы или нормальной форме Бойса – Кодда) (рис. 5).

Исключением является осознанная денормализация для задач OLAP (Online Analytical Processing). Необходимость многократного объединения таблиц может сделать запросы OLAP сложными и медленными. Осознанная денормализация позволяет хранить суммарные данные, предварительно вычисленные доли и другие агрегированные значения в одной таблице или структуре [18].

1.4. Таблицы-дубли [17].

Появление в базе данных нескольких таблиц с одинаковой структурой — это как лабиринт запутанных коридоров, где данные блуждают, теряются и дублируются без конца. Избыточность данных – это бомбы замедленного действия, которые рано или поздно разрушат целостность и согласованность базы данных (рис. 6).

Orders_2021
Orders_2022
Orders_2023

Рис. 6. Антипаттерн «Таблицы-дубли»
Fig. 6. "Duplicate tables" antipattern

Представим возможные последствия:

сложность ответа на вопрос, сколько заказов, например, поступило за последние 3 года;

возможные изменения структуры, которые потребуется дублировать и в старых таблицах;

сложность обработки граничных ситуаций, например, мы получили заказ в конце декабря, а выполнили в начале января.

В некоторых случаях создание таблиц-дублей используется как средство повышения производительности. В иных случаях целью может стать желание начинать каждый год «с чистого листа».

Чтобы избежать появления таблиц-дублей и связанных с ними проблем следует правильно определить сущности и атрибуты, соблюдать принципы проектирования и при необходимости применять встроенные средства повышения производительности.

Современные СУБД позволяют решать проблему производительности путем настройки, не затрагивая логический уровень. Можно настроить партиции (области физического хранения данных) так, чтобы старые данных располагались на отдельном жестком диске или на отдельном сервере. При этом прикладной программист будет видеть единую таблицу базы данных. Также для разных временных периодов может быть выбран разный способ хранения, часто более старые данные предпочитают хранить в колоночном виде (такую опцию предлагают отдельные современные СУБД, например, GreenPlum).

В целом, антипаттерн подводит нас к важнейшему принципу проектированию – к ориентации на правило Парето 80 / 20. Согласно этому принципу *необходимо избегать преждевременной оптимизации производительности и не пытаться ускорить выполнение всех запросов, а сначала выявить с помощью профилирования системы «узкие места» (которые и дают львиную долю задержек), и лишь потом сосредоточить усилия по оптимизации этих критических запросов.*

Также применение принципа Парето в контексте баз данных позволит:

- сосредоточить усилия на оптимизации тех запросов, которые формируют 80 % нагрузки на базу данных или выполняются 80 % времени, что поможет сделать работу с БД более эффективной;
- сконцентрироваться на индексировании тех столбцов, которые приводят к ускорению выполнения 80 % наиболее часто выполняемых запросов;
- сконцентрировать внимание на тех частях системы, которые являются «узким ме-

стом» и оказывают значительное влияние на общую производительность;

• кэшировать 20 % наиболее часто используемых данных, что может привести к улучшению производительности в 80 % случаев.

1.5. Бессодержательные имена.

Такое решение затрудняет понимание структуры и смысла таблиц и столбцов в них, создает сложности при сопровождении БД (рис. 7). Пример взят из реально существовавшего проекта.

table0045

field005	**field006**
Иванов И.И.	Ростов
Петров П.П.	Азов

Рис. 7. Антипаттерн «Бессодержательные имена»

Fig. 7. "Meaningless names" antipattern

Более информативные имена улучшают читаемость, понимание и поддержку кода, поэтому следует уделить этому аспекту достаточное внимание при создании программного обеспечения – разработать систему имен и придерживаться её.

При этом сама система может быть любой, но ее важно строго придерживаться.

1.6. «Винегрет» имен.

Смешанные или несогласованные способы именования элементов базы данных (рис. 8). Так же, как и антипаттерн «бессодержательные имена», может привести к путанице, увеличению сложности понимания кода и затруднению его сопровождения, что может подорвать продуктивность всей команды разработчиков. Если в базе уже существуют подобные нарушения, необходимо организовать рефакторинг.

Перечисленные нами два антипаттерна приводят к следующему принципу: *разработка является, в том числе, социально-экономической деятельностью.* А значит корректный выбор описательных имен, которые раскрывают истинный смысл таблиц и столбцов столь же важен, как и правильная структура базы данных. Необходимо внимание к затратам труда и понимание сложностей взаимодействия участников разработки [19].

	clients
Client_name	**idclient**
Иванов И.И.	1
Петров П.П.	2
	Spr_tovar
Goodname	**Kod_tov**
Стол	1
Стул	2

Рис.8. Антипаттерн «"Винегрет" имен»

Fig. 8. "Jumble of names" antipattern

2. Антипаттерны, связанные с жесткими и неподдающимися изменениям структурами данных.

2.1. ENUM hardcoding (жестко закодированные справочные значения).

Жесткое кодирование значений переменных (ENUM) делает программный код менее гибким и поддерживаемым при необходимости изменений [16] (рис. 9).

Недостатки данного антипаттерна включают в себя:

• сложность поддержки: жёстко закодированные перечисления усложняют сопровождение кода, поскольку любые изменения в перечислении потребуют обновления кода вручную;

• неочевидность: использование зашитых перечислений скрывает информацию о различных значениях, что делает код менее читаемым и понятным для других разработчиков;

• ограниченность возможностей: жесткое кодирование перечислений затрудняет добавление новых значений или модификацию существующих.

клиент	**статус**	
Иванов И.И.	4	1 – новый 2 – потенциальный ...
Петров П.П.	1	
Елизаров Е.Е.	4	

Рис. 9. Антипаттерн «ENUM hardcoding»

Fig. 9. "ENUM hardcoding" antipattern

Если перечисления могут изменяться динамически, их значения можно хранить в базе данных в справочнике, обеспечивая гибкость и удобство в обновлении (рис. 10).

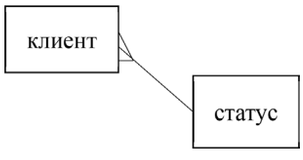

Рис. 10. Решение антипаттерна «ENUM hardcoding»

Fig. 10. Solution of the antipattern "ENUM hardcoding"

2.2. Ложная абстракция

Ситуация, когда в одной таблице размещены строки, относящиеся к двум или более различным понятиям, произвольно объединенных в одно целое (рис. 11). Как видим из рисунка, общего между «Категорией водителя» и «Категорией товара» лишь слово «Категория». А значит мы не сможем проконтролировать, чтобы в путевом листе вместо категории водителя не оказалась категория груза и наоборот. Либо такой контроль потребует лишних усилий программиста.

Код	**Категория**
1	Категория водителя А
2	Категория водителя Б
3	Категория водителя В
4	Категория груза 1
5	Категория груза 2

Рис. 11. Антипаттерн «Ложная абстракция»

Fig. 11. "False abstraction" antipattern

Вместо стабильной и надежной базы данных, ложная абстракция создает иллюзию устойчивости, которая может легко рухнуть под натиском изменений. Это может привести к катастрофическим сбоям и наводнениям данных, угрожая даже самому существованию системы.

Разнородные понятия следует разделять на отдельные таблицы-справочники, каждая из которых будет описывать конкретный тип данных.

2.3. Level Hardcoding [16] (жесткое кодирование уровней иерархии в таблице).

Жесткое привязывание логики или данных к определенному уровню или слою делает систему менее гибкой и расширяемой (рис. 12). Целесообразно разделение разнородных типов данных по разным таблицам в соответствии с их природой, чтобы каждая таблица содержала данные только определенного уровня или типа (рис. 13).

При использовании этого антипаттерна, база данных становится фрагментированной, как разбитое зеркало - каждый кусочек отражает что-то, но ни один не дает полной картины. Эта фрагментация приводит к потере целостности и недоступности полного контекста.

Обратим внимание, решение, представленное на рисунке 13 – это не единственный способ представления иерархий. Например, У. Карвин [15] критикует такой подход вводя для него антипаттерн «Наивная иерархия» и предлагает ряд альтернативных решений.

2.4. Полиморфная ассоциация.

Ситуация, когда одно поле внешнего ключа используется для ссылок на разные таблицы [15], это усложняет структуру БД и создает проблемы при запросах и поддержке данных (рис. 14). Главный недостаток – невозможность обеспечения ссылочной целостности данных [20].

товар	категория	Подкатегория 1	Подкатегория 2
сумочка	аксессуары		
брюки	одежда	мужская	летняя
майка	одежда	женская	

Рис. 12. Антипаттерн «Level Hardcoding»
Fig. 12. “Level Hardcoding” antipattern

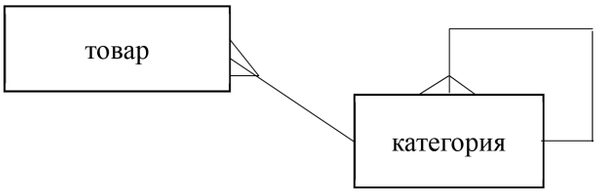

Рис. 13. Решение антипаттерна «Level Hardcoding»
Fig. 13. Solution of the antipattern “Level Hardcoding”

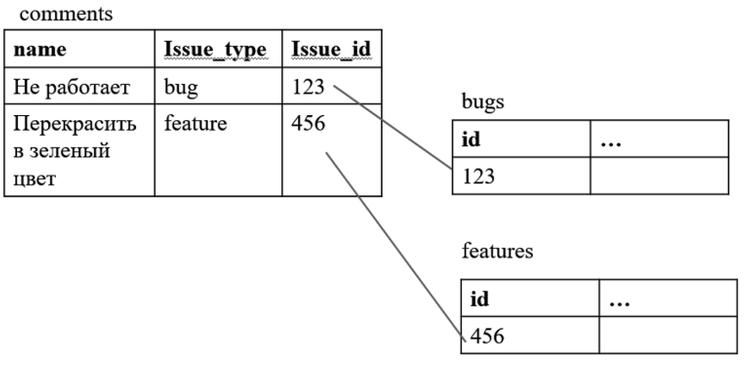

Рис. 14. Антипаттерн «Полиморфная ассоциация» [17]
Fig. 14. “Polymorphic association” antipattern

id	№ счета	Сумма счета	№ платежа	Сумма платежа
1	1	1000	12	600
2	1		1	400
3	2	500	6	500

Рис. 15. Антипаттерн «“Божественная” таблица»
Fig. 15. “ ‘Divine’ table” antipattern

Можно применить следующие способы, чтобы избежать использования полиморфных ассоциаций при создании базы данных:

- создание отдельных полей внешних ключей для каждой ассоциации, чтобы явно указать, к какой таблице относится ссылка (два ключа и LEFT JOIN):

```
SELECT * FROM Comments c
LEFT JOIN Bugs b USING (bug_id)
LEFT JOIN Features f USING (feature_id);
```

- применение паттерна «наследование в базе данных» (Table Inheritance): разделение общих и специфических атрибутов в несколько таблиц.

2.5. «Божественная» таблица (или Монолитная таблица).

Одна большая таблица, в которой сосредоточены данные из разных сфер или смыслов, снижает читаемость и поддерживаемость кода (рис. 15).

Причина применения антипаттерна – в легкости (кажущейся!) проведения, например, подсчета стоимости и задолженности одним запросом. Однако такое преимущество не стоит потери контроля целостности данных, аномалий удаления и сложностей с поддержкой операций модификации.

Избежать избыточности и улучшить структуру базы данных позволит разделение

монолитной таблицы на отдельные таблицы, каждая из которых будет содержать данные только для конкретной сущности (рис. 16).

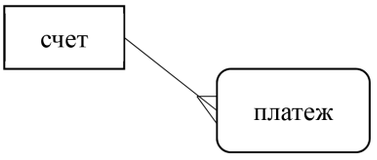

Рис. 16. Решение антипаттерна «“Божественная” таблица»

Fig. 16. Solution of the antipattern “‘Divine’ table”

2.6. Параллельные атрибуты и их группы [16].

Использование различных групп атрибутов для одной и той же сущности указывает на возможное отсутствие отдельной таблицы для этой сущности, что приводит к дублированию кода и снижению гибкости (рис. 17).

client
id
work_adress
work_phone
home_adress
home_phone

Рис. 17. Антипаттерн «Параллельные атрибуты и их группы»

Fig. 17. “Parallel attributes and their groups” antipattern

Аудит структуры данных поможет выявить повторяющиеся группы атрибутов и пропущенные сущности, которые требуют выделения в отдельные таблицы (рис. 18).

2.7. Параллельные связи.

Установление нескольких независимых отношений между одними и теми же таблицами усложняет обработку запросов и поддержку базы данных (рис. 19).

Если мы обратим внимание на приведенный пример, то увидим, что для его реализации потребуется 5 отдельных таблиц – включая 3 промежуточных. А ведь список вариантов связей может быть расширен, например, у статьи может быть еще переводчик, иллюстратор и т.д.

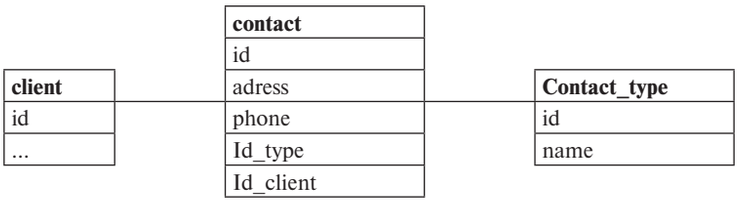

Рис. 18. Решение антипаттерна «Параллельные атрибуты и их группы»

Fig. 18. Solution of the antipattern “Parallel attributes and their groups”

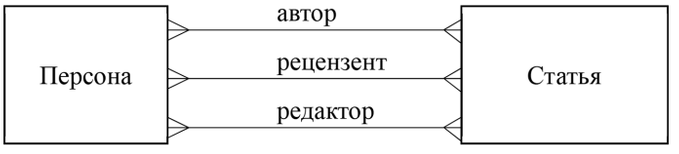

Рис. 19. Антипаттерн «Параллельные связи»

Fig. 19. “Parallel connections” antipattern

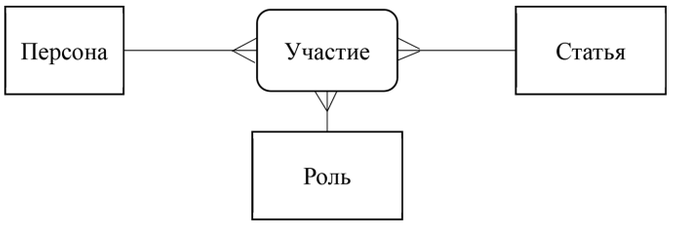

Рис. 20. Решение антипаттерна «Параллельные связи» [14, 20]

Fig. 20. Solution of the antipattern “Parallel connections”

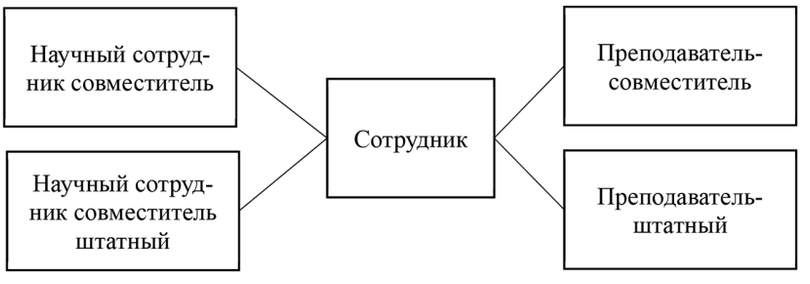

Рис. 21. Антипаттерн «Мозаичное наследование»

Fig. 21. “Mosaic inheritance” antipattern

2.8. Мозаичное наследование [15].

Использование фрагментированных или разрозненных методов наследования среди сущностей затрудняет анализ и поддержку кода. Также использование наследования по двум и более признакам усложняет написание запросов и накладывает дополнительные условия для выборки нужных данных (рис. 21).

Пересмотр структуры данных и возможное выделение дополнительных сущностей или атрибутов поможет упростить связи между данными и сделать структуру более понятной (рис. 20). В качестве решения применяется паттерн, также предложенный в книге М. Блаха [16].

Избежать мозаичного наследования и улучшить структуру данных можно с помощью разделения общих и специфических атрибутов в несколько таблиц, использование внешних ключей для связи между таблицами и явное объявление связей между родительскими и дочерними сущностями (рис. 22).

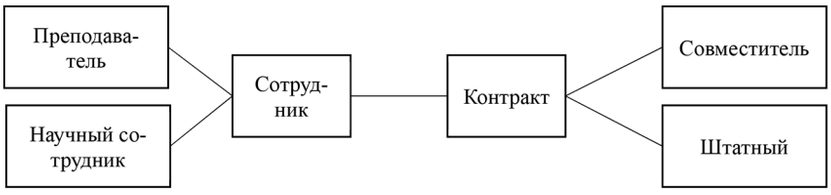

Рис. 22. Решение антипаттерна «Мозаичное наследование»

Fig. 22. Solution of the antipattern “Mosaic inheritance”

3. Антипаттерны, связанные с обходом стандартных возможностей баз данных и оптимизацией.

3.1. Изменение данных в рабочем порядке.

Внесение изменений в данные для реализации сложных запросов с целью хранения промежуточных результатов или пометки обработанных строк. Непредсказуемые изменения данных или состояния объектов приводят к лишней нагрузке, нестабильности и ошибкам в системе, в том числе при выполнении таких запросов параллельно (рис. 23).

клиент	статус	tmp
Иванов И.И.	4	1244
Петров П.П.	1	156
Елизаров Е.Е.	4	356

Рис. 23. Антипаттерн «Изменение данных в рабочем порядке»

Fig. 23. “Changing data in due course” antipattern

Использовать подобное решение – все равно что разобрать стену для того, чтобы наклеить обои.

Здесь необходимо вспомнить очередной принцип: *именно база данных является ядром информационной системы.*

Для сохранения промежуточных результатов и обеспечения целостности данных можно использовать временные таблицы, которые автоматически удаляются после завершения сессии.

3.2. Спекулятивное изменение метаданных.

Изменение метаданных (структуры или поведения) программы во время исполнения. Использование такого антипаттерна может привести к сложностям при отладке и поддержке (рис. 24).

	Добавляем для экзаменационной сессии		
студент	**subj1**	**subj2**	**subj3**
Иванов И.И.	4	5	3
Петров П.П.	2	4	5
Елизаров Е.Е.	3	4	3

Рис. 24. Антипаттерн «Спекулятивное изменение метаданных»

Fig. 24. “Speculative metadata modification” antipattern

Важно спроектировать адекватную структуру базы и изменять метаданные лишь при изменении задач. Любые изменения метаданных должны проходить через процесс тестирования для оценки влияния на существующие данные и приложения.

Отметим здесь еще один принцип проектирования: *никакое изощренное программирование не исправит фатальных недостатков архитектуры.*

3.3 Отказ от SQL в пользу программного обхода.

Использование альтернативных методов доступа к данным вместо SQL часто приводит к увеличению сложности и загрязнению кода (рис. 25).

Отказ от SQL означает, что вы отказываетесь от всех преимуществ, которые предлагают современные системы управления базами данных, таких как транзакционная целостность, репликация данных и механизмы восстановления после сбоев. Вместо этого вам придется реализовывать все эти функции самостоятельно, что грозит увеличением ошибок и уязвимостей.

```
Clients.first;
While not clients.eof
do
        begin
            …
          clients.next;
        end;
```

Рис. 25. Антипаттерн «Отказ от SQL в пользу программного обхода»

Fig. 25. “Abandoning SQL in favor of software bypass” antipattern

Каждая часть системы должна выполнять только ту задачу, за которую она отвечает, и операции над данными должны выполняться только в том слое, где они находятся. Применение ORM-фреймворков позволит представить данные из базы в виде объектов и работать с ними, не покидая среду языка программирования. Использование хранимых процедур для выполнения сложных операций над данными может упростить логику приложения и повысить производительность.

Исключением может служить использование курсоров при сложных запросах. Например, если требуется выполнить итерацию по набору данных с выполнением определенных действий для каждой строки, то использование курсоров в рамках программного кода может оказаться более удобным и эффективным, чем написание сложного и запутанного SQL-запроса.

3.4. Злоупотребление индексами и их некорректное использование.

Неправильное использование индексов, например, создание избыточных или ненужных индексов может снижать производительность базы данных (рис. 26). Так запросы в левой части рисунка не будут задействовать существующие

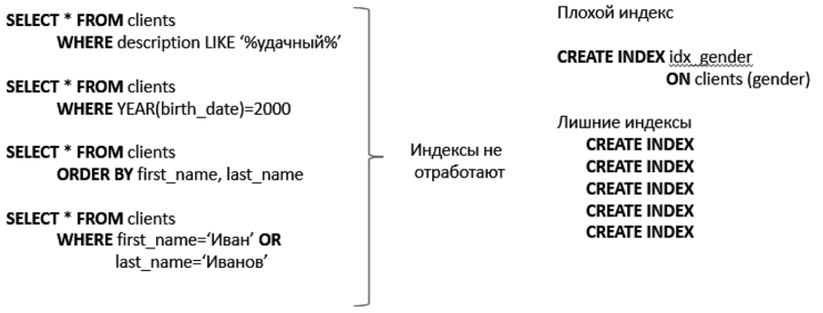

Рис. 26. Антипаттерн «Злоупотребление индексами и их некорректное использование»

Fig. 26. "Abuse of indexes and their incorrect use" antipattern

индексы. Индекс по полю с ограниченным набором значений обладает низкой селективностью и бесполезен. Лишние индексы замедляют операции обновления данных.

В таких случаях необходимо провести анализ производительности базы данных и запросов для определения неиспользуемых или избыточных индексов и для идентификации отсутствующих индексов, которые могут улучшить производительность запросов. Использование специализированных инструментов для мониторинга производительности базы данных может помочь выявить проблемные запросы и индексы.

Результаты

В результате проведенного анализа были рассмотрены разнообразные антипаттерны в проектировании баз данных и их негативное влияние на производительность, безопасность и масштабируемость системы. Был проведен обзор существующих антипаттернов, проанализированы их возможные последствия, предложены методы и стратегии по предотвращению и устранению антипаттернов в базах данных.

Важные направления для дальнейших исследований в данной области:

- более глубокий анализ конкретных случаев применения антипаттернов в реальных проектах;
- разработка инструментов и методик для автоматического выявления и устранения антипаттернов;
- исследование влияния антипаттернов на общую архитектуру информационных систем.

Для того, чтобы избежать подобных ошибок специалистам в области проектирования баз данных необходимо:

- систематически анализировать используемые шаблоны и методы при проектировании баз данных;
- уделять внимание современным требованиям к проектированию и администрированию баз данных;
- развивать навыки в области устранения антипаттернов для обеспечения качества и эффективности баз данных.

Литература

1. Харрингтон Джен Л. Проектирование реляционных баз данных. М.: Лори, 2006. 230 с.

2. Титовская Н.В., Титовский С.Н. Методика обучения будущих IT-специалистов проектированию и разработке баз данных на основе интерактивного подхода // Вестник Красноярского государственного педагогического университета им. В.П. Астафьева. 2019. № 4(50). С. 75–87. DOI: 10.25146/1995-0861-2019-50-4-164.

3. Яхина З.Т., Осипова А.Л., Ризаев И.С. Методология проектирования баз данных в процессе обучения // Образовательные технологии и общество. 2012. № 15(1). С. 525–536.

4. Ahmad R., Chyi W. A., Sarlan A., Kasbon R. Guiding novice database developers in database schema creation // IEEE Conference on e-Learning, e-Management and e-Services. 2015. № (9). С. 708–715. DOI: 10.1109IC3e.2014.7081243.

5. Александер К. Язык шаблонов. Города. Здания. Строительство. Пер. с англ. М.: Студия Артемия Лебедева, 2014. 1096 с.

6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Паттерны объектно-ориентированного проектирования. Пер. с англ. А. Слинкин. СПб.: Питер, 202. 448 с.

7. Аллен Э. Типичные ошибки проектирования. СПб.: Питер, 2003. 224 с.

8. Аручиди Н.А., Калугян К.Х., Щербаков С.М. Типичные ошибки (антипаттерны) учебно-методической деятельности в вузе // Международный научно-исследовательский журнал. 2021. № 2(104). С. 13–18. DOI: 10.23670/IRJ.2021.103.2.033.

9. Калугян К. Х., Щербаков С. М. Типичные ошибки представления результатов выпускных квалификационных работ по направлению «Прикладная информатика» // Информатика и образование. 2016. № 5(274). С. 20–28.

10. William J. Brown, Hays W. «Skip» McCormick, Scott W. Thomas. AntiPatterns in Project Management. Wiley, 2000.

11. Кучерова К.Н. Анализ масштабируемости баз данных и их приложений на этапе проектирования. Системный анализ в проектировании и управлении // Сборник научных трудов XXI Международной научно-практической конференции: в 2-х томах (Санкт-Петербург, 29–30 июня 2017 г.). СПб.: Санкт-Петербургский политехнический университет Петра Великого, 2017. С. 287–292.

12. Ржавин В.В., Обломов И.А., Фадеева К.Н. Использование шаблонов проектирова-

ния реляционных баз данных в практике высшей школы // Современные наукоемкие технологии. 2022. № 10-1. С. 84–88. DOI: 10.17513snt.39351.

13. Davidson L. Ten common database design mistakes. RedGate Hub, 2007.

14. Vitacolonna N. Conceptual Design Patterns for Relational Databases. 17th International Conference on Information and Software Technologies (IT 2011), 2011.

15. Karwin B. SQL Antipatterns: Avoiding the Pitfalls of Database. Pragmatic Bookshelf, 2010. 333 с.

16. Blaha M. Patterns of Data Modeling. CRC Press, 2010. 266 с.

17. Кириллов В., Громов Г. Введение в реляционные базы данных. СПб.: БХВ-Петербург, 2012. 464 с.

18. Kimball R., Ross M. The Data Warehouse Toolkit The Complete Guide to Dimensional Modeling. 2nd Edition, John Wiley & Sons, New York, 2002. 464 с.

19. Брукс Ф. Мифический человеко-месяц, или как создаются программные системы. СПб.: Питер, 2021. 368 с.

20. Blaha M. Referential Integrity Is Important For Databases. Modelsoft Consulting Corp, 2005.

References

1. Kharrington Dzhen L. Proyektirovaniye relyatsionnykh baz dannykh = Designing Relational Databases. Moscow: Laurie; 2006. 230 p. (In Russ.)

2. Titovskaya N.V., Titovskiy S.N. Methodology for teaching future IT specialists database design and development based on an interactive approach. Vestnik Krasnoyarskogo gosudarstvennogo pedagogicheskogo universiteta im. V.P. Astaf'yeva = Bulletin of the Krasnoyarsk State Pedagogical University named after V. P. Astafiev. 2019; 4(50): 75-87. DOI: 10.25146/1995-0861-2019-50-4-164. (In Russ.)

3. Yakhina Z.T., Osipova A.L., Rizayev I.S. Methodology for designing databases in the learning process. Obrazovatel'nyye tekhnologii i obshchestvo = Educational technologies and society. 2012; 15(1): 525-536. (In Russ.)

4. Ahmad R., Chyi W.A., Sarlan A., Kasbon R. Guiding novice database developers in database schema creation. IEEE Conference on e-Learning, e-Management and e-Services. 2015; (9): 708-715. DOI: 10.1109IC3e.2014.7081243.

5. Aleksander K. Yazyk shablonov. Goroda. Zdaniya. Stroitel'stvo. Per. s angl = Pattern Language. Cities. Buildings. Construction. Tr. from Eng. Moscow: Art. Lebedev Studio; 2014. 1096 p. (In Russ.)

6. Gamma E., Khelm R., Dzhonson R., Vlissides Dzh. Patterny ob»yektno-oriyentirovannogo proyektirovaniya. Per. s angl. A. Slinkin = Patterns of Object-Oriented Design. Tr. from Eng. A. Slinkin. Saint Petersburg: Piter; 202. 448 p. (In Russ.)

7. Allen E. Tipichnyye oshibki proyektirovaniya = Typical Design Mistakes. Saint Petersburg: Piter; 2003. 224 p. (In Russ.)

8. Aruchidi N.A., Kalugyan K.Kh., Shcherbakov S.M. Typical mistakes (antipatterns) of educational and methodological activities at the university. Mezhdunarodnyy nauchno-issledovatel'skiy zhurnal = International Research Journal. 2021; 2(104): 13-18. DOI: 10.23670/IRJ.2021.103.2.033. (In Russ.)

9. Kalugyan K.Kh., Shcherbakov S.M. Typical mistakes in presenting the results of final qualification works in the direction of «Applied Informatics». Informatika i obrazovaniye = Informatics and Education. 2016; 5(274): 20-28. (In Russ.)

10. William J. Brown, Hays W. «Skip» McCormick, Scott W. Thomas. AntiPatterns in Project Management. Wiley; 2000.

11. Kucherova K.N Analysis of scalability of databases and their applications at the design stage. Systems analysis in design and management. Sbornik nauchnykh trudov XXI Mezhdunarodnoy nauchno-prakticheskoy konferentsii: v 2-kh tomakh = Collection of scientific papers of the XXI International scientific and practical conference: in 2 volumes (St. Petersburg, June 29-30, 2017). Saint Petersburg: Peter the Great St. Petersburg Polytechnic University; 2017: 287-292. (In Russ.)

12. Rzhavin V.V., Oblomov I.A., Fadeyeva K.N. Using relational database design patterns in higher education practice. Sovremennyye naukoyemkiye tekhnologii = Modern science-intensive technologies. 2022; 10-1: 84–88. DOI: 10.17513snt.39351. (In Russ.)

13. Davidson L. Ten common database design mistakes. RedGate Hub; 2007.

14. Vitacolonna N. Conceptual Design Patterns for Relational Databases. 17th International Conference on Information and Software Technologies (IT 2011); 2011.

15. Karwin B. SQL Antipatterns: Avoiding the Pitfalls of Database. Pragmatic Bookshelf; 2010. 333 p.

16. Blaha M. Patterns of Data Modeling. CRC Press; 2010. 266 p.

17. Kirillov V., Gromov G. Vvedeniye v relyatsionnyye bazy dannykh = Introduction to relational databases. Saint Petersburg: BHV-Petersburg; 2012. 464 p. (In Russ.)

18. Kimball R., Ross M. The Data Warehouse Toolkit The Complete Guide to Dimensional Modeling. 2nd Edition, John Wiley & Sons, New York; 2002. 464 p.

19. Bruks F. Mificheskiy cheloveko-mesyats, ili kak sozdayutsya programmnyye sistemy = The Mythical Man-Month, or How Software Systems Are Created. Saint Petersburg: Piter; 2021. 368 p. (In Russ.)

20. Blaha M. Referential Integrity Is Important For Databases. Modelsoft Consulting Corp; 2005.

Сведения об авторах

Сергей Михайлович Щербаков
Ростовский государственный экономический университет (РИНХ), Ростов-на-Дону, Россия
Эл. почта: sergwood@mail.ru

Каринэ Хачересовна Калугян
Ростовский государственный экономический университет (РИНХ), Ростов-на-Дону, Россия
Эл. почта: kalugyan@yandex.ru

Анна Вадимовна Абрамова
Ростовский государственный экономический университет (РИНХ), Ростов-на-Дону, Россия
Эл. почта: ag-anna-92@yandex.ru

Вера Юрьевна Гречкина
Ростовский государственный экономический университет (РИНХ), Ростов-на-Дону, Россия
Эл. почта: wwwera@list.ru

Information about the authors

Sergey M. Shcherbakov
Rostov State University of Economics (RINH), Rostov-on-Don, Russia
E-mail: sergwood@mail.ru

Karine K. Kalugyan
Rostov State University of Economics (RINH), Rostov-on-Don, Russia
E-mail: kalugyan@yandex.ru

Anna V. Abramova
Rostov State University of Economics (RINH), Rostov-on-Don, Russia
E-mail: ag-anna-92@yandex.ru

Vera Y. Grechkina
Rostov State University of Economics (RINH), Rostov-on-Don, Russia
E-mail: wwwera@list.ru