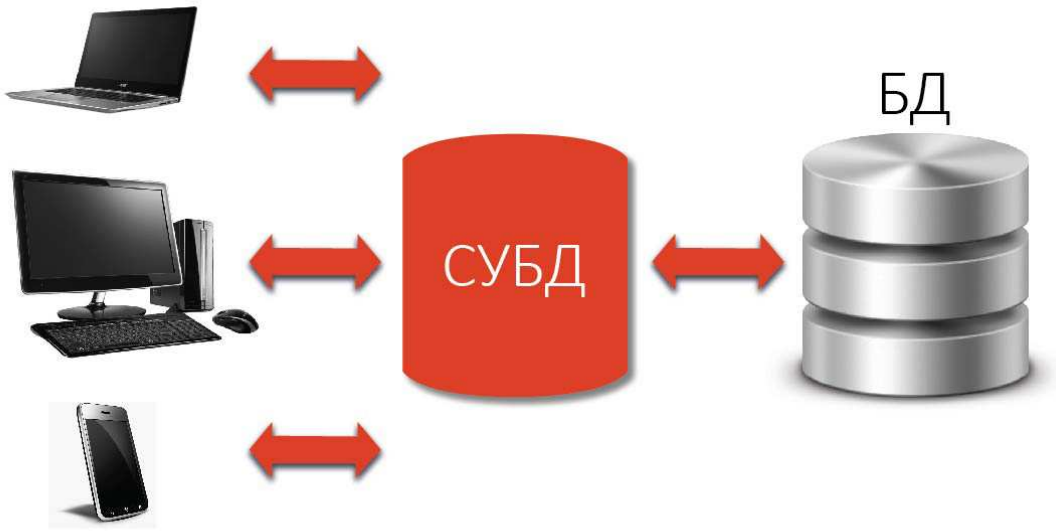
А. А. Рыбанов

ОЦЕНКА КАЧЕСТВА БАЗ ДАННЫХ



Волжский
2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ВОЛЖСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

А. А. Рыбанов

ОЦЕНКА КАЧЕСТВА БАЗ ДАННЫХ

Электронное учебное пособие

2024

УДК 004.6(07)
ББК 32я73
Р 931

Рецензенты:
Волгоградский государственный социально-педагогический университет, заведующий кафедрой методики преподавания математики и физики, ИКТ, профессор, д.п.н.
Смыковская Т.А.;
Директор ООО Научно-производственный центр «АИР»
Шуревский А.Н.

Издается по решению редакционно-издательского совета
Волгоградского технического университета

Рыбанов, А. А.
Оценка качества баз данных [Электронный ресурс] : учебное пособие / А. А. Рыбанов ; Министерство науки и высшего образования Российской Федерации, ВПИ (филиал) ФГБОУ ВО ВолгГТУ. – Электрон. текстовые дан. (1 файл: 2,12 МБ). – Волжский, 2024. – Режим доступа: http://lib.volpi.ru. – Загл. с титул. экрана.
ISBN 879-5-9948-4850-0

В учебном пособии рассматриваются показатели и методы оценивания качества реляционных баз данных. Приводятся показатели разнообразия типов данных физических схемах данных, приведены примеры их практического применения. Рассмотрена процедура оценки сложности физических схем реляционных баз данных. Представлены математические модели для оценки полноты и надежности баз данных. В приложениях приводятся рекомендации об именовании объектов баз данных и об оформлении исходных кодов программ. Учебное пособие соответствует актуальным требованиям федерального государственного образовательного стандарта высшего образования. Учебное пособие адресовано студентам высших учебных заведений, обучающимся по направлениям 09.03.01 «Информатика и вычислительная техника» и 09.03.04 «Программная инженерия».

Ил. 7, табл. 13, библиограф.: 8 назв.

ISBN 879-5-9948-4850-0

ВВЕДЕНИЕ

Современные базы данных – один из тех объектов в сфере информатизации, от которых иногда требуется особенно высокое качество и наличие возможности его оценки. Но что означает качество баз данных, какие требования следует предъявлять к их качеству, какими характеристиками можно описывать качество, как их оценивать и измерять? Для этого могут быть полезны методы и стандарты, разработанные для анализа сложных программных средств.

При комплексном анализе качества баз данных не всегда удается четко разделить требования и значения характеристик качества для каждого из этих объектов [1, 2]. Одна СУБД может обрабатывать различные по структуре, составу и содержанию данные, а одни и те же данные могут управляться различными СУБД. При анализе качества баз данных целесообразно рассматривать два компонента: систему программ управления данными и совокупность данных, упорядоченных по некоторым правилам. Хотя эти компоненты тесно взаимодействуют при реализации конкретной базы данных, первоначально они создаются независимо и могут рассматриваться в своем жизненном цикле как два объекта, которые различаются [2]:

- номенклатурой и содержанием показателей качества, определяющих их назначение, функции и потребительские свойства;

- технологией и средствами автоматизации разработки и обеспечения всего жизненного цикла объекта;

- категориями специалистов, обеспечивающих создание, эксплуатацию или применение баз данных;

- комплектами эксплуатационной и технологической документации, поддерживающими жизненный цикл объекта.

Практически весь набор характеристик и атрибутов из стандарта ISO 9126 «Качество программных средств» в той или иной степени может ис-

пользоваться в составе требований к СУБД. Особенности состоят в изменении акцентов при их выборе и упорядочении. Во всех случаях важнейшими характеристиками качества СУБД являются требования к функциональной пригодности процессов формирования и изменения информационного наполнения баз данных администраторами, а также доступа к данным и представления результатов пользователям.

Различия требований к характеристикам качества привели к созданию широкого спектра локальных, специализированных и распределенных СУБД. В зависимости от области применения, приоритет при оценке качества может отдаваться различным конструктивным характеристикам: надежности и защищенности применения (финансовая сфера), удобству использования малоквалифицированными пользователями (социальная сфера), эффективности использования ресурсов (сфера материально-технического снабжения). Однако практически во всех случаях сохраняется некоторая роль других конструктивных показателей качества – для каждого из них необходимо оценивать его приоритет для конкретной сферы применения, меры и шкалы необходимых и допустимых характеристик качества.

В системах баз данных доминирующее значение приобретают сами данные, их хранение и обработка. Для оценивания качества информации может применяться общий методический подход к выделению адекватной номенклатуры стандартизированных в ISO 9126 базовых характеристик и субхарактеристик. Однако их содержание для применения к качеству баз данных требуется уточнить. Выделяемые показатели качества должны иметь практический интерес для пользователей и быть упорядочены в соответствии с приоритетами практического применения. Кроме того, каждый выделяемый показатель качества должен быть пригоден для достоверного экспертного оценивания или измерения, а также для сравнения с требуемым значением.

При разработке базы данных в техническом задании и спецификации на нее должен формализоваться представительный набор функциональных требований к качеству базы данных, адекватный ее назначению и области применения, а также требованиям заказчика и потенциальных пользователей. Так же как для программных систем, характеристики качества информации можно разделить на функциональные и конструктивные. Их номенклатура, содержание и субхарактеристики базируются на ISO 9126. Может быть заложена основа для стандартизированного формирования требований к качеству баз данных и при изложении содержания характеристик качества использованы номенклатура и описания характеристик, в которых объекты для анализа качества – «программы», заменены термином и содержанием – «информация баз данных». Однако номенклатура показателей качества не всегда может ограничиваться только характеристиками информации в базе данных, а должна включать ряд уточнений, отражающих комплексную эффективность и функциональную пригодность ее применения в реальных условиях.

Формализация характеристик качества информации баз данных, на основе стандартов, разработанных для оценивания программных средств, открывает путь для применения апробированных на комплексах программ методов систематизации, определения и повышения их качества. Использование стандартизированных характеристик качества информации баз данных позволяет упорядочить выбор требований к ним и оценивание достигнутого качества. Это должно способствовать повышению качества баз данных в целом, с учетом их программных и информационных компонентов, возможности достоверного определения их реальных характеристик при разработке, испытаниях и сертификации.

1. ПОКАЗАТЕЛИ И МЕТОДЫ ОЦЕНИВАНИЯ КОРРЕКТНОСТИ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Качество информационных систем существенно зависит от качества баз данных, которые в настоящее время играют роль ядра информационных систем. Оценивание качества программного обеспечения (ПО) информационных систем (ИС) неразрывно связано с процессом верификации, т.е. с подтверждением того, что функциональные возможности ПО соответствуют их описаниям в программной документации. Функциональные возможности, которые не имеют описания в документации или не соответствуют описанию, называются недекларированными возможностями (НДВ) [1]. Они могут являться как следствием ошибок разработчика, так и результатом выполнения умышленно внедрённого в программу кода. Как случайные, так и преднамеренные НДВ (последние называются также программными закладками) представляют угрозу информационной безопасности программных систем. Цель верификации программ – выявление и регистрация НДВ для последующего их устранения. Любое программное средство обладает определённой корректностью. Понятие корректность является более узким, чем понятие качество, так как последнее включает в себя такие характеристики, как надёжность, мобильность, понятность, сопровождаемость программной системы. Корректность характеризует функциональные возможности программы, т.е. одну из составляющих качества. Можно считать программу абсолютно корректной, если в ней отсутствуют НДВ. Однако на практике не удаётся избежать появления случайных НДВ, поэтому абсолютно корректных программ не встречается. Кроме того, говоря о корректности программного продукта, всегда подразумевают его соответствие некоторому эталонному объекту или совокупности формализованных эталонных характеристик и правил. Корректность программы наиболее полно определяется степенью её соответствия программной спецификации. При отсутствии полностью формализованной

спецификации в качестве эталона иногда используются неформализованные представления разработчика, пользователя или заказчика программ. В результате эталон, которому должна соответствовать программа, часто оказывается неопределенным, неточным и понятие корректности становится субъективным. С другой стороны, сами по себе формальные спецификации могут оказаться некорректными с точки зрения заказчика, поэтому для любой программы, корректной относительно спецификаций, существует риск обнаружения его некорректности относительно требований заказчика или пользователя. Из сказанного следует, что связь между процессом верификации и обеспечением корректности ПО такова: верификация способствует повышению корректности, но не стопроцентному её обеспечению. Оценивание качества ИС неразрывно связано с верификацией всех её компонентов. В настоящее время существует множество методов и средств автоматизированной верификации ПО, часто применяемых совместно, взаимно дополняющих друг друга. Но не менее важным, чем ПО, компонентом ИС является база данных (БД). Она также оказывает влияние на качество ИС и нуждается в верификации. Для проведения полноценной верификации требуется:

- правильное понимание сущности базы данных;
- система количественных и качественных показателей корректности базы данных;
- наличие автоматизированных систем верификации базы данных.

1.1. База данных как программный компонент информационной системы

Важным компонентом ИС является база данных (БД). На современном этапе БД должны рассматриваться не только как хранилища данных, но и как полноценные программные компоненты. Обоснованием данного положения служит практика использования промышленных СУБД и ИС,

основанная в значительной степени на использовании клиентсерверной архитектуры и технологии активного сервера. Суть последней заключается в том, что функции ИС, выполняющие обработку данных в БД, реализуются не в клиентских приложениях, а на стороне сервера СУБД в виде хранимых подпрограмм БД. Слово «хранимый» указывает на то, что коды этих подпрограмм располагаются вместе с данными и являются, таким образом, объектами БД.

Известно три типа хранимых подпрограмм: функции, процедуры и триггеры. Функции и процедуры запускаются на выполнение путём явного вызова. Клиентскому приложению, чтобы вызвать процедуру, необходимо предварительно соединиться с БД, где расположен её откомпилированный код. Обращение к процедуре производится по имени с передачей входных данных и, возможно, с заданием выходных параметров.

Функция отличается от процедуры тем, что она всегда возвращает атомарное значение определённого типа и вызывается путём использования в арифметических и логических выражениях на месте операнда.

Наконец, триггер – это специальный тип хранимых подпрограмм. Он выполняется автоматически в ответ на вставку, обновление или удаление записей в таблице и служит для поддержания корректности и согласованности (целостности) данных. Реализация хранимых подпрограмм возможна как на языке баз данных, так и на языках программирования высокого уровня C, C++, Pascal, Java и т.п. [1].

Из сказанного следует, что БД, поддерживаемые промышленными СУБД (такими как Oracle, Microsoft SQL Server, DB2), имеют двойственную природу. Оставаясь хранилищами данных, они одновременно играют роль программного обеспечения в ИС или, точнее, становятся полноценными программными компонентами, напоминающими динамически подключаемые библиотеки операционной системы Windows или COM-

объекты. Налицо сращивание данных с методами их обработки в одной сущности – базе данных.

Подобное сращивание в терминах объектно-ориентированного и компонентно-ориентированного подходов получило название инкапсуляции. Есть все основания утверждать, что БД, будучи программным компонентом, инкапсулирует данные и методы их обработки. При этом именно БД в составе ИС является активным сервером, работающим в среде СУБД, подобно тому, как обычный EXE или DLL-файл работает в среде операционной системы.

Как и любая другая часть ИС, БД нуждаются в верификации и тестировании. Учитывая то, что верификация есть подтверждение корректности, необходимо заметить, что корректность БД должна оцениваться при помощи системы математических показателей.

К числу наиболее часто встречающихся сегодня на практике функциональных показателей корректности БД относятся следующие [3]: ·

- *полнота накопленных описаний объектов* – относительное число объектов и документов, имеющихся в БД, к общему числу объектов в аналогичной БД;

- *достоверность* – степень соответствия записей БД реальным объектам в данный момент времени;

- *идентичность данных* – относительное число описаний объектов, не содержащих ошибки, к общему числу документов об объектах в БД;

- *актуальность данных* – относительное число устаревших данных к общему числу накопленных и обрабатываемых записей.

Помимо функциональных показателей, существуют конструктивные показатели корректности, отличающиеся большей универсальностью, независимостью от сферы применения базы данных:

- *объём БД* – число описаний объектов или документов в БД, доступных для хранения и обработки;

- *оперативность* – степень соответствия динамики изменения данных в процессе сбора и обработки состояниям реальных объектов или величина запаздывания между появлением (изменением) характеристик реального объекта и его отражением в БД;

- *периодичность* – промежуток времени между поставками двух последовательных, достаточно различающихся информацией версий БД;

- *глубина ретроспективы* – интервал времени от даты выпуска (записи в БД) самого раннего документа до настоящего времени;

- *динамичность* – относительное число изменяемых описаний объектов к общему числу записей в БД за некоторый интервал времени, определяемый периодичностью издания версий БД.

Нетрудно видеть, что все перечисленные показатели корректности затрагивают содержимое таблиц БД, но не отражают её логическую структуру. С их помощью нельзя оценить качество схемы БД, нельзя установить, нормализованы ли таблицы, обеспечена ли целостность информации на логическом и внешнем уровнях абстрагирования. Наконец, согласно изложенному выше, БД на современном этапе развития ИС является программным компонентом. Поэтому к БД применимы многие характеристики ПО, на основе которых определяется корректность поведения программных систем и, в частности, проверяется отсутствие НДВ.

1.2. Виды и показатели корректности баз данных

Оценивание корректности БД предполагает анализ её содержимого, логической структуры и программной составляющей (хранимых подпрограмм). В связи с этим необходимо различать три вида корректности БД: корректность содержимого, корректность схемы данных и программная корректность. Рассмотрим каждый из перечисленных видов по отдельности.

1.2.1. Корректность содержимого

Любая запись в БД должна адекватно отражать характеристики реального существующего объекта, экземпляра сущности предметной области; если же в записи фиксируется информация о взаимодействии объектов, то речь должна идти о взаимодействии, имеющем место в реальной действительности. Очевидно, что корректность содержимого определяется функциональными и конструктивными показателями, рассмотренными ранее.

1.2.2. Корректность схемы данных

Традиционно под реляционной схемой понимается набор взаимосвязанных схем отношений или, с точки зрения обычного пользователя, весь набор таблиц БД. В данной работе предлагается более широкое понимание этого термина, а именно: совокупность логически связанных друг с другом таблиц, представлений, индексов, ограничений целостности и умолчаний. Можно выделить следующие подвиды корректности схемы данных.

1. *Точность отображения концептуальной схемы (ER-схемы) на реляционную*. Концептуальная модель реляционной БД обычно представляется в виде диаграмм сущность–связь, или ER-диаграмм, на которых показываются объекты предметной области и связи (способы взаимодействия) между ними. При концептуальном проектировании не оперируют понятиями «таблица», «внешний ключ» и т.п. Но ER диаграммы легко преобразуются в схему реляционной БД по набору типовых правил. В итоге сущностям, представленным на ER-схеме, соответствуют таблицы реляционной БД, а связям – вспомогательные таблицы и внешние ключи. Типовые правила отображения дают возможность «обратного проектирования» – получения ER-схемы по имеющейся реляционной схеме БД. Однако не всегда «обратное проектирование» даёт ER-схему, эквивалентную исходной концептуальной модели. Причиной этого в ряде случаев является не-

точность прямого преобразования, в результате чего схема БД получается некорректной по отношению к исходной ER-модели.

Точность отображения, очевидно, может быть определена как соответствие ER-схемы, полученной в результате «обратного проектирования», исходной концептуальной схеме. Точность отображения характеризуется множеством показателей, учитывающих детализацию сравнения на уровне атрибутов, сущностей, связей и участков ER-диаграмм, образованных двумя отдельно взятыми сущностями и одной ассоциацией между ними.

В каждом случае необходимо рассчитывать относительное число правильно отображённых на реляционную схему сущностей, атрибутов, связей и участков, к общему числу соответствующих элементов, определённых на этапе концептуального проектирования. С другой стороны, в процессе создания реляционной схемы могут появиться недекларированные атрибуты, сущности и связи. Процесс верификации БД должен включать в себя оценивание их процентного содержания в общем числе отражённых на реляционной схеме сущностей, атрибутов и связей соответственно. Чем ниже процентное содержание недекларированных элементов схемы данных, тем корректнее БД по отношению к концептуальной модели.

2. *Нормализованность таблиц*. Основная цель проектирования БД – группирование атрибутов по таблицам так, чтобы минимизировать избыточность данных и по возможности сократить объём памяти, необходимый для физического хранения таблиц. Теория нормальных форм и нормализации описывает один из формальных методов проектирования БД, в котором идентификация таблиц основана на выявлении первичных ключей и функциональных зависимостей между атрибутами [4].

Высшей нормальной формой на практике оказывается, как правило, нормальная форма Бойса–Кодда. Согласно её требованиям, все неключевые атрибуты таблицы должны полностью функционально зависеть от по-

тенциального ключа, а функциональных зависимостей между различными неключевыми атрибутами быть не должно. Нарушение этих требований приводит к избыточности данных и, как следствие, к аномалиям вставки, обновления и удаления записей.

Современные методы проектирования (в том числе упомянутый ранее метод ER-моделирования) позволяют автоматически приводить таблицы БД к нормальной форме Бойса–Кодда. Однако ошибки могут возникать, особенно если разработчиками принимается нешаблонное решение о частичной денормализации таблиц.

Проще всего оценивать нормализованность БД при помощи относительного числа полностью нормализованных таблиц к общему числу таблиц. Более сложной задачей является оценивание степени допустимости денормализации для данного проекта, если таковая имеет место.

3. *Логическая целостность*. Важной составляющей корректности схемы данных является логическая целостность. С каждой предметной областью связаны определённые требования целостности, которые тем или иным образом ограничивают диапазоны возможных значений атрибутов сущностей и связей, говорят о допустимости либо недопустимости комбинаций некоторых значений. Реализация требований целостности в БД позволяет избавиться от появления отрицательных возрастов и масс, от повторения одного и того же ИНН у различных лиц, от некорректной последовательности дат начала и завершения работы (в случае, когда они в результате ошибки пользователя меняются местами) – словом, избежать появления логически противоречивой, несогласованной, «невозможной» с позиции здравого смысла информации.

Простые требования целостности реализуются при помощи специальных объектов БД, называемых ограничениями. Большинством реляционных СУБД сегодня поддерживаются такие ограничения целостности, как первичный ключ, внешний ключ, уникальность значений, запрет неопре-

делённых значений и ограничение на основе логического условия. Более сложные требования целостности реализуются процедурно с помощью специальных подпрограмм – триггеров. Следует заметить, что целостность в данном случае пересекается с программной корректностью, которая будет рассматриваться далее. И в первом, и во втором случае речь идёт об автоматическом поддержании целостности. Во время выполнения операций вставки, удаления и обновления неверные данные либо не принимаются совсем, либо автоматически корректируются.

Целесообразно оценивать логическую целостность БД при помощи следующих показателей.

3.1. *Полнота реализации требований целостности* – отношение реализованных в БД ограничений и триггеров к общему числу заявленных требований. Ограничения, которые не были заявлены в документации, не рассматриваются.

3.2. *Относительное число не заявленных в документации ограничений целостности к общему числу реализованных*. Если недекларированных ограничений нет, показатель принимает нулевое значение, что говорит о корректности схемы данных.

3.3. *Согласованность ограничений с содержимым таблиц*. Возможны ситуации, когда дополнительное ограничение целостности или новый триггер добавляются в уже заполненную БД. При этом нельзя допустить, чтобы новое ограничение вошло в противоречие с уже добавленными в таблицы записями. Для обычных ограничений целостности эта проблема решена на уровне СУБД. Система запрещает вводить ограничение, если в БД уже имеются строки таблиц, которые ему не удовлетворяют. Но СУБД не в состоянии выполнить подобную проверку для триггера. Поэтому возможны противоречия: с одной стороны, имеется сложное ограничение целостности, реализуемое триггером; с другой стороны, записи, появившиеся

в БД до создания триггера, могут заведомо не удовлетворять этому ограничению.

Показатель согласованности ограничений и содержимого таблиц рассчитывается как отношение записей, удовлетворяющих всем ограничениям, к общему числу записей. Расчёт этого показателя неразрывно связан с анализом программного кода триггеров, цель которого – выяснить декларативный смысл каждого триггера. Задача эта сама по себе нетривиальна и требует отдельного рассмотрения.

3.4. *Непротиворечивость ограничений*. В результате ошибок проектирования в БД могут возникать конфликты между ограничениями и триггерами. Когда разные триггеры и ограничения предъявляют к данным противоречивые требования, вплоть до взаимоисключающих, таблицы БД могут оказаться недоступными для вставки и обновления строк.

Свойство непротиворечивости ограничений затруднительно оценить количественно. Вероятно, речь должна идти о качественном показателе или комплексе количественных и качественных показателей, позволяющих оценить степень изолированности самих ограничений друг от друга, степень доступности данных, с которыми связано множество ограничений и триггеров, а также долю противоречивых ограничений во всей БД.

3.5. *Сложность реализации требования целостности*. Даже простые ограничения могут быть реализованы с помощью триггеров. Теоретически это допустимо, но на практике подобные решения снижают эффективность системы [5]. Кроме того, очевидно, что они затрудняют последующую верификацию БД, увеличивают время анализа, так как приходится восстанавливать декларативный смысл каждого триггера. Декларативный смысл обычного ограничения целостности всегда ясен из его описания.

Сложность реализации требования целостности – качественный показатель, оцениваемый для каждого триггера. Если данное требование может быть полностью реализовано при помощи стандартных ограничений,

сложность является очень высокой. В противном случае вычисление показателя производится на основе семантического анализа команд, выполняемых в теле триггера. Возможные значения показателя сложности реализации: «очень низкая», «низкая», «средняя», «высокая», «очень высокая».

1.2.3. Программная корректность баз данных

Корректность программных процедур существенно зависит от их топологической и информационной сложности: чем выше сложность, тем больше вероятность появления неумышленных НДВ, с одной стороны, и тем сильнее усложняется обнаружение любых НДВ, с другой стороны.

К настоящему времени предложено множество метрик оценивания сложности программ [6]. Оценивание сложности хранимых подпрограмм БД может производиться с использованием метрик размера программ и метрик сложности потока управления.

Наиболее распространённой на практике метрикой оценивания размера программ является метрика Холстеда, включающая следующие характеристики:

- словарь операторов n_1 – число уникальных операторов программы;
- словарь операндов n_2 – число уникальных операндов программы;
- общее число операторов программы N_1;
- общее число операндов программы N_2.

Характеристики n_1 и n_2 в сумме составляют словарь программы n, а N_1 и N_2 – длину программы N. На основе перечисленных базовых характеристик рассчитывается объём программы, измеряемый в логических единицах информации (символах, операторах, операндах):

$$V = n \log_2 n .$$

Дополнительно в модели Холстеда вводится характеристика n^* – теоретический словарь программы, т.е. словарь, необходимый для написа-

ния программы с учётом того, что программа сводится к вызову ранее реализованной функции. Теоретический словарь используется при расчёте потенциального объёма программы:

$$V = n^* \log_2 n^* .$$

Метрики сложности потока управления базируются на представлении программы в виде ориентированного графа *G = (V, E)*, в котором *V* – множество вершин, соответствующих операторам, *E* – множество дуг, отражающих переходы между операторами. Для оценивания хранимых подпрограмм БД достаточно применения метрики Маккейба (цикломатического числа Маккейба), характеризующей трудоёмкость тестирования. Цикломатическое число рассчитывается по формуле:

$$Z(G) = e - v + 2p .$$

где *e* – количество дуг графа *G*; *v* – количество вершин; *p* – число компонентов связности графа.

Дополнительно можно использовать метрику Джилба. В ней логическая сложность программы оценивается содержанием в исходном коде операторов ветвления. В рамках метрики Джилба определены следующие характеристики:

- *абсолютная сложность* – количество операторов условия;

- *относительная сложность* – отношение числа операторов условия к общему числу операторов программы.

Названные метрики удобно применять при оценивании сложности каждой хранимой подпрограммы в отдельности. Для оценивания сложности программной составляющей БД в целом предлагается использовать показатели сложности объектно-ориентированных программ [7]. Изначально они создавались как характеристики методов программного класса. Так как базам данных присуще свойство инкапсуляции, хранимые подпрограммы можно рассматривать как аналогию методов программного класса

в объектно-ориентированной программе, а элементы данных БД (атрибуты таблиц) выступают при этом аналогами полей класса.

Исходя из сказанного, можно определить следующие характеристики программной сложности БД.

1. *Суммарная сложность подпрограмм БД*. Вычисляется как алгебраическая сумма сложностей (весов) всех подпрограмм. С тем, какой именно показатель (из перечисленных выше) будет характеризовать сложность отдельно взятой подпрограммы, необходимо определиться заранее, до выполнения вычислений.

2. *Группа показателей, отражающая наличие сцепления между подпрограммами*. Сцепление подпрограмм имеет место в том случае, если они используют общие атрибуты таблиц. Малое сцепление в программном классе или компоненте желательно, поскольку оно увеличивает инкапсуляцию и снижает вероятность возникновения ошибок в поведении компонента. Наиболее простым показателем сцепления является разность между числом пар несцепленных и числом пар сцепленных подпрограмм. Отрицательный показатель сцепления всегда приравнивается к нулю.

Верификация баз данных имеет большое практическое значение, поскольку во многих случаях именно база данных берёт на себя функции ядра ИС, совмещая в себе данные и программные процедуры их обработки. Качество ИС напрямую связано с качеством БД, поэтому разработка и развитие автоматизированных систем комплексного автоматизированного оценивания корректности БД повлечёт за собой повышение качества ИС. Предложенная система показателей корректности БД не является окончательной. Одно из возможных направлений её доработки – добавление характеристик качества постреляционных (объектно-реляционных, объектно-ориентированных и других) БД, а также распределённых БД. Помимо универсальных показателей и метрик, могут использоваться показатели, учитывающие специфику конкретной СУБД.

2. ОЦЕНКА ПОКАЗАТЕЛЕЙ РАЗНООБРАЗИЯ ТИПОВ ДАННЫХ В ФИЗИЧЕСКИХ СХЕМАХ БАЗ ДАННЫХ

Основное влияние на эксплуатационные характеристики базы данных (БД) оказывает физическая организация данных. Для того чтобы база данных функционировала правильно и была максимально оптимизирована, важно тщательно выбирать типы данных для каждого поля в таблицах [1]. Разработчики баз данных должны знать и понимать, как использовать каждый из них правильно. Определяемые пользователями типы данных позволяют существенно усилить контроль над данными и повысить их целостность [2]. Правильно выбранные типы полей позволяют уменьшить физический размер строк таблицы и, соответственно, базу данных, делая чтение и запись быстрее и эффективнее.

2.1. Рекомендации по корректировке типов полей

Рекомендации по корректировке полей рассмотрим на примере СУБД MySQL. MySQL поддерживает большое разнообразие типов данных, и выбор правильного типа для хранения данных имеет решающее значение для достижения хорошей производительности. Для некоторых полей таблицы БД типы данных могут быть изменены без существенного влияния на существующую схему базы данных. Рассмотрим следующие примеры.

Возраст человека. Для хранения возраста человека рекомендуется использовать поле с типом TINYINT UNSIGNED, который может хранить максимальное значение 256 . Однако, не задумываясь, кто-то может использовать тип INTEGER UNSIGNED. Но при использовании типа данных TINYINT для хранения возраста требуется 1 байт, в замен 4 байтов при использовании типа данных INTEGER.

Типы DATE и TIME. Типы полей DATE и TIME занимают по 3 байта, в то время как поле типа DATETIME – 8 байт, но хранение всей информа-

ции в одном поле более предпочтительно для оптимизации обработки данных.

Замена BIGINT на INT. Уменьшить размер первичного ключа с 8 до 4 байтов можно путем изменения его типа данных BIGINT UNSIGNED AUTO_INCREMENT на тип данных INT UNSIGNED AUTO_INCREMENT. Данное преобразование возможно не только для первичного ключа, но и для всех внешних ключей, которые определены как BIGINT. Данный поход может значительно уменьшить пространство, необходимое для индексов в сильно нормализованной базе данных.

IP-адреса. Полю базы данных, предназначенному для хранения IPv4 адресов, как правило, присваивают тип VARCHAR(15), размер которого в среднем составляет 12 байт. В целях уменьшения исходного размера столбца, рекомендуется для поля, содержащего IPv4-адрес, определять тип данных INT UNSIGNED, требующий 4 байта. Преобразования IP-адреса возможно с помощью функций INET_ATON() и INET_NTOA().

Пример:

```
SET @ip = '255.139.67.15';
SELECT @ip, INET_ATON(@ip) AS str_to_i,
  INET_NTOA (INET_ATON (@ip)) AS i_to_str;
```

MD5 значения. Распространенной практикой является хранение значения MD5 в поле, имеющим тип CHAR (32). Для более эффективного хранения шестнадцатеричного значения MD5 можно использовать функции UNHEX() и HEX(), сохраняя данные в типе данных BINARY(16). Выполнение этого преобразования позволяет уменьшить размер поля в два раза.

Пример:

```
SET @str = 'wwwvolpiru';
SELECT MD5(@str),  LENGTH(MD5(@str)) AS len_md5,
  LENGTH(UNHEX (MD5 (@str))) AS len_unhex;
```

Функция PROCEDURE ANALYSE() позволяет получить рекомендации сервера баз данных MySQL по корректировке типов полей для таблицы, заполненной реальными данными, наличие которых играет существенную роль при принятии решений. Предлагаемые функцией PROCEDURE ANALYSE() оптимальные типы данных для каждого поля ориентированы на решение проблемы уменьшения размера таблицы и, как следствие, всей базы данных.

Синтаксис использования функции:

SELECT * FROM <table_name>

PROCEDURE ANALYSE([<max_elements>, [<max_memory]]);

где <table_name> – имя анализируемой таблицы базы данных; <max_elements> – максимальное количество различных значений, содержащихся в поле, которое функция PROCEDURE ANALYSE использует для проверки того, является ли оптимальным типом для поля тип ENUM (по умолчанию 256); <max_memory> – максимальное количество памяти, которое может занимать тип поля ENUM (по умолчанию 8192 байт).

Для исключения в результатах работы функции PROCEDURE ANALYSE рекомендаций типа ENUM для полей, которые содержат более 16 уникальных значений или занимают более 256 байтов нужно задать следующие значения аргументов:

SELECT * FROM <table_name> PROCEDURE ANALYSE(16, 256);

Необходимо учитывать, что выдаваемые функцией PROCEDURE ANALYSE результаты – это всего лишь рекомендации, на достоверность которых существенно влияет количество строк в таблице. И если таблица базы данных в дальнейшем будет пополняться новыми данными, то эти рекомендации могут оказаться неверными, поэтому принятие решения об их применении остается за администратором базы данных.

Общие положения по применению типов полей можно резюмировать в виде следующих рекомендаций:

- рекомендуется объявлять поля как NOT NULL;

- рекомендуется использовать типы полей, которые занимают меньше памяти;

- для цифровых данных не рекомендуется использовать символьные поля;

- рекомендуется использовать поля фиксированной длины, т.к. они обрабатываются быстрее;

- по возможности, рекомендуется использовать типы ENUM и SET;

- рекомендуется для хранения дат и времени использовать специализированные типы;

- рекомендуется использовать функцию PROCEDURE ANALYSE для проверки характеристик полей таблиц, заполненных реальными данными;

- рекомендуется оптимизировать таблицы, подверженные частой модификации.

2.2. Метрики разнообразия типов данных

Физическая схема реляционной базы данных содержит все детали, необходимые конкретной СУБД для создания базы: наименования таблиц и столбцов, типы полей, определения первичных и внешних ключей, индексов.

Контроль качества физической схемы БД невозможен без числовых показателей. При отсутствии количественных измерений трудно принимать какие-либо проектные решения. Существующие модели количественной оценки сложности физических схем реляционных баз данных [3, 5, 6] не учитывают разнообразие используемых в них типов данных, поэтому задача формирования системы количественных критериев для оценки разнообразия типов данных, используемых в физической схеме БД, является актуальной.

Рассмотрим применение показателей экологического разнообразия для решения задачи оценки разнообразия типов данных в физической схеме базы данных для СУБД MySQL.

Разнообразие – это понятие, которое имеет отношение к размаху изменчивости или различий между некоторыми множествами или группами объектов. При оценке разнообразия типов данных в физической схеме базы данных принимается во внимание следующие два фактора:

а) видовое богатство, то есть количество типов данных, входящих в физическую схему БД;

б) выравненность или равномерность распределения обилия типов данных в физической схеме БД.

Для оценки разнообразия типов данных в физической схеме базы данных воспользуемся следующими мерами доминирования, учитывающими выравненность: индекс разнообразия Симпсона, индекс разнообразия Шеннона, индекс выравненности Симпсона, индекс выравненности Пиелу.

Индекс разнообразия Симпсона (D) рассчитывается по формуле:

$$D = \frac{1}{\sum_{i=1}^{S} p_i^2}, \qquad (2.1)$$

где S – количество типов данных в физической схеме БД (видовое богатство); p_i – доля i-го типа данных в суммарной численности полей всех типов.

Чем больше индекс разнообразия Симпсона приближается к видовому богатству S, тем разнообразнее (с точки зрения используемых типов данных) рассматриваемая физическая схема БД.

В таблице 2.1-2.2 приведены данные для расчета индекс разнообразия Симпсона для физических схем учебных MySQL баз данных catalog и northwind (dev.mysql.com).

Таблица 2.1 - Данные для расчета индекса разнообразия Симпсона для базы данных catalog

№	Тип данных	p_i	Количество полей данного типа
1	BLOB	0,025974	2
2	DATE	0,012987	1
3	INT	0,519481	40
4	SMALLINT	0,025974	2
5	TEXT	0,116883	9
6	TINYINT	0,012987	1
7	VARCHAR	0,272727	21
8	YEAR	0,012987	1
Видовое богатство S =8			77

Таблица 2.2 - Данные для расчета индекса разнообразия Симпсона для базы данных northwind

№	Тип данных	p_i	Количество полей данного типа
1	BLOB	0.022472	2
2	DATE	0.056180	5
3	DECIMAL	0.033708	3
4	DOUBLE	0.011236	1
5	INT	0.033708	3
6	MEDIUMINT	0.044944	4
7	SMALLINT	0.089888	8
8	TEXT	0.044944	4
9	TINYINT	0.078652	7
10	VARCHAR	0.584270	52
Видовое богатство S =10			89

Тип данных считается доминирующим в физической схеме БД, если его количество составляет 50 и более % от всего числа рассматриваемых типов, редким – если менее 10 %, а уникальным – менее 2 % (рисунок 2.1).

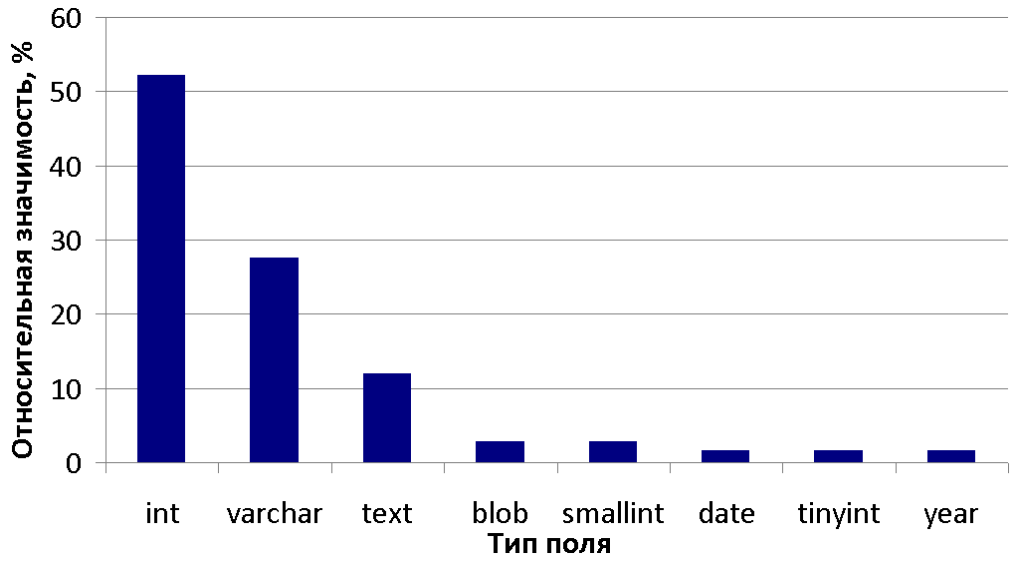


Рисунок 2.1. Кривая значимости типов данных физической схемы БД catalog

Индекс разнообразия Шеннона (H) рассчитывается по формуле:

$$H = -\sum_{i=1}^{S} p_i \ln p_i \,. \tag{2.2}$$

Чем больше значения индекса Шеннона, тем выше разнообразие типов данных в физической схеме БД.

Индекс выравненности Симпсона (E) рассчитывается по формуле:

$$E = \frac{D}{S}. \tag{2.3}$$

Чем больше E приближается к единице, тем равномернее представлены типы данных в физической схеме базы данных.

Индекс выравненности Пиелу (e) рассчитывается на основе индекса Шеннона:

$$e = \frac{H}{\ln S}. \tag{2.4}$$

Индекс Пиелу характеризует выравненность типов данных в физической схеме БД. Величина индекса Пиелу изменяется от 0 до 1. Чем более равномерно представлены в физической схеме БД составляющие ее типы данных, тем ближе его значение к единице.

В таблице 2.3 приведены показатели разнообразия типов данных для различных физических схем учебных и промышленных БД.

Таблица 2.3 – Показатели разнообразия типов данных для физических схем БД

Наименование физической схемы БД	**Акроним**	**Назначение БД**	*S*	*D*	*H*	*E*	*e*
flight	FL	учебная	9	1.9692	0.9508	0.4923	0.6858
world	WO	учебная	5	3.0316	1.3115	0.6063	0.8149
music	MU	учебная	4	2.1304	0.9911	0.5326	0.7149
employees	EM	учебная	5	3.2799	1.3478	0.6559	0.8374
university	UN	учебная	4	2.0864	0.9693	0.5216	0.6992
classicmodels	CL	промышленная	8	2.5652	1.3559	0.3207	0.652
retailer	RE	учебная	8	2.5728	1.3659	0.3216	0.6568
chinook	CH	учебная	4	2.3406	0.9907	0.5851	0.7146
contracts	CO	учебная	6	3.0422	1.3281	0.5070	0.7413
northwind	NO	учебная	10	2.7342	1.5354	0.2734	0.6668
sakila	SA	учебная	14	5.4359	1.9983	0.3883	0.7572
catalog	CA	учебная	8	2.7797	1.3044	0.3475	0.6273
moodle	MO	промышленная	14	3.0758	1.5446	0.2197	0.5853
kadr_oop	KO	промышленная	10	2.6165	1.3174	0.2617	0.5722

Сравнительный анализ показателей разнообразия типов данных для физических схем БД (табл. 2.3), показывает, что по значениям индексов учебные БД не выделяются на фоне промышленных БД, а также значения некоторых индексов подчиняется определенным правилам.

Так, значение индекса выравненности Симпсона для рассмотренных физических схем баз данных различной сложности (рис. 2.2) находится в интервале от 0.2197 до 0.65597. Значение индекса выравненности Пиелу лежит в интервале от 0.57215 до 0.83742.

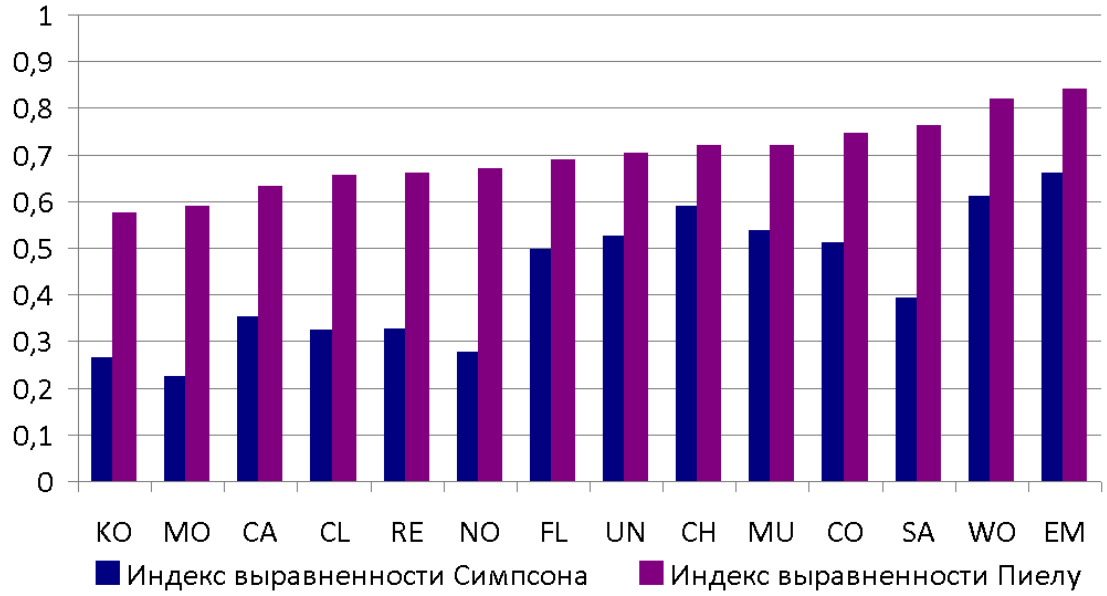


Рисунок 2.2. Индексы выравненности Симпсона и Пиелу

2.3. Метрики подобия баз данных по составу типов полей

Для решения задачи оценки сходства двух баз данных по составу в типов данных в физических схемах применим коэффициенты подобия Жаккара и Серсена-Чекановского.

Для вычисления сходства по качественным признакам, воспользуемся индексом сходства видового состава Жаккара, который рассчитывается как:

$$I_J = \frac{a}{a+b+c}, \tag{2.5}$$

где a – число общих типов данных для двух баз данных, b – число типов, имеющихся только в первой базе данных, c – число типов имеющихся только во второй базе данных.

Для оценки сходства количественных признаков воспользуемся индексом Сересена-Чехановского, рассчитываемым как:

$$I_s = \frac{2\sum \min(x_{ai}; x_{bi})}{\sum x_{ai} + \sum x_{bi}}, \tag{2.6}$$

где x_{ai} и x_{bi} – количество полей i-го типа в первой и второй базе данных; $\min(x_{ai}; x_{bi})$ – наименьшее из значений величин обилия i-го типа в сравниваемых базах данных.

По индексу Жаккара состава по типам полей для баз данных catalog и northwind сходен на 64%:

$$I_J = \frac{7}{7+1+3} = 0.64.$$

По индексу Серсена-Чекановского состав по типам полей для БД catalog и northwind сходен на 41%, поэтому в данном случае можно говорить о низкой степени сходства двух сравниваемых баз данных:

$$I_S = \frac{2 \cdot (2+1+0+0+3+0+2+4+1+21+0)}{(77+89)} = 0.41.$$

В таблице 2.4 представлены индексы подобия для физических схем БД различной сложности.

Таблица 2.4 - Индексы подобия Жаккара и Серсена-Чекановского для физических схем БД

Акроним БД	FL	WO	MU	EM	UN	CL	RE	CH	CO	NO	SA	CA	MO	KO
FL	-	0.29	0.33	0.29	0.60	0.20	0.20	0.00	0.11	0.06	0.13	0.04	0.06	0.08
WO	0.60	-	0.29	0.33	0.29	0.18	0.18	0.13	0.22	0.07	0.27	0.10	0.27	0.07
MU	0.27	0.21	-	0.13	0.30	0.05	0.08	0.03	0.02	0.17	0.29	0.04	0.13	0.08
EM	0.26	0.43	0.13	-	0.29	0.30	0.30	0.29	0.57	0.20	0.27	0.24	0.19	0.25
UN	0.62	0.52	0.33	0.18	-	0.20	0.20	0.00	0.11	0.07	0.20	0.08	0.06	0.08
CL	0.08	0.12	0.09	0.36	0.07	-	0.45	0.20	0.27	0.64	0.22	0.51	0.38	0.38
RE	0.08	0.12	0.20	0.36	0.07	0.88	-	0.33	0.56	0.70	0.47	0.56	0.38	0.50
CH	0.00	0.09	0.14	0.25	0.00	0.70	0.75	-	0.43	0.52	0.29	0.64	0.29	0.27
CO	0.05	0.13	0.11	0.39	0.02	0.71	0.78	0.86	-	0.56	0.33	0.61	0.33	0.45
NO	0.17	0.15	0.17	0.25	0.17	0.50	0.80	0.27	0.45	-	0.50	0.41	0.50	0.43
SA	0.03	0.08	0.29	0.10	0.06	0.44	0.51	0.44	0.44	0.63	-	0.33	0.47	0.26
CA	0.20	0.18	0.09	0.30	0.33	0.45	0.78	0.20	0.40	0.64	0.47	-	0.29	0.38
MO	0.00	0.01	0.01	0.01	0.00	0.05	0.05	0.05	0.05	0.09	0.11	0.04	-	0.50
KO	0.01	0.02	0.00	0.09	0.00	0.22	0.24	0.26	0.29	0.30	0.27	0.28	0.14	-

Акроним БД

В верхней части таблицы 2.4 приведены значения коэффициентов сходства Жаккара (I_J), в нижней – значения коэффициентов сходства

Серсена-Чекановского (I_S). Например, из таблицы 2.4 следует, что при оценке сходства проектов БД *RE, CH, CO* с проектом базы данных *MO* значения индексов I_S, учитывающих качественные признаки, равны: $I_S(RE,MO) = I_S(CH,MO) = I_S(CO,MO) = 0.05$. В то же время, при оценке сходства проектов БД *RE, CH, CO* с проектом базы данных *MO* значения индексов I_J, учитывающих количественные признаки, имеют совершенно разные значения: $I_J(RE,MO) = 0.38$, $I_J(CH,MO) = 0.29$, $I_J(CO,MO) = 0.33$.

В качестве меры, одновременно учитывающей качественные и количественные признаки сходства БД по составу типов полей, примем следующий интегральный индекс:

$$I = w_1 I_J + w_2 I_S, \tag{2.7}$$

где w_1, w_2 – весовые коэффициенты для индексов Жаккара и Серсена-Чекановского.

Данная метрика может рассматриваться в качестве дополнительного показателя в процессе автоматизированного анализа на наличие плагиата в исходных кодах физических схем баз данных. А также разделения БД по схожим свойствам в целях исследования качества проектов баз данных.

Рассмотренные показатели разнообразия типов данных *(2.1)–(2.4)* в физических схемах БД дополняют существующие на данный момент метрические характеристики баз данных. Дополнительные исследования закономерностей распределения значений показателей разнообразия типов данных на больших коллекциях баз данных позволят сформировать набор правил для качественной оценки физических схем баз данных.

Предложенная система количественных показателей *(2.5)–(2.7)* расширяет множество критериев для анализа на наличие плагиата в исходном программном кода.

3. ОЦЕНКА СЛОЖНОСТИ ФИЗИЧЕСКИХ СХЕМ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Метрики являются полезными механизмами для улучшения качества программных продуктов [3]. К сожалению, почти все показатели ориентированы на оценку качества программного кода и практически не применимы к базам данных (БД). Традиционными индикаторами "качества" реляционных баз данных являются оценка нормализации схемы базы данных и верификация содержащейся в базе данных информации [2, 3, 4, 5].

В то же время, контроль качества схемы базы данных невозможен без числовых показателей. При отсутствии количественных измерений трудно принимать какие-либо проектные решения [6]. Знание размера, сложности и качества унаследованных БД необходимо для их преобразования и повторного использования. Для построения объективного представления о базе данных необходимо использовать связанный набор метрик, которые будут отражать целостное представление о качестве базы данных [7, 8].

База данных является ядром информационной системы. Применение количественных метрик физических схем баз данных (БД) позволяет разработчикам БД:

– изучить сложность разработанной физической схемы базы данных;

– оценить объем работ, выполненных разработчиком физической схемы БД;

– оценить усилия по реализации физической схемы БД;

– выбрать наилучшую физическую схему БД из нескольких альтернативных вариантов.

В настоящее время актуальной является задача измерения количественных метрик физических схем БД.

3.1. Получение количественных метрик физической схемы реляционной базы данных

Получение количественных метрик физической схемы реляционной базы данных рассмотрим для наиболее распространенной СУБД *MySQL*, используемой при проектировании веб-ориентированных информационных систем любой сложности [2, 3].

Можно выделить два направления получения количественных метрик физических схем баз данных:

- получение количественных метрик БД в терминах реляционной модели с помощью *SQL*-запросов в *СУБД MySQL*;

- получение количественных метрик БД в терминах теории графов на концептуальномграфе физической схемы БД.

Приведем для примера физическую схему БД веб-ориентированной информационной системы «Гостиница» (рис. 3.1).

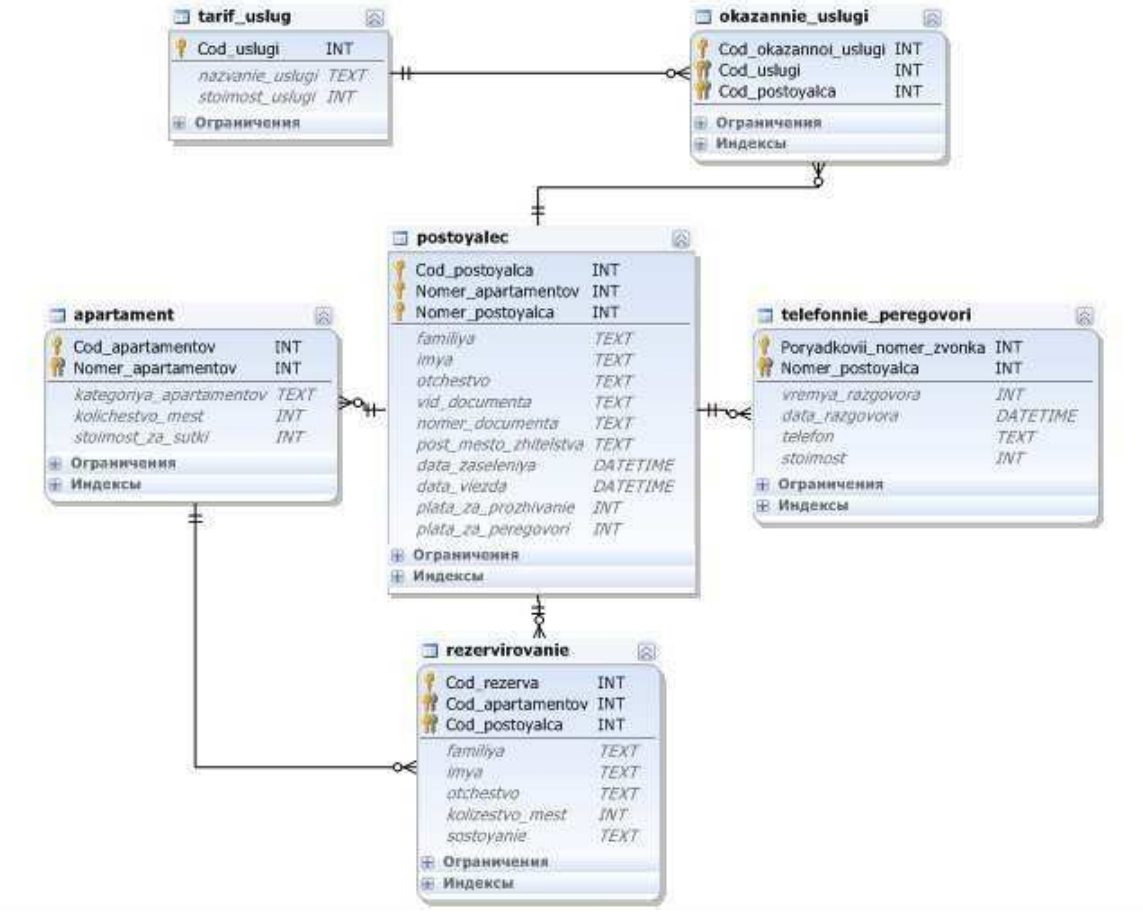


Рисунок 3.1. Физическая схема БД «Гостиница»

3.2. Получение количественных метрик БД в терминах реляционной модели

Получение количественных метрик БД в терминах реляционной модели с помощью SQL-запросов в СУБД MySQL возможно на основе служебной базы данных INFORMATION_SCHEMA.

INFORMATION_SCHEMA – база данных, которая хранит информацию о всех других базах данных, находящихся на сервере MySQL. В состав INFORMATION_SCHEMA входят следующие таблицы:

- CHARACTER_SETS: предоставляет информацию о доступных наборах символов;
- COLLATION_CHARACTER_SET_APPLICABILITY: показывает, какой набор символов применим для сортировки;
- COLLATIONS: предоставляет информацию о сортировке для каждой кодировки;
- COLUMN_PRIVILEGES: предоставляет информацию о привилегиях столбцов;
- COLUMNS: предоставляет информацию о столбцах в таблицах;
- ENGINES: предоставляет информацию о механизмах хранения;
- EVENTS: предоставляет информацию о запланированных событиях;
- FILES: предоставляет информацию о файлах, в которых хранятся табличные данные MySQL;
- GLOBAL_VARIABLES и SESSION_VARIABLES: предоставляют информацию о переменных состояния сервера;
- GLOBAL_STATUS и SESSION_STATUS: предоставляют информацию о состоянии сервера;
- KEY_COLUMN_USAGE: описывает, у каких ключевых столбцов есть ограничения;
- PARTITIONS: предоставляет информацию о табличных разделах;

- PLUGINS: предоставляет информацию о плагинах сервера;

- REFERENTIAL_CONSTRAINTS: предоставляет информацию о внешних ключах;

- ROUTINES: предоставляет информацию о сохраненных подпрограммах (и процедуры и функции);

- SCHEMA_PRIVILEGES: предоставляет информацию о привилегиях схемы базы данных;

- SCHEMATA: предоставляет информацию о базах данных;

- STATISTICS: предоставляет информацию о табличных индексах;

- TABLE_CONSTRAINTS: описывает связи таблиц;

- TABLE_PRIVILEGES: предоставляет информацию о привилегиях таблиц;

- TABLES: предоставляет информацию о таблицах в базах данных;

- TRIGGERS: предоставляет информацию о триггерах;

- USER_PRIVILEGES: предоставляет информацию о глобальных привилегиях;

- VIEWS: предоставляет информацию о представлениях в базах данных.

Рассмотрим SQL-запросы для получения исходных количественных метрик физической схемы БД.

Запрос №1. Получение количества таблиц схемы БД:

```
SELECT tables.table_schema AS "Имя схемы БД",
  count(tables.table_name) AS "Количество таблиц схемы БД"
  FROM information_schema.tables
   WHERE tables.table_schema = <имя БД>
```

Запрос №2. Получение количества атрибутов схемы БД:

```
SELECT tables.table_schema AS "Имя схемы БД",
  count(columns.column_name) AS "Количество атрибутов схемы БД"
  FROM information_schema.tables,
```

```
        information_schema.columns
    WHERE tables.table_name = columns.table_name
        AND tables.table_schema = <имя БД>
```

Запрос №3. Получение количества внешних ключей схемы БД:

```
SELECT  table_constraints.constraint_schema  AS "Имя схемы БД",
    count(table_constraints.constraint_type)
    AS "Количество внешних ключей схемы БД"
    FROM information_schema.table_constraints,
            information_schema.tables
    WHERE table_constraints.constraint_type = "FOREIGN KEY"
        AND table_constraints.table_name = tables.table_name
        AND tables.table_schema = <имя БД>
```

Запрос №4. Получение коэффициента внешних связей схемы БД:

```
SELECT tables.table_schema AS "Имя схемы БД",
  (SELECT count(table_constraints.constraint_type)
    FROM information_schema.table_constraints,
            information_schema.tables
    WHERE table_constraints.constraint_type = "FOREIGN KEY"
    AND table_constraints.table_name = tables.table_name
    AND tables.table_schema = <имя БД>) / count(columns.column_name)
   AS "Коэффициент внешних связей схемы БД"
   FROM information_schema.tables,
           information_schema.columns
   WHERE tables.table_name = columns.table_name
    AND tables.table_schema = <имя БД>
```

Метрические характеристики физической схемы БД «Гостиница», представленной на рисунке 3.1, приведены в таблицах 3.1-3.2.

Таблица 3.1 - Метрические характеристики физической схемы БД «Гостиница»

Метрическая характеристика	**Значение**
Количество таблиц схемы БД	6
Количество атрибутов схемы БД	38
Связность схемы БД	36
Коэффициент нормализации схемы БД	1
Количество внешних ключей схемы БД	6
Глубина дерева связей схемы БД	2
Коэффициент внешних связей схемы БД	0.158

Таблица 3.2 - Метрические характеристики физической схемы БД «Гостиница»

Метрическая характеристика	**Количество атрибутов**	**Количество внешних ключей**
Среднее значение	6.3	1
Минимальное значение	3	1
Максимальное значение	13	2
Стандартное отклонение	3.78	0.89

Рассмотрим SQL-запросы для получения исходных количественных метрик для таблиц физической схемы БД.

Запрос №5. Получение количества атрибутов в таблице.

```
SELECT tables.table_name AS "Имя таблицы",
  count(columns.column_name) AS "Количество атрибутов"
  FROM information_schema.tables, information_schema.columns
  WHERE tables.table_name = columns.table_name
      AND tables.table_schema = <имя БД>
  GROUP BY tables.table_name
```

Запрос №6. Получение количества внешних ключей в таблице.

```
SELECT table_constraints.table_name AS "Название таблицы",
  count(table_constraints.constraint_type)
  AS "Количество внешних ключей"
```

```
FROM information_schema.table_constraints,
        information_schema.tables
WHERE table_constraints.constraint_type = 'FOREIGN KEY'
  AND table_constraints.table_name = tables.table_name
  AND tables.table_schema = <имя БД>
GROUP BY table_constraints.table_name
```

Метрические характеристики для таблиц физической схемы «Гостиница» , представленной на рисунке 3.1, приведены в таблицах 3.3-3.4.

Таблица 3.3 - Метрические характеристики таблиц БД «Гостиница»

Таблица БД	**Количество атрибутов таблицы**	**Количество внешних ключей таблицы**	**Глубина деревасвязей таблицы**	**Коэффициент внешних связей таблицы**
apartament	5	1	1	0.20
okazanie_uslugi	3	2	0	0.67
postoyalec	13	0	2	0
rezervirovanie	8	2	0	0.25
tarif_uslug	3	0	1	0
telefonnie_peregovori	6	1	0	0.16

Таблица 3.4 - Метрические характеристики таблиц БД «Гостиница»

Таблица БД	**Количество атрибутов в составе первичного ключа**	**Количество внешних ключей в составе первичного ключа**	**Коэффициент идентифици-рующих атрибутов таб-лицы**	**Коэффициент внешних ключей в составе первичного ключа**
apartament	2	1	0.4	0.5
okazanie_uslugi	3	2	1	0.67
postoyalec	3	0	0.231	0
rezervirovanie	3	2	0.375	0.67
tarif_uslug	1	0	0.33	0
telefonnie_peregovori	2	1	0.33	0.5

3.3. Получение количественных метрик БД в терминах реляционной модели

Получение количественных метрик БД в терминах теории графов возможно в концептуальном графе физической схемы БД.

Концептуальный граф, соответствующий физической схеме БД «Гостиница», представлен на рисунке 3.2.

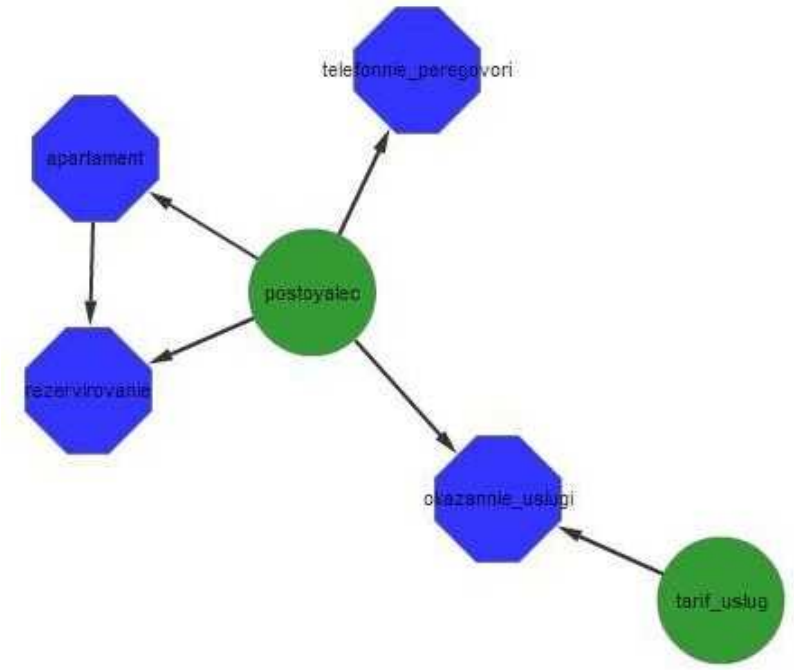
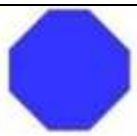


Рисунок 3.2. Концептуальный граф схемы БД

Конструктивные элементы концептуального графа для физической схемы БД приведены в таблице 3.5.

Таблица 3.5 - Конструктивные элементы концептуального графа физической схемы БД

Обозначение	Пояснение
	Зависимая таблица физической схемы БД
	Независимая таблица физической схемы БД
	Связь между таблицами физической схемы БД

Для описания *метрических характеристик физической схемы БД*, представленной в виде концептуального графа G, могут применяться следующие метрики графов:

1) Порядок концептуального графа физической схемы БД (количество вершин):

$$n(G) = n.$$

2) Размер концептуального графа физической схемы БД (количество ребер):

$$s(G) = m.$$

3) Диаметра концептуального графа физической схемы БД – длина максимального пути между вершинами концептуального графа, выраженная числом дуг, составляющих этот путь:

$$diam(G) = \max_{d_{ij} \in D} d_{ij}.$$

4) Структурная избыточность $R(G)$ концептуального графа физической схемы БД показывает превышение общего количества связей между вершинами графа G над минимальным количеством связей:

$$R(G) = \frac{m}{n-1} - 1.$$

5) Реберная плотность $Q(G)$ – характеризует близость концептуального графа G к полносвязному графу:

$$Q(G) = \frac{2m}{n \cdot (n-1)}.$$

6) Абсолютная глубина концептуального графа $H'(G)$: где $N_{j \in P}$ – длина j-го пути, принадлежащего множеству всех путей P в концептуальномграфе G:

$$H'(G) = \sum_{j}^{|P|} N_{j \in P}.$$

7) Средняя глубина концептуального графа $h(G)$:

$$h(G) = \frac{1}{|P|}\sum_{j}^{|P|} N_{j \in P}.$$

Количественные метрики БД в терминах теории графов для концептуального графа физической схемы БД представлены в таблице 3.6.

Таблица 3.6 - Метрики концептуального графа физической схемы БД

Метрическая характеристика	**Значение**
Порядок концептуального графа	6
Размер концептуального графа	6
Диаметр концептуального графа	2
Структурная избыточность концептуального графа	0.2
Реберная плотность концептуального графа	0.4
Абсолютная глубина концептуального графа	8
Средняя глубина концептуального графа	1.14

Рассмотренные подходы к автоматизированному получению количественных метрик, описывающих физическую схему БД, могут быть положены в основу веб-ориентированной информационной системы количественной оценки физических схем БД.

3.4. Оценка сложности реляционных баз данных

Как уже было рассмотрено ранее, метрики базы данных можно вычислять автоматизированно, что гарантирует точность и повторяемость таких измерений, а также позволяет строить на их основе различные аналитические отчеты [9].

В MySQL с помощью запросов к информационной базе данных INFORMATION_SCHEMA, которая хранит информацию относительно всех других баз данных, могут быть получены следующие исходные количественные метрики для i-ой таблицы физической схемы базы данных: количество атрибутов в таблице ($m_{i.1}$); количество ключей, включая уникальные индексы ($m_{i.2}$); количество внешних ключей ($m_{i.3}$); количество не-

уникальных индексов ($m_{i.4}$); количество различных типов данных ($m_{i.5}$); количество атрибутов, входящих в состав первичного ключа ($m_{i.6}$); количество атрибутов, входящих в состав уникальных индексов ($m_{i.7}$); количество атрибутов, входящих в состав неуникальных индексов ($m_{i.8}$); количество атрибутов, входящих в состав внешних ключей ($m_{i.9}$); количество таблиц-родителей ($m_{i.10}$); количество таблиц-потомков ($m_{i.11}$).

Данные метрики могут быть использованы для оценки сложности физической схемы базы данных.

Рассмотрим уже существующий алгоритм оценки сложности базы данных [8], в котором для каждой таблицы БД вычисляется вес сложности Wi по формуле: $W_i = m_{i.1} + m_{i.2} + m_{i.3} + m_{i.4}$. Сложность физической схемы базы данных C вычисляется как сумма весов сложности её таблиц: $C = \sum W_i$.

В алгоритме учитываются не все метрические характеристики, а также не учитывается степень влияния каждого параметра на полученный результат.

В работе [9] предлагается модель с большим количеством метрик, но остается нерешенной задача получения коэффициентов влияния этих метрик на сложность базы данных. Актуальной является задача построения модели оценки сложности базы данных с учетом весовых коэффициентов метрик.

3.4.1. Модель оценки сложности физической схемы базы данных

Предлагается следующая модель для оценки сложности физической схемы базы данных:

$$C = \sum W_i ,$$

где $W_i = \alpha_1 m_{i.1} + \alpha_2 m_{i.2} + \alpha_3 m_{i.3} + \alpha_4 m_{i.4} + \alpha_5 m_{i.5}$.

Коэффициенты α_k определим с помощью процедуры Саати [10]. Матрица парных сравнений метрик оценки сложности таблиц БД с оценками согласованности ИС=0.1259 и ОС=0.1124 приведена в таблице 3.7.

Таблица 3.7 - Матрица парных сравнений метрик сложности таблиц БД

Метрики таблицы БД	m_1	m_2	m_3	m_4	m_5	**Вектор приоритетов**	α_k
m_1	1	1/5	1/7	5	3	0.844	0.105
m_2	5	1	1/3	9	7	2.537	0.315
m_3	7	3	1	9	5	3.936	0.489
m_4	1/5	1/9	1/9	1	1/3	0.242	0.030
m_5	1/3	1/7	1/5	3	1	0.491	0.061
Сумма	13.533	4.454	1.787	27.000	16.333	8.050	1

На основании оценки сложности физической схемы БД можно судить о трудоемкости проектных работ, выполненных разработчиком БД. Например, сложность физической схемы БД employees в 1.577 раз больше, чем сложность физической схемы БД music (рис. 3.3). На основании этого можно косвенно оценить, что трудоемкость работ по созданию БД employees примерно в 1.577 раз больше, чем для БД music.

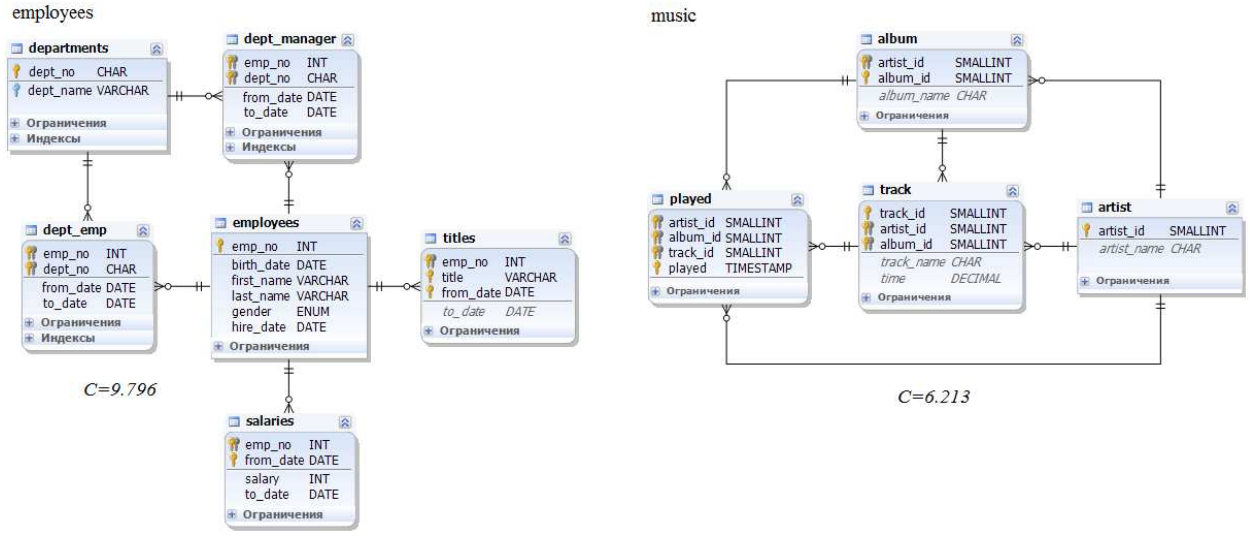


Рисунок 3.3. Физические схемы БД employees и БД music

В таблице 3.8 приведены результаты расчета сложности для физических схем различных проектов БД по предлагаемой модели.

Таблица 3.8 - Результаты расчета сложности различных физических схем БД

Наименование проекта БД	**Аконим**	**Сложность БД**	**Нормализованное значение сложности БД**
flight	FL	4.894	0.0262
world	WO	5.113	0.0274
music	MU	6.213	0.0333
employees	EM	9.796	0.0525
university	UN	8.589	0.0600
classicmodels	CL	13.843	0.0741
retailer	RE	14.920	0.0787
chinook	CH	16.900	0.0905
contracts	CO	18.734	0.1003
northwind	NO	23.151	0.1240
sakila	SA	35.246	0.1887
catalog	CA	29.576	0.1584

3.4.2. Сравнительный анализ полученных результатов

Выполним сопоставление полученных по модели оценок сложности физических схем баз данных и количества информации, содержащейся в sql-скриптах этих базы данных. Для проведения количественного анализа sql-срипта физической схемы базы данных воспользуемся понятием информационной энтропии. Информационная энтропия для случайных и независимых значений x_i с M возможными состояниями рассчитывается по формуле: $H(x) = -\sum_{i=1}^{M} p_i \cdot \log_2 p_i$.

Для исследования sql-срипта физической схемы базы данных с помощью понятия информационной энтропии необходимо определить роль и диапазон значений величины *x*. Sql-срипт базы данных на самом низком уровне своей реализации представляет собой поток байтов, в котором x

может принимать значения в диапазоне [0;255], следовательно, в роли вероятностей p_i будет выступать частота повторов конкретного байта.

Информационная энтропия – это статистический параметр, который показывает вероятность встречаемости определённых байтов в файле. На рисунке 3.4 приведена гистограмма распределения повторов одинаковых байтов в sql-скрипте физической схемы базы данных, которая позволяет визуально оценить информационную энтропию.

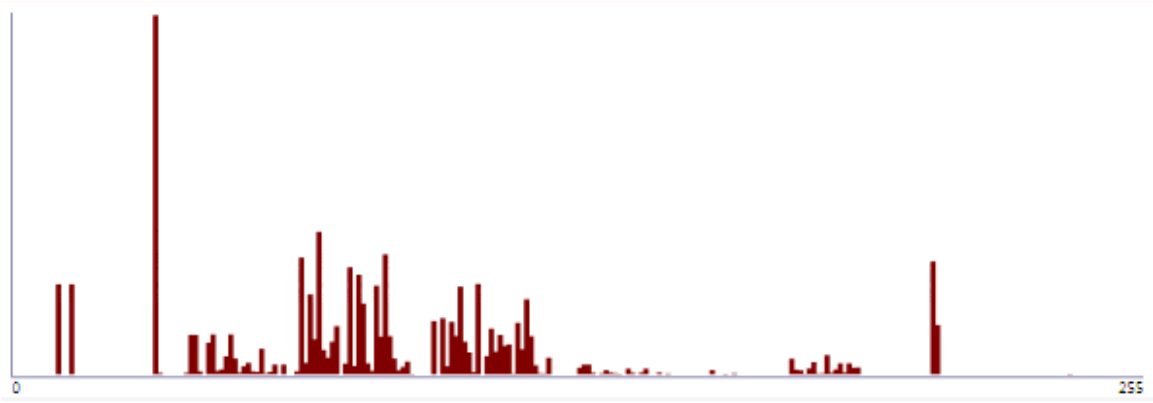


Рисунок 4. – Гистограмма sql-скрипта физической схемы БД Chinook:
ось X - значение байта, ось Y - количество байтов с заданным значением

Количество информации в сообщении, содержащем n символов I_n, по Шеннону равно: $I_n = -n \sum_{i=1}^{M} p_i \cdot \log_2 p_i$.

В таблице 3.9 приведены результаты расчета энтропии и количества информации для sql-скриптов физических схем различных проектов БД.

Таблица 3.9 - Результаты расчета информационной энтропии
для различных проектов БД

Наименование проекта БД	**Аконим**	**Энтропия sql-скрипта, бит**	**Количество информации в sql-скрипте, бит**	**Нормализованное значение количества информации**
flight	FL	5.8862	15733.8102	0.0351
world	WO	5.8200	16703.3192	0.0373
music	MU	5.7430	18061.8459	0.0403

employees	EM	5.6903	19751.0573	0.0440
university	UN	5.7335	22051.0135	0.0492
classicmodels	CL	5.6597	32916.7644	0.0734
retailer	RE	5.6084	35714.4131	0.0797
chinook	CH	5.6008	47589.8962	0.1061
contracts	CO	5.8200	52246.0959	0.1165
northwind	NO	5.5462	53099.1014	0.1184
sakila	SA	5.4819	66430.1994	0.1482
catalog	CA	5.6548	68084.1489	0.1518

Сопоставление оценок сложности физической схемы БД и количества информации, содержащейся в sql-скрипте БД, приведено на рисунке 3.5.

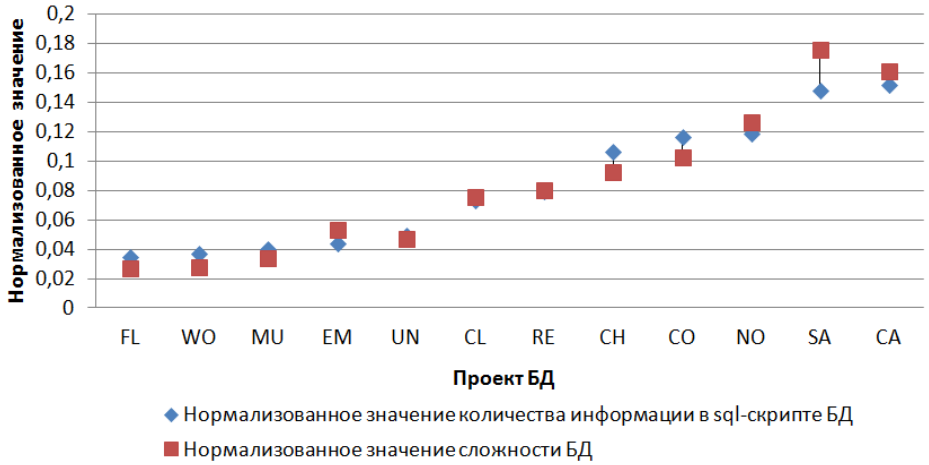


Рисунок 3.5. – Сравнительный анализ нормализованных значений сложности БД и соответствующих им значений, количества информации sql-скрипта БД

Значение коэффициента корреляции Пирсона r=0.9662 говорит о наличии сильной связи между нормализованными значениями оценок сложности БД и количеством информации в sql-скрипте БД .

Наиболее точную оценку сложности реляционной базы данных можно получить в результате анализа её физической схемы. Предложенная модель оценки сложности физической схемы БД может быть использована для объективной косвенной оценки трудоемкости выполненных разработчиком базы данных проектных работ. Рассмотренный подход к автомати-

зированному получению количественных метрик, описывающих физическую схему реляционной БД, может быть взят за основу веб-ориентированной информационной системы количественной оценки физических схем реляционных базы данных.

4. ОЦЕНКА ПОЛНОТЫ И НАДЕЖНОСТИ БАЗ ДАННЫХ

Столкнувшись с необходимостью иметь под рукой базу данных по какой-либо теме, пользователь часто оказывается озадачен тем, какую (или какие) базы ему лучше приобрести, чем та или иная база отличается от других в лучшую или худшую сторону. При этом ключевыми сравниваемыми параметрами баз данных можно считать следующие:

- полнота базы (т.е. наличие всех сведений по теме);

- качество навигационного сопровождения (т.е. легкость и точность нахождения нужной информации в базе);

- оперативность обновления (т.е. темп пополнения базы свежими данными).

И если два последних параметра любой пользователь может самостоятельно оценить по паспортным данным и по демонстрационной версии базы, то о полноте судить значительно труднее (хорошо, если покупатель знает десяток-другой объектов или документов, обязанных присутствовать в базе данной тематики, и может убедиться в том, сколько из них наличествует в этой конкретной базе), чаще всего единственным источником информации о полноте является честное слово разработчика базы. Однако существуют объективные методы оценки полноты баз данных.

4.1. Математическая модель оценки полноты и надежности для двух баз данных

Для построения математической модели оценки полноты и надежности базы данных введем следующие ограничения.

1. Будем называть каждый информационный объект, имеющийся в базе данных (БД), документом. Это может быть математическая формула, описание медицинского препарата, набор сведений о конкретном учебном заведении, реквизиты коммерческой фирмы, статья из научного журнала, адреса и телефоны административных учреждений, нормативный акт и т.п., в зависимости от тематической направленности БД.

2. Каждый документ имеет набор идентификаторов, однозначно его определяющих (например, дата создания документа, создатель документа, исходящий номер, источник информации о документе и т.п.), т.е. в любой БД наличие данного документа можно установить по комплексу его идентификаторов (в частном случае – по какому-либо одному идентификатору, если этот идентификатор – уникальный).

3. Любой документ имеет одни и те же идентификаторы в разных базах данных.

4. Каждая БД комплектуется документами самостоятельно, независимо от других БД, т.е. ни одна БД не является поставщиком первичной информации для другой БД.

В рамках этих положений можно далее рассуждать так. Пусть объективно существует N документов по данной теме. Конкурирующие разработчики БД разыскивают эти документы с разной эффективностью, в результате в базе БД_1 содержится n_1 документов:

$$n_1 = \alpha_1 N, \qquad (4.1)$$

в базе БД_2 содержится n_2 документов:

$$n_2 = \alpha_2 N \qquad (4.2)$$

документов (α_1 и α_2 – эффективность наполнения, т.е. коэффициенты полноты баз БД_1 и БД_2, соответственно). Сравнивая содержимое этих баз, можно обнаружить, что некоторое количество n_{12} документов присутствует в обеих БД. Это количество (при независимости источников пополнения баз) равно:

$$n_{12} = \alpha_1 \alpha_2 N. \tag{4.3}$$

Имея эти три уравнения, нетрудно по фактически измеренным величинам n_1, n_2 и n_{12} вычислить три неизвестных:

$$\begin{aligned} \alpha_1 &= \frac{n_{12}}{n_2}, \\ \alpha_2 &= \frac{n_{12}}{n_1}, \\ N &= \frac{n_1 n_2}{n_{12}}. \end{aligned} \tag{4.4}$$

Все исходные величины (n_1, n_2 и n_{12}) можно понимать как дискретные случайные переменные, для которых стандартное отклонение равно квадратному корню из самой величины, т.е.:

$$\begin{aligned} \Delta n_1 &= \sqrt{n_1}, \\ \Delta n_2 &= \sqrt{n_2}, \\ \Delta n_{12} &= \sqrt{n_{12}}. \end{aligned} \tag{4.5}$$

Относительная ошибка для этих исходных величин составляет:

$$\begin{aligned} \frac{\Delta n_1}{n_1} &= \frac{1}{\sqrt{n_1}}, \\ \frac{\Delta n_2}{n_2} &= \frac{1}{\sqrt{n_2}}, \\ \frac{\Delta n_{12}}{n_{12}} &= \frac{1}{\sqrt{n_{12}}}. \end{aligned} \tag{4.6}$$

Тогда в соответствии с правилами определения относительной ошибки функции от независимых случайных величин получим для относительной ошибки α_1, α_2 и N следующие выражения:

$$\frac{\Delta \alpha_1}{\alpha_1} = \sqrt{\frac{\Delta n_2}{n_2} + \frac{\Delta n_{12}}{n_{12}}} = \sqrt{\frac{1}{n_2} + \frac{1}{n_{12}}}, \tag{4.7}$$

$$\frac{\Delta \alpha_2}{\alpha_2} = \sqrt{\frac{\Delta n_1}{n_1} + \frac{\Delta n_{12}}{n_{12}}} = \sqrt{\frac{1}{n_1} + \frac{1}{n_{12}}}, \tag{4.8}$$

$$\frac{\Delta N}{N} = \sqrt{\frac{1}{n_1} + \frac{1}{n_2} + \frac{1}{n_{12}}}. \tag{4.9}$$

Анализируя последние три формулы, нетрудно видеть, что надежность полученных расчетных данных тем выше, чем больше каждая из трех измеренных величин. И если наименьшая из них (а это n_{12}) мала, то расчет становится малодостоверным.

Таким образом, по результатам анализа всего двух частично перекрывающихся баз можно составить представление и о полноте каждой из них, и об общем количестве реально существующих документов по теме, в том числе и о количестве документов, отсутствующих в данных базах.

4.2. Математическая модель оценки полноты и надежности для трех баз данных

Пусть теперь в нашем распоряжении три БД на одну и ту же тему. В результате анализа их содержимого мы можем получить семь параметров:

n_1 - полное число документов в БД_1;

n_2 - полное число документов в БД_2;

n_3 - полное число документов в БД_3;

n_{12} - число документов, имеющихся и в БД_1, и в БД_2;

n_{13} - число документов, имеющихся и в БД_1, и в БД_3;

n_{23} - число документов, имеющихся и в БД_2, и в БД_3;

n_{123} - число документов, имеющихся во всех трех базах.

Обозначим через α_1, α_2, α_3 и N неизвестные величины коэффициентов полноты баз и полного количества документов по теме, тогда для этих четырех неизвестных у нас имеется семь уравнений, что позволяет значительно повысить точность расчета, и вот каким образом.

Запишем исходные уравнения:

$$n_1 = \alpha_1 N, \qquad (4.10)$$

$$n_2 = \alpha_2 N, \qquad (4.11)$$

$$n_3 = \alpha_3 N, \qquad (4.12)$$

$$n_{12} = \alpha_1 \alpha_2 N, \qquad (4.13)$$

$$n_{13} = \alpha_1 \alpha_3 N, \qquad (4.14)$$

$$n_{23} = \alpha_2 \alpha_3 N, \qquad (4.15)$$

$$n_{123} = \alpha_1 \alpha_2 \alpha_3 N. \qquad (4/16)$$

Для получения решения достаточно использовать только четыре уравнения из этих семи, и таких четверок можно составить:

$$C_7^4 = \frac{7 \cdot 6 \cdot 5 \cdot 4}{1 \cdot 2 \cdot 3 \cdot 4} = 35.$$

Иными словами, можно получить 35 комплектов решений. Не все из них равноценны, и вот почему. Относительная точность определения дискретной случайной величины обратно пропорциональна корню квадратному из этой величины (в соответствии с формулами *(4.6)*), а в расчетных формулах с умножением и делением таких случайных величин их относительные ошибки суммируются, как показано в выражениях *(4.7)*. Если учесть, что все n_i, как правило, больше любого из n_{ij}, а те, в свою очередь, существенно больше n_{ijk}, то отсюда следует, что по возможности следует избегать использования n_{ijk} в расчетах, т.е. не пользоваться последним (седьмым) уравнением из набора *(4.10)*. Тогда число комбинаций уравнений уменьшится до

$$C_6^4 = \frac{6 \cdot 5 \cdot 4 \cdot 3}{1 \cdot 2 \cdot 3 \cdot 4} = 15.$$

Эти 15 уравнений можно разделить на 4 группы, в каждой из которых решения получаются циклической перестановкой индексов:

1. Имеются три системы типа:

$$\begin{cases} n_i = \alpha_i N, \\ n_j = \alpha_j N, \\ n_k = \alpha_k N, \\ n_{ij} = \alpha_i \alpha_j N. \end{cases} \quad \text{Решения:} \quad \begin{aligned} \alpha_i = \frac{n_{ij}}{n_j}, \alpha_j = \frac{n_{ij}}{n_i}, \\ \alpha_k = \frac{n_k n_{ij}}{n_i n_j}, N = \frac{n_i n_j}{n_{ij}}. \end{aligned}$$

2. Имеется шесть систем типа:

$$\begin{cases} n_i = \alpha_i N, \\ n_j = \alpha_j N, \\ n_{ij} = \alpha_i \alpha_j N, \\ n_{ik} = \alpha_i \alpha_k N. \end{cases} \quad \text{Решения:} \quad \begin{array}{c} \alpha_i = \dfrac{n_{ij}}{n_j},\ \alpha_j = \dfrac{n_{ij}}{n_i}, \\ \alpha_k = \dfrac{n_{ik}}{n_i},\ N = \dfrac{n_i n_j}{n_{ij}}. \end{array}$$

3. Имеется три системы типа:

$$\begin{cases} n_i = \alpha_i N, \\ n_j = \alpha_j N, \\ n_{ik} = \alpha_i \alpha_k N, \\ n_{jk} = \alpha_j \alpha_k N. \end{cases} \quad \begin{array}{l} \text{Система неразрешима,} \\ \text{т.к. уравнения линейно} \\ \text{зависимы:} \end{array} \quad \frac{n_i}{n_j} = \frac{n_{ik}}{n_{jk}}.$$

4. Имеется три системы типа:

$$\begin{cases} n_i = \alpha_i N, \\ n_{ij} = \alpha_i \alpha_j N, \\ n_{ik} = \alpha_i \alpha_k N, \\ n_{jk} = \alpha_j \alpha_k N. \end{cases} \quad \text{Решения:} \quad \begin{array}{c} \alpha_k = \dfrac{n_{ij} n_{ik}}{n_i n_{jk}},\ \alpha_j = \dfrac{n_{ij}}{n_i}, \\ \alpha_k = \dfrac{n_{ik}}{n_i},\ N = \dfrac{n_{jk} n_j^2}{n_{ij} n_{ik}}. \end{array}$$

Итак, получено 12 комплектов решений, причем расчетные формулы в этих комплектах частично повторяются:

Итак, получено 12 комплектов решений:

$\alpha_1 = \dfrac{n_{12}}{n_2}$ (5 раз), $\alpha_1 = \dfrac{n_{13}}{n_3}$ (5 раз) , $\alpha_1 = \dfrac{n_1 n_{23}}{n_2 n_3}$ (1 раз), $\alpha_1 = \dfrac{n_{12} n_{13}}{n_1 n_{23}}$ (1 раз).

$\alpha_2 = \dfrac{n_{12}}{n_1}$ (5 раз), $\alpha_2 = \dfrac{n_{23}}{n_3}$ (5 раз), $\alpha_2 = \dfrac{n_2 n_{13}}{n_1 n_3}$ (1 раз), $\alpha_2 = \dfrac{n_{12} n_{13}}{n_2 n_{13}}$ (1 раз).

$\alpha_3 = \dfrac{n_{13}}{n_1}$ (5 раз), $\alpha_3 = \dfrac{n_{23}}{n_2}$ (5 раз), $\alpha_3 = \dfrac{n_3 n_{12}}{n_1 n_2}$ (1 раз), $\alpha_3 = \dfrac{n_{13} n_{23}}{n_3 n_{12}}$ (1 раз).

$N = \dfrac{n_1 n_2}{n_{12}}$ (3 раза), $N = \dfrac{n_1 n_3}{n_{13}}$ (3 раза), $N = \dfrac{n_2 n_3}{n_{23}}$ (3 раза).

$N = \dfrac{n_{23} n_1^2}{n_{12} n_{13}}$ (1 раз), $N = \dfrac{n_{13} n_2^2}{n_{12} n_{23}}$ (1 раз), $N = \dfrac{n_{12} n_3^2}{n_{13} n_{23}}$ (1 раз).

Причем расчетные формулы в этих комплектах частично повторяются. При вычислении среднего значения, дисперсии и среднеквадратичного отклонения каждой из этих величин частота повторения формул должна быть учтена введением соответствующих весовых коэффициентов. В простейшем случае расчет усредненных величин можно проводить по формулам:

$$\alpha_{1m} = \frac{\left(5\frac{n_{12}}{n_2} + 5\frac{n_{13}}{n_3} + \frac{n_1 n_{23}}{n_2 n_3} + \frac{n_{12} n_{13}}{n_1 n_{23}}\right)}{12}, \tag{4.17}$$

$$\alpha_{2m} = \frac{\left(5\frac{n_{12}}{n_1} + 5\frac{n_{23}}{n_3} + \frac{n_2 n_{13}}{n_1 n_3} + \frac{n_{12} n_{13}}{n_2 n_{13}}\right)}{12}, \tag{4.18}$$

$$\alpha_{3m} = \frac{\left(5\frac{n_{13}}{n_1} + 5\frac{n_{23}}{n_2} + \frac{n_3 n_{12}}{n_1 n_2} + \frac{n_{13} n_{23}}{n_3 n_{12}}\right)}{12}, \tag{4.19}$$

$$N_m = \frac{3\left(\frac{n_1 n_2}{n_{12}} + \frac{n_1 n_3}{n_{13}} + \frac{n_2 n_3}{n_{23}} +\right) + \frac{n_{23} n_1^2}{n_{12} n_{13}} + \frac{n_{13} n_2}{n_{12} n_{23}} + \frac{n_{12} n_3^2}{n_{13} n_{23}}}{12}. \tag{4.20}$$

Дисперсия и среднеквадратичное отклонение для α_{1m} рассчитываются по следующим формулам *(4,21)* и *(4.22)*, соответственно:

$$D_{\alpha_1} = \frac{5\left(\alpha_{1m} - \frac{n_{12}}{n_2}\right)^2 + 5\left(\alpha_{1m} - \frac{n_{13}}{n_3}\right)^2 + \left(\alpha_{1m} - \frac{n_1 n_{23}}{n_2 n_3}\right)^2 + \left(\alpha_{1m} - \frac{n_{12} n_{13}}{n_1 n_{23}}\right)^2}{12}, \tag{4.21}$$

$$\Delta\alpha_1 = \sqrt{D_{\alpha_1}}. \tag{4.22}$$

Аналогичные формулы можно записать и для остальных трех переменных.

При наличии сведений о трех базах появляется возможность и заглянуть в «кухню» каждого из разработчиков баз. Дело в том, что если исходные данные для расчета представить в несколько иной форме:

m_1 - число документов, присутствующих только в БД_1,

m_2 - число документов, присутствующих только в БД_2,

m_3 - число документов, присутствующих только в БД_3,

m_{12} - число документов, имеющихся только в БД_1 и в БД_2,

m_{13} - число документов, имеющихся только в БД_1 и в БД_3,

m_{23} - число документов, имеющихся только в БД_2 и в БД_3,

m_{123} - число документов, имеющихся во всех трех базах,

то появляется возможность выявить и недобросовестное поведение разработчика, «откачивающего» документы из чужой базы в свою, и обратную ситуацию - наличие мощного источника информации, доступного только одному из разработчиков, и, наконец, наличие некоего «джентльменского набора» документов, обязанных быть представленными в любой базе данных, относящейся к заявленной теме.

Связь между числами m и n очевидна:

$n_1 = m_1 + m_{12} + m_{13} + m_{123}$,

$n_2 = m_2 + m_{12} + m_{23} + m_{123}$,

$n_3 = m_3 + m_{13} + m_{23} + m_{123}$,

$n_{12} = m_{12} + m_{123}$,

$n_{13} = m_{13} + m_{123}$,

$n_{23} = m_{23} + m_{123}$,

$n_{123} = m_{123}$.

Естественно считать, что для истинно независимых БД должно было выполняться условие, что число «тройных совпадений» (т.е. документов, имеющихся во всех трех базах), должно быть меньше любого из трех чисел «двойного совпадения» (это следует из формулы *(4.16)*). Если же имеет место ситуация, например, когда база БД_1 пополняется из базы БД_2, то документы из массива m_{23} «перекочуют» в массив m_{123}, а из массива m_2 в массив m_{12}. В случае наличия уникального источника информации, доступного только для разработчика БД_1, например, мы получим аномально высокое значение m_1. Наконец, «джентльменский набор» обязан целиком войти в m_{123}, никак не повлияв на содержание остальных исходных данных. И вообще, любое аномальное отклонение какой-либо одной или двух величин несет в себе скрытую информацию о тех или иных технологических особенностях комплектования баз.

Выше упоминалось, что, кроме чисто статистических механизмов комплектования баз данных, существует некоторое количество докумен-

тов, являющихся «индикаторами» тематики базы данных. Например, едва ли кто осмелится назвать базу правовой информации наименованием «Федеральное законодательство России», если в этой базе будет отсутствовать такой документ, как Конституция РФ. Или если в базе данных «Неорганические материалы» будут отсутствовать сведения о простых химических элементах (не соединениях, а именно элементах). С точки зрения задач настоящей работы существование «джентльменского набора» обязательных документов добавляет еще одно неизвестное к нашим четырем и избыточность системы уравнений *(4.10)-(4.16)* уменьшается. Но при этом существенную роль начинает играть именно последнее из этих уравнений, т.к. именно там в максимальной степени сказывается влияние новой неизвестной величины. И сами уравнения с учетом этой неизвестной выглядят теперь несколько иначе:

$$n_1 = \alpha_1 N + x, \qquad (4.24)$$

$$n_2 = \alpha_2 N + x, \qquad (4.25)$$

$$n_3 = \alpha_3 N + x, \qquad (4.26)$$

$$n_{12} = \alpha_1 \alpha_2 N + x, \qquad (4.27)$$

$$n_{13} = \alpha_1 \alpha_3 N + x, \qquad (4.28)$$

$$n_{23} = \alpha_2 \alpha_3 N + x, \qquad (4.29)$$

$$n_{123} = \alpha_1 \alpha_2 \alpha_3 N + x. \qquad (4.30)$$

Если принять тот же способ решения этой системы уравнений, т.е. составлять из семи уравнений возможные комбинации по пять, то в нашем распоряжении оказывается $C_7^5 = \dfrac{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5} = 21$ комбинация уравнений.

Можно, как и раньше, распределить их по следующим группам:

1. Имеются три системы типа:

$$\begin{cases} n_1 = \alpha_1 N + x, \\ n_2 = \alpha_2 N + x, \\ n_3 = \alpha_3 N + x, \\ n_{12} = \alpha_1\alpha_2 N + x \\ n_{13} = \alpha_1\alpha_3 N + x. \end{cases}$$ Решение: $$\alpha_1 = \frac{n_{12} - n_{13}}{n_2 - n_3},$$ $$\alpha_2 = \frac{n_2 - x}{N},$$ $$\alpha_3 = \frac{n_3 - x}{N},$$ $$x = \frac{n_2 n_{13} - n_3 n_{12}}{n_2 - n_3 - n_{12} + n_{13}},$$ $$N = \frac{n_1 - x}{\alpha_1}.$$

2. Имеются три системы типа:

$$\begin{cases} n_1 = \alpha_1 N + x, \\ n_2 = \alpha_2 N + x, \\ n_{12} = \alpha_1\alpha_2 N + x \\ n_{13} = \alpha_1\alpha_3 N + x \\ n_{23} = \alpha_2\alpha_3 N + x. \end{cases}$$ Решение: $$\alpha_1 = \frac{n_1 - x}{N},$$ $$\alpha_2 = \frac{n_2 - x}{N},$$ $$\alpha_3 = \frac{n_{13} - n_{23}}{n_1 - n_2},$$ $$x = \frac{n_1 n_{23} - n_2 n_{13}}{n_1 - n_2 - n_{13} + n_{23}},$$ $$N = \frac{(n_1 - x)(n_2 - x)}{n_{12} - x}.$$

3. Имеются три системы типа:

$$\begin{cases} n_1 = \alpha_1 N + x, \\ n_{123} = \alpha_1\alpha_2\alpha_3 N + x, \\ n_{12} = \alpha_1\alpha_2 N + x \\ n_{13} = \alpha_1\alpha_3 N + x \\ n_{23} = \alpha_2\alpha_3 N + x. \end{cases}$$

Простых решений не видно, а сложные подстановки дадут слишком большую ошибку результата, поэтому эта группа систем исключается из рассмотрения.

4. Имеются шесть системы типа:

$$\begin{cases} n_1 = \alpha_1 N + x, \\ n_2 = \alpha_2 N + x, \\ n_{12} = \alpha_1\alpha_2 N + x \\ n_{13} = \alpha_1\alpha_3 N + x \\ n_{123} = \alpha_1\alpha_2\alpha_3 N + x. \end{cases}$$ Решение: $$\alpha_1 = \frac{n_{12} - x}{n_2 - x},,$$ $$\alpha_2 = \frac{n_{12} - n_{123}}{n_1 - n_{13}},$$

$$\alpha_3 = \frac{n_{13} - x}{n_1 - x}$$
$$x = \frac{n_{12} - n_1\alpha_2}{1 - \alpha_2},$$
$$N = \frac{n_2 - x}{\alpha_2}.$$

5. Имеются три системы типа:

$$\begin{cases} n_1 = \alpha_1 N + x, \\ n_2 = \alpha_2 N + x, \\ n_{13} = \alpha_1\alpha_3 N + x \\ n_{23} = \alpha_2\alpha_3 N + x \\ n_{123} = \alpha_1\alpha_2\alpha_3 N + x. \end{cases}$$

Решение:

$$\alpha_1 = \frac{n_1 - n_{12}}{(n_1 - \alpha_2)N}, ,$$
$$\alpha_2 = \frac{n_{12} - n_{123}}{n_1 - n_{13}},$$
$$\alpha_3 = \frac{n_{13} - n_{123}}{n_1 - n_{12}}$$
$$x = \frac{n_{12} - n_1\alpha_2}{1 - \alpha_2},$$
$$N = \frac{n_{23} - x}{\alpha_2\alpha_3}.$$

Итак, из перечисленных систем можно получить 18 частных решений (учитывающих только часть уравнений общей системы *(4.24)-(4.30)*). Не загромождая текст формулами для вычисления средних значений, дисперсии и среднеквадратичной ошибки, скажем только, что все эти вычисления производятся по той же методике, которая использовалась для оценки полноты баз и полного количества документов.

ЛИТЕРАТУРА

1. *Глухарёв М.Л.* Разработка показателей и методов оценивания корректности реляционных баз данных // Известия Петербургского университета путей сообщения. - 2008. - № 1 (14). - С. 135-148.

2. *Липаев В.* Анализ качества баз данных // Открытые системы. СУБД. - 2002. - № 3. - С. 5

3. *Черняев А.О., Рыбанов А.А.* Разработка и исследование алгоритмов автоматизированного проектирования логических схем реляционных баз данных // *В мире научных открытий*. – 2010. – № 4–11 (10). – С. 128–129.

4. Нидзий А.В., Рыбанов А.А. Исследование метрических характеристик физических схем реляционных баз данных // Научное обозрение. Педагогические науки. – 2019. – № 3– 3. – С. 75– 77.

5. Рыбанов А.А., Свиридова О.В., Короткова Н.Н., Лясин Д.Н., Абрамова О.Ф. Модель оценки сложности физической схемы реляционной базы данных // Инженерный вестник Дона. – 2019. – № 3 (54). – С. 12.

6. Piattini M., Calero C., Genero M. Table oriented metrics for relational Databases // Software Quality Journal. – 2001. – №9 (2). – pp. 79-97.

7. Calero C., Piattini M. and Genero M. A case study with relational database metrics // Proceedings ACS/IEEE International Conference on Computer Systems and Applications, Beirut, Lebanon. – 2001, pp. 485–487. doi: 10.1109/AICCSA.2001.934049

8. Paul R. A, Kunii T. L., Shinagawa Y. and Khan M. F. Software metrics knowledge and databases for project management // in IEEE Transactions on Knowledge and Data Engineering. – vol. 11. no. 1. – pp. 255-264, Jan.–Feb. 1999. doi: 10.1109/69.755633

Приложение №1. Соглашения об именовании объектов баз данных

База данных только тогда становится базой данных, когда её структура оптимально продумана с точки зрения полноты функциональности, быстродействия, стабильности работы, прозрачности сопровождения; данные в базе своевременно пополняются, обновляются, подвергаются процедурам проверки на предмет соблюдения бизнес-правил и минимизации логических аномалий. О требованиях безопасности и говорить не приходится - это безусловное требование. Но и это ещё не всё. База данных должна быть обязательно документирована: наличие логической и физической модели, описание таблиц, полей и других объектов; наличие регламентов обслуживания (SOP - Standard Operating Procedures), регламентов по обновлению и других документов. Таблицы без комплекта документации - набор табличек и не более того. Нет документации - нет системы. Руководящие принципы по проектированию и разработке баз данных способствуют унификации структуры, единообразию в программировании при коллективной работе, облегчению понимания, распространения и сопровождения.

1.ОБЩИЕ ПОЛОЖЕНИЯ ОБ ИМЕНОВАНИЯХ

Для всех пользовательских объектов баз данных и полей принимаются и устанавливаются наименования:

1.1 преимущественно используя *under_score* нотацию;

1.2 полными словами, сочетанием терминов, отражающими предметный смысл;

1.3 на английском языке;

1.4 в нижнем регистре;

1.5 общая длина наименований - до 30 символов;

1.6 слова - в единственном числе, разделяемые символом нижнего подчеркивания (но не пробелами), например:

- *client;*
- *client_address;*
- *client_contact;*
- *branch;*
- *branch_address;*
- *branch_history;*
- *branch_by_manager*.

1.7 Не используйте зарезервированные слова в наименованиях объектов баз данных, так как это может привести к непредсказуемым ситуациям.

2. ИМЕНОВАНИЕ ТАБЛИЦ

Шаблон:

<существительное_в_единственном_числе>

[_<существительное_в_единственном_числе>]

2.1 Таблицы именуются существительными в единственном числе, поскольку таблица подразумевает хранение множественных экземпляров сущности - записей. Кроме того, так сокращается длина наименования таблицы.

2.2 Наименование таблицы должно раскрывать её смысловое предназначение, характеризовать содержащиеся в ней записи без необходимости лишний раз выполнять инструкцию SELECT * FROM table.

2.3 Наименования таблиц не должны иметь префиксов, т.к., во-первых, незачем смущать Бизнес-пользователей различными примесями, а во-вторых, есть понятие схема базы данных.

2.4 Если наименование состоит из более чем одного слова, то ключевое слово располагается первым (см. пример выше), чтобы в списке (в браузере) объектов таблицы визуально компоновались вместе в алфавитном порядке. Последнее условие особенно важно для баз данных с большим количеством таблиц.

3. ИМЕНОВАНИЕ ПРЕДСТАВЛЕНИЙ

Шаблон: <префикс><имя_таблицы_или_смысловое_наименование>

3.1 Для именований представлений устанавливаются такие же положения, как и для таблиц, за исключением префиксов. Чтобы отличить таблицы от представлений, последние должны иметь короткие префиксы, например, *v_* .

3.2 В исключительных случаях для поддержания *legacy* кода, в целях обеспечения обратной совместимости на период миграции систем допускается именовать представления с отступлениями от обозначенных выше правил.

4. ИМЕНОВАНИЕ ХРАНИМЫХ ПРОЦЕДУР

Шаблон: <префикс><имя_таблицы_или_логической_сущности><действие>

4.1 Наименования серверных хранимых процедур должны начинаться с короткого префикса, например, *p_* .

4.2 Наименования серверных хранимых процедур должны оканчиваться такими глаголами, как *get, insert, update, delete, calculate, archive, ...* или одноименными суф-

фиксами *_get, _add, _upd, _del, _calc, _arch, ...*, отражающими их основные действия, например:

- *p_client_address_add;*
- *p_client_address_upd;*
- *p_client_address_del;*
- *p_client_cashback_calculate.*

5. ИМЕНОВАНИЕ ФУНКЦИЙ

Шаблон: <префикс><имя_таблицы_или_логической_сущности><действие>()

5.1 Наименования пользовательских серверных функций должны начинаться с коротких префиксов, например, *f_*.

5.2 Наименования пользовательских функций должны говорить об их функциональности, например:

- *ft_user_details_get();*
- *f_address_parse().*

6. ИМЕНОВАНИЕ ТРИГГЕРОВ

Шаблон: <префикс><имя_таблицы><действие>

6.1 Наименования табличных триггеров должны начинаться с короткого префикса, например, *tr_*, а затем в единственном числе имя таблицы, на которую навешивается триггер.

6.2 Наименования табличных триггеров должны оканчиваться суффиксами *_add, _upd, _del*, отражающими их основные действия, например,

- *tr_client_address_upd;*
- *tr_branch_history_add_upd.*

7. ИМЕНОВАНИЕ ИНДЕКСОВ

Шаблон: <префикс><имя_таблицы>_<имя_поля>[_<имя_поля>]

7.1 Наименования индексов должны начинаться с префиксов:

- *pck_* - PRIMARY CLUSTERED KEY - первичный (единственный уникальный) кластерный ключ ;
- *pnk_* - PRIMARY NONCLUSTERED KEY - первичный (единственный уникальный) некластерный ключ;

- *ack_* - ALTERNATE CLUSTERED KEY - альтернативный (уникальный) кластерный ключ;

- *ank_* - ALTERNATE NONCLUSTERED KEY - альтернативный (уникальный) некластерный ключ;

- *uci_* - UNIQUE CLUSTERED INDEX - уникальный кластерный индекс;

- *ni_* - UNIQUE NONCLUSTERED INDEX - уникальный некластерный индекс.

- *_ci_* - CLUSTERED INDEX – кластерный индекс;

- *_ni_* - NONCLUSTERED INDEX - обычный индекс.

7.2 далее следуют наименование таблицы и, если необходимо, наименования полей индекса, например,

- *pck_client;*

- *uni_branch_code.*

8. ИМЕНОВАНИЕ СВЯЗЕЙ ССЫЛОЧНОЙ ЦЕЛОСТНОСТИ

Шаблон: <префикс><имя_подчиненной_таблицы>__<имя_таблицы>

8.1 Наименования связей ссылочной целостности (Referential Integrity) должны начинаться с короткого префикса, например, ref_ (с возможным, если необходимо, указанием порядкового номера связи). Далее следует имя подчиненной (зависимой) таблицы, а затем - имя старшей таблицы, например:

- *ref_metro__locality;*

- *ref_transaction__account;*

- *ref2_transaction__account.*

9. ИМЕНОВАНИЕ ПОЛЕЙ

9.1 Одинаковые по бизнес-назначению поля в разных таблицах должны иметь по возможности одинаковые системные наименования. В этом случае облегчается понимание базы данных, некоторые ETL-инструменты автоматически строят мэппинги между полями.

9.2 Системные наименования полей-идентификаторов, ключевых полей должны начинаться с префикса *id_*.

Приложение №2. Соглашения об оформлении исходных программных кодов

Хорошо оформленный структурированный программный код: способствует пониманию логики обработки данных; сокращает время поиска и отладки ошибок, минимизирует количество новых ошибок; обеспечивает отчуждаемость кода, передачу кода (а также его приёмку) другим разработчикам; существенно облегчает сопровождение, доработку, а следовательно, снижает операционные затраты; качественно характеризует автора-разработчика (порядок в коде = порядок в голове).

Рекомендуемые правила оформления программных процедур, триггеров, функций, sql-пакетных инструкций (далее - просто процедуры):

1. Наименования пользовательских процедур и функций должны говорить об их функциональном предназначении (см. Соглашения об именовании объектов).

2. Наименования переменных и параметров должны говорить об их функциональном предназначении и быть полными словами на английском языке в нижнем регистре в *under_score* нотации.

3. Переменные должны объявляться DECLARE в одном месте, в начале процедуры, не будет проблем с их областью видимости.

4. Начальные значения переменным должны присваиваться SET в следующих строках сразу после инструкций DECLARE.

5. Должна быть обязательной практика документирования процедур: в начале тела процедуры перед блоком объявления переменных должно быть содержательное, предметное описание процедуры; в комментариях должны быть прописаны ограничения и допущения использования процедур, если таковые имеются; при наличии нескольких необязательных входных параметров желательны примеры вызова; желательны указание авторства, даты / номера версии процедуры.

6. Входные и выходные параметры процедур должны сопровождаться комментариями.

7. Комментарии не влияют на производительность процедур. Простые однострочные комментарии следуют после двойного дефиса --, а блоки текста обрамляются /* ... */. Комментарии важны для бизнес-критичных расчетов, в точках ветвления логики процедур, нетривиальных преобразованиях данных. Комментарии должны раскрывать бизнес цель, ожидаемый результат.

8. Для большей читабельности кода следует отделять между собой логические фрагменты процедуры 2-3 пустыми строками, а отдельные блоки - разделительными комментарными строками из специальных символов.

9. Особое внимание следует уделять форматированию sql-запросов. Современные редакторы среды разработки позволяют один раз настроить шаблон правила форматирования, а затем многократно применять их "в одно касание".

9.1 Так уже стало общепринятой практикой выстраивать запрашиваемые поля вертикально, каждое поле в отдельной строке.

9.2 Запрашиваемые поля следует компоновать в смысловые группы, наиболее значимые поля - первыми (чтобы в выборке были слева), второстепенные поля - последними (чтобы в выборке были справа). Не следует разбрасывать поля как попало, будто на минном поле.

9.3 Зарезервированные слова и системные функции должны быть написаны в верхнем регистре, а наименования пользовательских объектов и полей - в нижнем регистре, чтобы дополнительно фокусировать взгляд читателей.

9.4 Таблицам, представлениям, табличным функциям обязательно назначать короткие алиасы (например, первая буква имени таблицы). Использование псевдонимов позволяет легко подменить в sql-запросе один табличный объект на другой, предотвращает коллизии неодназначного упоминания полей, сокращает объем кода.

9.5 Все поля в sql-запросе должны иметь квалификационную ссылку на табличный объект (<алиас>.).

9.6 Необходимо полностью квалифицировать табличный объект, указывая Database Owner, а отдельных случаях - Server Name, Database Name.

9.7 Задействованные в соединениях (JOIN) табличные объекты очень хорошо выстраивать вертикально, каждый объект в отдельной строке.

9.8 Вертикальные отступы в два пробела между разделами SELECT, FROM & JOIN, WHERE & GROUP BY & HAVING & ORDER BY также способствуют фокусировке взгляда. Вертикальный отступ в один пробел не так заметен, а большее количество отступов сильно смещает весь sql-запрос вправо при наличии вложенных запросов.

9.9 Контекстные комментарии облегчают дальнейшее сопровождение sql-запросов.

10. Потенциально проблемные, недоделанные участки кода можно помечать сигнальными комментариями, например, -- !!!!!!!!!!!!!!!!!!!!!!!!!!!!

11. Большие процедуры следует стараться разбивать на логически обособленные процедуры, которые можно было бы вызывать из других процедур. Не следует перегружать процедуры, пытаясь запрограммировать всё в одной. Декомпозиция кода облегчает его отладку.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

1 ПОКАЗАТЕЛИ И МЕТОДЫ ОЦЕНИВАНИЯ КОРРЕКТНОСТИ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ..... 6

1.1 База данных как программный компонент информационной системы..... 7

1.2 Виды и показатели корректности баз данных..... 10

2 ОЦЕНКА ПОКАЗАТЕЛЕЙ РАЗНООБРАЗИЯ ТИПОВ ДАННЫХ В ФИЗИЧЕСКИХ СХЕМАХ БАЗ ДАННЫХ 19

2.1 Рекомендации по корректировке типов полей..... 19

2.2 Метрики разнообразия типов данных..... 22

2.3 Метрики подобия баз данных по составу типов полей 27

3 ОЦЕНКА СЛОЖНОСТИ ФИЗИЧЕСКИХ СХЕМ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ 30

3.1 Получение количественных метрик физической схемы реляционной базы данных 31

3.2 Получение количественных метрик БД в терминах реляционной модели 32

3.3 Получение количественных метрик БД в терминах реляционной модели..... 37

3.4 Оценка сложности реляционных баз данных 39

4 ОЦЕНКА ПОЛНОТЫ И НАДЕЖНОСТИ БАЗ ДАННЫХ..... 45

4.1 Математическая модель оценки полноты и надежности для двух баз данных..... 45

4.2 Математическая модель оценки полноты и надежности для трех баз данных..... 48

ЛИТЕРАТУРА 56

Приложение №1. Соглашения об именовании объектов … 57

Приложение №2. Соглашения об оформлении исходных программных кодов … 61

Электронное учебное издание

Александр Александрович **Рыбанов**

ОЦЕНКА КАЧЕСТВА БАЗ ДАННЫХ

Учебное пособие

Электронное издание сетевого распространения

Редактор Матвеева Н.И.

Темплан 2024 г. Поз. № 2.
Подписано к использованию 17.04.2024. Формат 60x84 1/16.
Гарнитура Times. Усл. печ. л. 4,0.

Волгоградский государственный технический университет.
400005, г. Волгоград, пр. Ленина, 28, корп. 1.

ВПИ (филиал) ВолгГТУ.
404121, г. Волжский, ул. Энгельса, 42а.