16+

Мамедли Р.Э.

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Учебное пособие

Министерство науки и высшего образования РФ
ФГБОУ ВО «Нижневартовский государственный университет»

Мамедли Р.Э.

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Нижневартовск
2021

ББК 32.972.34 **16+**
УДК 004.65:004.43
М 22

Печатается по решению
Ученого совета ФГБОУ ВО «Нижневартовский государственный университет»

Мамедли Р.Э.

М 22 **Системы управления базами данных:** Учебное пособие. – Нижневартовск: Изд-во Нижневартовского государственного университета, 2021. – 214 с.

ISBN 978-5-00047-585-0

В учебном пособии рассматриваются вопросы организации баз данных. Изложены принципы проектирования реляционных баз данных, нормализация отношений. Подробно рассматриваются операции реляционной алгебры, синтаксис и применение языка SQL. Даются характеристики системы управления базами данных различных поколений. Материал подготовлен на основе учебного курса, который читается автором в Нижневартовском государственном университете.

Учебное пособие ориентировано на студентов специальностей 09.03.02 «Информационные системы и технологии» и 09.03.01 «Информатика и вычислительная техника», а также студентов родственных специальностей и разработчиков баз данных.

ББК: 32.972.34
УДК: 004.65:004.43

ISBN 978-5-00047-585-0

Оглавление

Лекция 1. Основные понятия

1.1. Введение

В век информации непрерывно производятся и собираются различные данные в процессе деятельности людей. Люди и организации нуждаются в актуальных данных для решения своих повседневных задач. Собранные данные используются для получения информации. Для принятия правильных решений требуется актуальная и достоверная информация.

Для хранения и управления данными в современных компьютерных системах применяются базы данных. Базы данных участвуют почти во всех областях нашей жизни: в учебных заведениях, в управлении бизнесом или производством, в банковском деле, в решении научных, инженерных и медицинских задач. Базы данных возникли из-за необходимости управлять большими объемами данных в организованном и эффективном виде.

1.2. Области применения баз данных

Объем генерируемых данных постоянно растет. Данные собираются из разных источников: работа и учеба, социальные сети, магазины, аптеки, туристические фирмы, наши друзья и родственники предоставляют нам обильный объем данных. Предприятия тоже работают с данными. Они ведут учёт клиентов, предлагаемых услуг и продуктов, работников и денежных средств. Организации должны хранить эти и многие другие данные. Люди, занятые в управлении, нуждаются в данных для принятия важных решений.

Операторы сотовой связи, такие как Мегафон, Билайн и МТС, как известно, хранят данные десятков миллионов клиентов о миллиардах телефонных звонков, а новые данные добавляются ежесекундно. Этим компаниям нужно не только хранить огромное количество данных и управлять им, но и уметь быстро находить в них любые данные. Социальные сети, как Facebook или ВКонтакте, хранят и обрабатывают миллионы постов в день. Поисковые системы, как Google или Яндекс, обрабатывают миллионы запросов в день, а результаты дают практически мгновенно.

Как можно обработать столько данных? Как можно найти среди огромного количества данных искомое? Для решения этих задач используются базы данных. *Базы данных* – специальные структуры, которые позволяют компьютерным системам хранить, управлять и извлекать данные очень быстро. Современные информационные системы основываются на базах данных. Для специалиста в области информационных систем важно понимать, как создаются и как эффективно используются базы данных.

1.3. Данные и информация

Для правильного проектирования баз данных нужно понимать разницу между данными и информацией. *Данные* состоят из необработанных фактов. Например, предположим, что есть термометр на улице, который периодически отправляет смартфону уличную температуру,

влажность, давление и т.д. В памяти смартфона хранятся числа разного значения, следовательно, есть факты, но они не особо полезны в таком формате. Число 20 не очень информативно, но, если сказать, что «сейчас на улице температура 20°C выше нуля» – данные имеют смысл.

Информация – результат обработки данных с целью выявления ее смысла. Обработка данных может быть простой, например, вычисление среднего значения, или сложной, например, составление прогнозов или вывод с применением статистического моделирования. Чтобы раскрыть смысл информации, требуется контекст. Так, показание средней температуры 20 градусов ничего не значит, если неизвестен его контекст: в градусах по Фаренгейту или Цельсию? Это температура комнаты, воды или наружного воздуха? Информация может быть использована в качестве основы для принятия решений. Например, узнав температуру на улице, можно решить, какую одежду надеть.

Необработанные данные должны быть правильно отформатированы для хранения, обработки и представления. Например, даты могут храниться в формате григорианского календаря в базе данных, но отображаться в различных форматах, таких как «день-месяц-год» или «месяц/день/год». Ответы респондентов, как «Да/Нет», могут нуждаться в преобразовании в формат «Y/N» или «0/1» для хранения. Более сложное форматирование требуется, когда обрабатываются сложные типы данных, такие как звуки, видео или изображения.

В современном информационном веке производство точной, актуальной и своевременной информации – ключ к правильному принятию решений. В свою очередь, правильное принятие решений является ключом к успеху.

Данные являются основой информации, которая является основой *знаний,* информации и фактов о конкретной теме. Знание подразумевает знакомство, осознание и понимание информации применительно к окружающей среде. Ключевым свойством знаний является то, что «новые» знания могут быть получены из «старых».

На основе вышеперечисленного, можно подвести итоги:

- Информация производится путем обработки данных.
- Точная, актуальная и своевременная информация является ключом к принятию правильных решений.
- Правильное принятие решений является ключом к успеху.

Знания и информация требуют своевременных и точных данных. Данные должны быть правильно сгенерированы и сохранены в доступном формате. Среда хранения данных должна тщательно управляться. *Управление данными* – это дисциплина, которая фокусируется на правильном создании, хранении и извлечении данных. Управление данными является основным видом деятельности для любого бизнеса, государственного учреждения, сервисной компании или учебной организации.

База данных (БД) – это доступная, интегрированная структура, которая состоит из двух частей:

- Данные – необработанные факты.

- *Метаданные* или данные о данных, с помощью которых данные интегрируются и управляются.

Метаданные содержат характеристики данных и набор отношений, которые связывают данные. Например, метаданные хранят имя каждого элемента данных, тип значений (числовые, даты или текст). Таким образом, метаданные представляют более полную картину данных в БД. Учитывая определение метаданных, можно определить БД как «коллекцию данных с самоописанием».

1.4. Система управления базами данных

Система управления базами данных (СУБД) – это набор программ, которые управляют структурой БД и контролируют доступ к данным, хранящимся в БД.

СУБД служит посредником между пользователем и БД. Сама структура БД хранится в виде набора файлов, и единственный способ получить доступ к данным в этих файлах – через СУБД. Она представляет конечному пользователю (или прикладной программе) единое интегрированное представление данных в БД. СУБД получает запросы приложений и переводит их в сложные операции, необходимые для выполнения. Она скрывает большую часть внутренней сложности БД от прикладных программ и пользователей. Прикладная программа может быть написана программистом с использованием языка программирования высокого уровня, такого как Python, Java или C#, или она может быть создана с помощью инструментов СУБД.

Наличие СУБД между приложениями пользователя и БД дает некоторые важные преимущества. Во-первых, СУБД позволяет совместно использовать данные в БД нескольким приложениям или пользователям. Во-вторых, СУБД объединяет представления данных разных пользователей в единый универсальный репозиторий данных.

Из преимуществ использования СУБД можно перечислить следующие:

- *Улучшенный обмен данными*. СУБД помогает создать среду, в которой пользователи получают доступ к управляемым данным.
- *Улучшенная безопасность данных*. Чем больше пользователей получают доступ к данным, тем выше риск нарушения безопасности. СУБД обеспечивает соблюдение политики конфиденциальности и безопасности данных.
- *Улучшенная интеграция данных*. Более широкий доступ к хорошо управляемым данным обеспечивает интегрированное представление о деятельности организации и более четкое представление об общей картине.
- *Минимизация несогласованности данных*. *Несогласованность данных* существует, когда разные версии одних и тех же данных появляются в разных местах. Например, несогласованность данных возникает, когда отдел продаж хранит имя торгового представителя как «Саша Иванов», а отдел кадров хранит имя того же человека как «Иванов Александр Петрович». Вероятность несогласованности данных значительно снижается в правильно спроектированной БД.

- *Улучшенный доступ к данным*. СУБД позволяет быстро получать ответы на специальные запросы. С точки зрения БД, *запрос* – это особое требование, выданное СУБД для манипулирования данными, например, для чтения или обновления данных. СУБД отправляет ответ *(набор результатов запроса)* в приложение. Например, при работе с большими объемами данных о продажах, пользователям могут потребоваться ответы на вопросы:
  – Каков был объем продаж по продуктам за последние шесть месяцев?
  – Какова величина бонуса продаж для каждого из продавцов за последние три месяца?
  – У скольких из клиентов долг составляет 3 миллиона рублей и более?
- *Улучшенное принятие решений*. Улучшенный доступ к управляемым данным позволяют генерировать качественную информацию для принятия правильных решений. Качество генерируемой информации зависит от качества лежащих в основе данных. *Качество данных* – это комплексный подход к повышению точности, достоверности и актуальности данных.
- *Повышение производительности пользователя*. Доступность данных в сочетании с инструментами, которые преобразуют данные в полезную информацию, дает возможность пользователям принимать быстрые, обоснованные решения.

1.5. Разновидности БД

СУБД может использоваться для создания разных типов БД. Каждая БД хранит определенную коллекцию данных и используется для определенной цели. По мере развития технологий, для классификации БД использовались разные методы. Например, БД могут быть классифицированы по количеству поддерживаемых пользователей, по месту расположения данных, по типу хранимых данных или степени структурированности данных.

В зависимости от количества пользователей, БД классифицируются:

- *Однопользовательская БД* – поддерживает только одного пользователя одновременно. Другими словами, если пользователь А использует БД, пользователи В и С должны ждать, пока пользователь А не завершит работу. Однопользовательская БД, которая работает на персональном компьютере, называется *персональной БД*.
- *Многопользовательская БД* – поддерживает несколько пользователей одновременно. Когда многопользовательская БД поддерживает относительно небольшое количество пользователей (обычно менее 50) или определенный отдел в организации, она называется *БД рабочей группы*. Когда БД используется всей организацией и поддерживает множество пользователей (более 50, обычно несколько сотен) во многих отделах, она называется *корпоративной БД*.

Расположение также может быть использовано для классификации БД:

- *Централизованная БД* – поддерживает данные, расположенные на одном сервере.

- *Распределенная БД* – поддерживает данные, распределенные по нескольким различным сервером.

Как централизованные, так и распределенные БД требуют четко определенной инфраструктуры (аппаратное обеспечение, операционные системы, сетевые технологии и т.д.) для реализации и эксплуатации БД. Как правило, инфраструктура принадлежит и поддерживается организацией, которая создает и управляет БД. Но в последние годы популярность использования облачных БД растет.

Облачная БД – это БД, которая создается и поддерживается с использованием облачных служб данных, таких как Yandex Object Storage, Microsoft Azure или Amazon AWS. Владельцы данных не должны знать или беспокоиться о том, какое аппаратное и программное обеспечение используется для поддержки их БД.

В некоторых контекстах, таких как исследовательские среды, популярным способом классификации БД является тип данных, хранящихся в них. Используя этот критерий, БД группируются в две категории:

- *Универсальные БД* – содержат широкий спектр данных, используемых в разных областях, например, БД переписи населения, которая содержит общие демографические данные.
- *Специальные БД* – содержат данные, ориентированные на конкретные предметные области. Данные в БД этого типа используются главным образом для академических или исследовательских целей. Примером специальных БД можно показать БД географических информационных систем (ГИС), в которых хранятся геопространственные данные.

Наиболее популярный способ классификации БД сегодня основан на том, как они будут использоваться, на чувствительности ко времени получаемой от них информации:

- *Оперативная БД* – предназначена для поддержки повседневных операций компании, также известная как *БД оперативной обработки транзакций (online transaction processing – OLTP), транзакционная БД* или *производственная БД.*
- *Аналитическая БД* – ориентирована на хранение исторических данных и бизнес-показателей, используемых для принятия решений. Такой анализ обычно требует расширенную обработку данных для получения информации. Аналитические БД позволяют пользователю выполнять расширенный анализ бизнес-данных с использованием сложных инструментов.

Аналитические БД включают два основных компонента: хранилище данных и инструменты аналитической обработки:

- *Хранилище данных (ХД)* – это специализированная БД, в которой хранятся данные в формате, оптимизированном для поддержки принятия решений. ХД содержит исторические данные, полученные из оперативных БД, а также данные из других внешних источников.
- *Аналитическая обработка в реальном времени (online analytical processing – OLAP)* – это набор инструментов, которые работают вместе, чтобы обеспечить расширенную среду анализа данных для извлечения, обработки и моделирования

данных из ХД. В последнее время эта область применения БД выросла в важности и использовании, вплоть до того, что она превратилась в собственную дисциплину, бизнес-аналитику. Термин *бизнес-аналитика* описывает комплексный подход к сбору и обработке бизнес-данных с целью генерирования информации для поддержки принятия бизнес-решений.

БД также могут быть классифицированы для отражения степени структурирования данных.

- *Неструктурированные данные* – это данные, которые существуют в исходном (необработанном) состоянии, то есть в том формате, в котором они были собраны. Следовательно, неструктурированные данные существуют в формате, который не поддается обработке, дающей информацию.
- *Структурированные данные* – это результат форматирования неструктурированных данных для облегчения хранения, использования и генерации информации.
- *Полуструктурированные данные* – данные, обработанные в определенной степени. Например, если посмотреть на типичную веб-страницу, данные будут представлены в заранее подготовленном формате для передачи некоторой информации.

Указанные выше типы БД сосредоточены на хранении и управлении высокоструктурированными данными. Однако организации также используют полуструктурированные и неструктурированные данные. Существуют большое количество ценной информации, которую можно найти в электронных письмах, заметках и документах компании. Неструктурированные и полуструктурированные потребности хранения и управления данными решаются с помощью нового поколения баз данных, известных как базы данных XML. *Расширенный язык разметки (XML)* – это специальный язык, используемый для представления и обработки элементов данных в текстовом формате. БД XML поддерживает хранение и управление полуструктурированными XML-данными.

Таблица 1

Сравнение известных СУБД

БД	**Количество пользователей**		**Расположение данных**		**Использование данных**		**XML**
	Один	Много	Централ.	Распред.	Операт.	Аналит.	
MS Access	Да	Да	Да		Да		
MS SQL Server	Да	Да	Да	Да	Да	Да	Да
IMB DB2	Да	Да	Да	Да	Да	Да	Да
Oracle RDBMS	Да	Да	Да	Да	Да	Да	Да
PostgreSQL	Да	Да	Да	Да	Да	Да	Да

База данных – это компьютерная структура, в которой хранятся данные конечного пользователя и осуществляется управление ими. Одна из первых задач специалиста по базам

данных – обеспечить правильную структуру данных конечного пользователя для получения достоверной и своевременной информации. Для этого необходим хороший дизайн базы данных.

1.6. Проектирование БД

Проектирование БД – разработка структуры БД, которая будет использоваться для хранения данных и управления ими. Проектирование БД является очень важным аспектом работы с БД. Даже хорошая СУБД будет плохо работать с плохо спроектированной БД.

Данные являются одним из наиболее ценных активов организации. Данные о клиентах, сотрудниках, заказах и квитанциях имеют жизненно важное значение для существования предприятия. Отслеживание ключевых показателей роста и эффективности необходимо для стратегических и тактических планов; следовательно, данные организации не должны обрабатываться небрежно.

Современные СУБД достаточно просты в использовании, поэтому многие опытные пользователи приобретают ложное чувство уверенности о своей способности создать функциональную базу данных. Большинство пользователей могут эффективно ориентироваться в создании объектов БД, но без должного понимания структуры БД они часто создают некорректные, чрезмерно упрощенные структуры, которые мешают системе правильно хранить данные, и это приводит к неполным или ошибочным результатам.

Правильный дизайн базы данных требует от разработчика точного определения предполагаемого использования БД. При проектировании оперативной БД нужно учитывать точность и согласованность данных и скорость работы. Проектирование базы данных хранилища данных учитывает использование исторических и агрегированных данных.

Проектирование структур данных с использованием двумерных структур таблиц является процессом декомпозиции. Данные должны быть правильно разложены на составные части, каждая часть должна храниться в своей собственной таблице. Кроме того, отношения между этими таблицами должны быть внимательно рассмотрены и реализованы.

Хорошо спроектированная БД облегчает управление данными и генерирует точную и ценную информацию. Плохо спроектированная БД может стать причиной для трудно отслеживаемых ошибок, которые могут привести к неправильному принятию решений, а плохое принятие решений может привести к краху организации.

1.7. Файловые системы хранения данных

Проектирование БД, позволяющих избежать ошибок предыдущих систем, требует от разработчика понимания этих проблем и способов их устранения.

Для реализации своих задач, организация должна работать с данными. До появления компьютеров использовались бумажно-карандашные системы. Документы были организованы в папки с файлами и картотекой. Поскольку количество данных было относительно мало, ручная система выполняла свою роль в качестве хранилища данных.

Однако по мере роста организаций и усложнения требований к отчетности отслеживание данных в ручной файловой системе становилось все труднее.

Создание отчетов из файловых систем вручную было медленным и громоздким. Например, работа бухгалтера требовала недели интенсивных усилий каждый квартал, даже когда использовалась хорошо разработанная ручная система. Поэтому было необходимо создать компьютерную систему, которая бы отслеживала данные и создавала необходимые отчеты.

Первоначально компьютерные файлы были похожи на ручные документы. Когда пользователи хотели получить данные из компьютера, они отправляли запросы к *специалисту обработки данных (ОД)*. Для каждого запроса специалист ОД должен был создавать программы для извлечения данных из файла, обрабатывать и представлять их в виде печатного отчета.

Учитывая, что каждый файл в системе использовал свою собственную прикладную программу для хранения, извлечения и изменения данных, по мере разработки все большего числа компьютерных файлов появились проблемы контроля и управления данными. Критика метода файловой системы служит двум основным целям:

- Понимание недостатков файловой системы позволяет понять развитие современных БД.
- Непонимание таких проблем может привести к их дублированию в среде БД.

Файловым системам присущи следующие ограничения:

- *Длительные сроки разработки*. Самая простая задача поиска данных требует обширного программирования. В файловых системах программистам приходилось указывать, что нужно делать и как это делать. Современные БД используют язык манипулирования данными, который позволяет пользователю указывать, что должно быть сделано.
- *Трудность получения быстрых ответов*. Необходимость написания программ для создания даже самых простых отчетов делает невозможными специальные запросы. Если срочно нужна информация, ее получение на следующей неделе или в следующем месяце не будет отвечать требованиям.
- *Комплексное системное администрирование*. Системное администрирование становится более сложным, так как количество файлов в системе увеличивается. Каждый файл должен иметь свои собственные программы управления файлами, которые позволяют пользователю добавлять, изменять и удалять записи; перечислить содержимое файла; и создавать отчеты. Поскольку специальные запросы невозможны, программы создания отчетов о файлах могут быстро размножаться.
- *Отсутствие безопасности и ограниченный обмен данными*. Еще одна проблема хранилища данных файловой системы – отсутствие безопасности и ограниченный обмен данными. Обмен данными между несколькими пользователями создает много рисков для безопасности. С точки зрения создания программ управления данными и составления отчетов, функции безопасности и совместного использования данных сложно программировать и, следовательно, они часто исключаются из среды файловой системы.

- *Обширное программирование*. Любое изменение структуры файла, независимо от того, насколько оно незначительное, вызывает изменения во всех программах, которые используют данные в этом файле. Модификации могут приводить к ошибкам (багам), и дополнительное время тратится на отладку этих ошибок.

Эти ограничения, в свою очередь, приводят к проблемам структурной и информационной зависимости. Файловая система демонстрирует *структурную зависимость,* что означает, что доступ к файлу зависит от его структуры. Прикладные программы файловой системы подвержены изменениям в файловой структуре, они проявляют структурную зависимость.

Даже изменения характеристик данных, такие как изменение поля с целого на десятичное, требуют изменений во всех программах, которые обращаются к файлу. Поскольку все программы доступа к данным могут изменяться при изменении *типа данных*, означает, что файловая система демонстрирует *зависимость от данных*. И наоборот, *независимость данных* существует, когда можно изменять характеристики данных, не влияя на способность программы получать доступ к данным.

Практическая значимость зависимости данных заключается в разнице между *логическим форматом данных* (то, как человек рассматривает данные) и *физическим форматом данных* (как компьютер должен работать с данными). Любая программа, которая обращается к файлу файловой системы, должна указывать компьютеру не только, что делать, но и как это делать.

Структура файловой системы затрудняет объединение данных из нескольких источников, а отсутствие безопасности делает файловую систему уязвимой для нарушений безопасности. Организационная структура способствует хранению одних и тех же данных в разных местах. *Избыточность данных* существует, когда одни и те же данные хранятся без необходимости в разных местах.

Неконтролируемая избыточность данных создает условия для следующего:

- *Нарушения безопасности данных*. Наличие нескольких копий данных увеличивает вероятность того, что копия данных будет подвержена несанкционированному доступу.
- *Несогласованность данных*. Несогласованность данных существует, когда разные и конфликтующие версии одних и тех же данных появляются в разных местах. Отчеты будут давать противоречивые результаты, которые зависят от того, какая версия данных используется.
- *Ошибки ввода данных*. Ошибки ввода данных чаще возникают, когда сложные записи (например, 10-значные номера телефонов) сделаны в нескольких разных файлах.

В идеале, изменение значения поля должно быть сделано только в одном месте. Однако избыточность данных способствует возникновению ненормальных условий, заставляя изменения значений полей во многих разных местах. Аномалии данных классифицируется следующим образом:

- *Аномалия обновления.*
- *Аномалия добавления.*
- *Аномалия удаления.*

1.8. Системы баз данных

Проблемы, присущие файловым системам, стали причиной появления системы баз данных (БД). В отличие от файловой системы с ее множеством отдельных и несвязанных файлов, система БД состоит из логически связанных данных, хранящихся в одном хранилище логических данных. Термин «логический» означает, что хранилище данных представляется единым целым, а данные могут быть физически распределены по нескольким хранилищам и расположениям. Поскольку хранилище данных представляет собой единую логическую единицу, БД представляет серьезное изменение в способе хранения, доступа и управления данными.

Система базы данных – определяет и регулирует сбор, хранение, управление и использование данных в среде БД. С точки зрения общего управления, система баз данных состоит из пяти основных частей, показанных на рисунке 1: аппаратное обеспечение, программное обеспечение, люди, процедуры и данные.

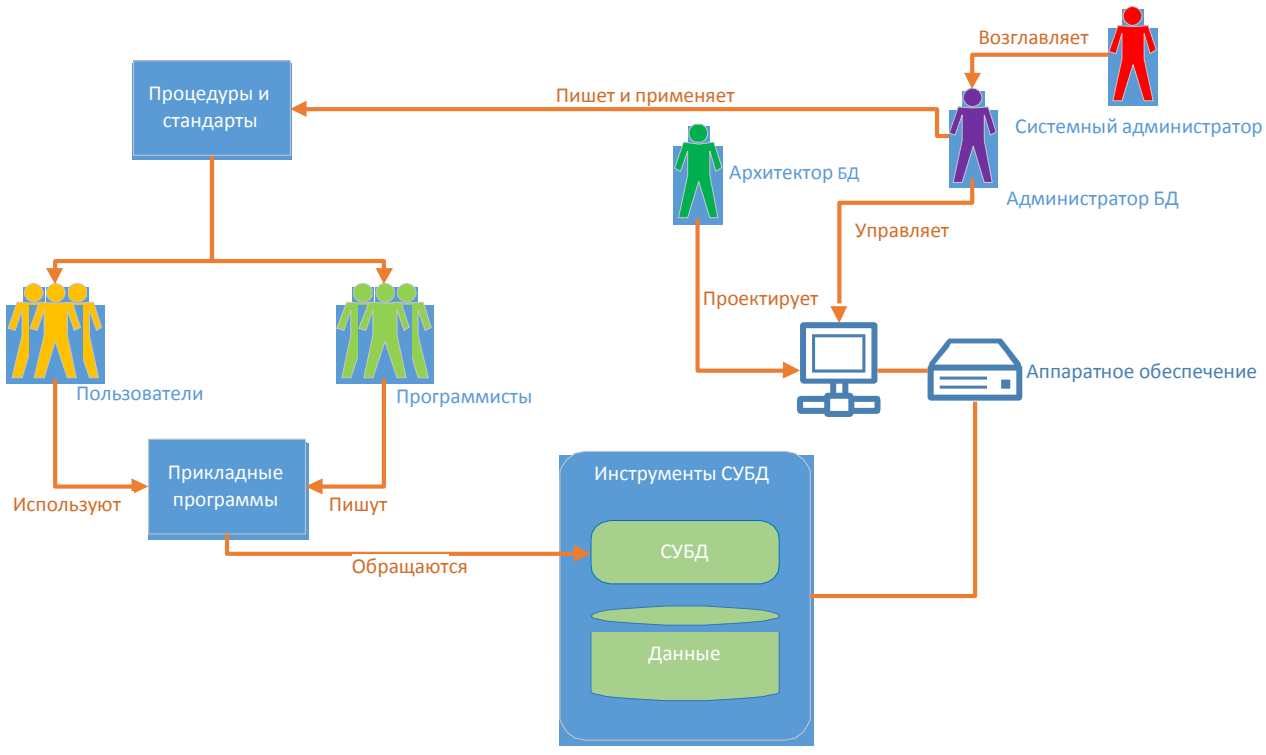


Рис. 1. Окружение системы базы данных

Пять компонентов, показанных на рисунке 1:

- *Аппаратное обеспечение*. Аппаратное обеспечение относится ко всем физическим устройствам, включая компьютеры (серверы, ПК, планшеты, рабочие станции и т.д.), устройства хранения данных, принтеры, сетевые устройства и другие устройства (банкоматы, ID-ридеры, и т.д.).

- *Программное обеспечение*. Для полноценного функционирования системы БД необходимы три типа программного обеспечения: операционная система, СУБД, прикладные программы.
  - *Операционная система* управляет всеми аппаратными компонентами и позволяет запускать все остальное программное обеспечение на компьютерах. Примерами являются Microsoft Windows, Linux, Mac OS, UNIX и IBM z/OS.
  - *СУБД* управляет базой данных в системе БД. Некоторыми примерами СУБД являются Microsoft SQL Server, Oracle RDBMS, Oracle MySQL, IBM DB2 и PostgreSQL.
  - *Прикладные программы* используются для доступа к данным в СУБД и управления ими. Прикладные программы чаще всего используются для создания отчетов, таблиц и другой информации, облегчающей принятие решений.
- *Люди*. Этот компонент включает в себя всех пользователей системы БД. На основе рабочих функций в системе БД могут быть определены пять типов пользователей: системные администраторы, администраторы баз данных, архитекторы БД, системные аналитики/программисты и другие пользователи. Каждый тип пользователя выполняет как уникальные, так и дополнительные функции.
  - *Системные администраторы* контролируют общие операции системы БД.
  - *Администраторы базы данных* (database administrator – dba) управляют СУБД и обеспечивают правильное функционирование базы данных.
  - *Архитекторы баз данных* проектируют структуру БД. Если архитектура плохая, невозможно создать полезную среду БД. Поскольку организации стремятся оптимизировать свои ресурсы данных, описание работы архитектора базы данных было расширено, чтобы охватить новые измерения и растущие обязанности.
  - *Системные аналитики и программисты* разрабатывают и внедряют прикладные программы. Они проектируют и создают экраны ввода данных, отчеты и процедуры, с помощью которых пользователи получают доступ к данным БД и манипулируют ими.
  - *Пользователи* – это люди, которые используют прикладные программы для повседневной работы организации. Например, продавцы, руководители, менеджеры и директора классифицируются как пользователи. Пользователи высокого ранга используют информацию, полученную из БД, для принятия тактических и стратегических решений.
- *Процедуры* – это инструкции и правила, которые определяют структуру и использование системы БД. Процедуры являются критическим компонентом системы. Процедуры играют важную роль в компании, поскольку они обеспечивают соблюдение стандартов, по которым ведется бизнес внутри

организации и с клиентами. Процедуры также помогают гарантировать, что компании имеют организованный способ мониторинга и аудита данных, которые поступают в БД, и информации, полученной из этих данных.

- *Данные* – охватывают сбор фактов, хранящихся в БД. Поскольку данные являются исходным материалом, из которого генерируется информация, определение того, какие данные вводить в БД и как организовать эти данные, является важной частью работы архитектора БД.

Система БД добавляет новое измерение в структуру управления организации. Сложность этой управленческой структуры зависит от размера организации, ее функций и корпоративной культуры. Следовательно, системы БД могут создаваться и управляться на разных уровнях сложности и с разным соблюдением точных стандартов.

В дополнение к различным уровням сложности системы БД менеджеры также должны учитывать еще один важный факт: решения должны быть экономически, тактически и стратегически эффективными. Наконец, технология, которая уже используется, может повлиять на выбор системы БД.

1.9. Преимущества и недостатки СУБД

СУБД выполняет несколько важных функций, которые гарантируют целостность и согласованность данных в БД. Большинство из этих функций прозрачны для пользователей, и они могут быть реализованы только за счет использования СУБД. Эти задачи включают управление словарем данных, управление хранением данных, преобразование и представление данных, управление безопасностью, управление многопользовательским доступом, управление резервным копированием и восстановлением, управление целостностью данных, языки доступа к базе данных, а также интерфейсы связи с БД. Каждая из этих функций объясняется следующим образом:

- *Управление словарем данных*. СУБД хранит определения элементов данных и их отношений (*метаданных*) в словаре данных. В свою очередь, все программы, которые обращаются к данным в БД, работают через СУБД. СУБД использует *словарь данных* для поиска необходимых структур и связей компонентов данных, что избавляет от необходимости кодировать такие сложные отношения в каждой программе. СУБД обеспечивает абстрагирование данных и устраняет структурную зависимость, и зависимость от данных из системы. Например, на рисунке 2 показано, как Microsoft SQL Server представляет определение данных для таблицы «Страны».
- *Управление хранением данных*. СУБД создает и управляет сложными структурами, необходимыми для хранения данных, что избавляет разработчика от сложной задачи определения и программирования физических характеристик данных. Хотя пользователь видит БД как единое хранилище данных, СУБД фактически хранит ее в нескольких физических файлах данных. Такие файлы данных могут даже храниться на разных носителях. СУБД может выполнять запросы к БД параллельно.

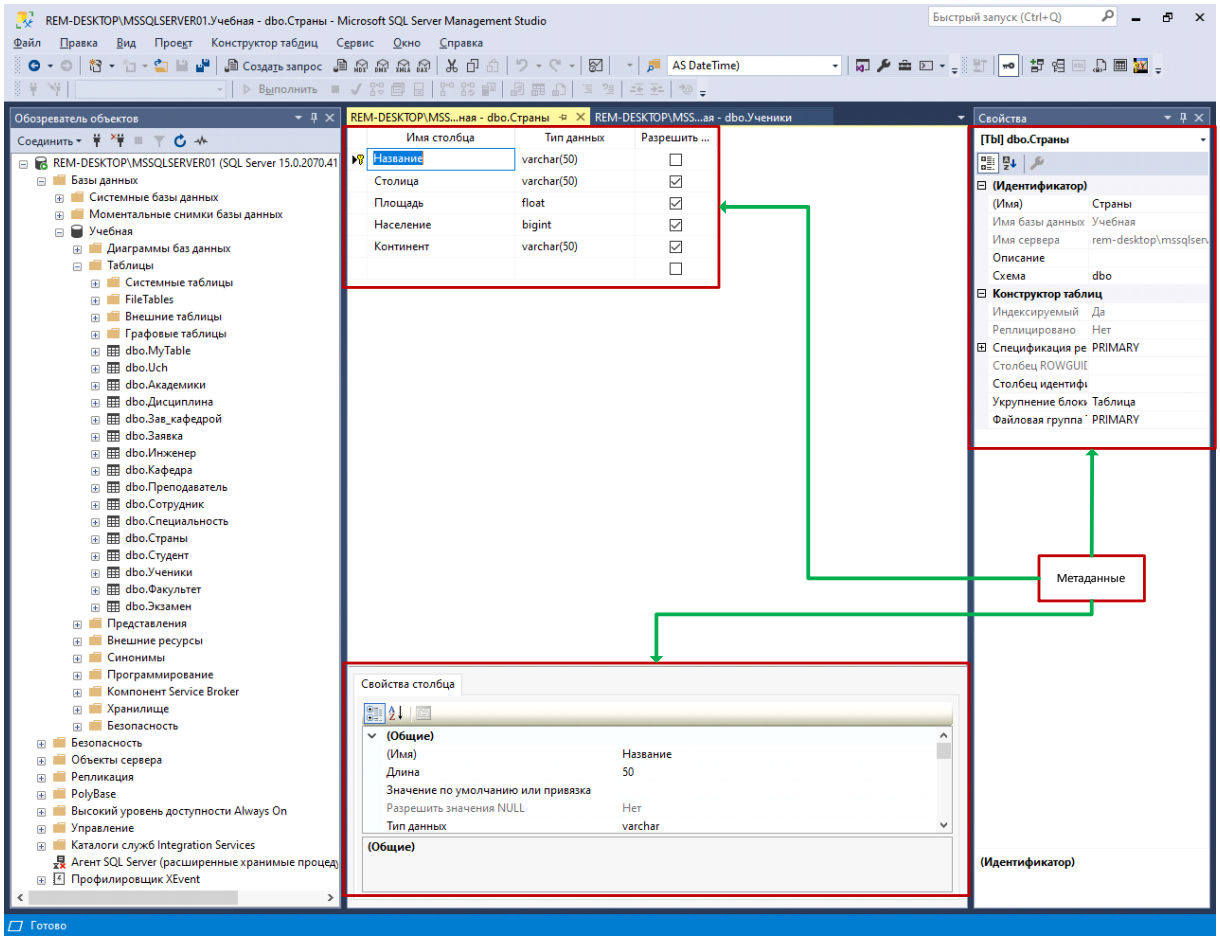


Рис. 2. Метаданные в MS SQL Server

- *Преобразование и представление данных*. СУБД преобразует введенные данные в соответствии с требуемыми структурами данных, и таким образом, избавляет разработчика от рутинной работы по различению логического формата данных и физического формата данных. То есть СУБД форматирует физически извлеченные данные, чтобы они соответствовали логическим ожиданиям пользователя.
- *Управление безопасностью*. СУБД создает систему безопасности, которая обеспечивает безопасность пользователей и конфиденциальность данных. Правила безопасности определяют, какие пользователи к каким элементам данных могут обращаться и какие операции с данными (чтение, добавление, удаление или изменение) могут выполнять. Это особенно важно в многопользовательских системах БД. Все пользователи БД могут быть аутентифицированы в СУБД с помощью имени пользователя и пароля или с помощью биометрической аутентификации, такой как сканирование отпечатков пальцев. СУБД использует эту информацию для назначения прав доступа различным компонентам БД, таким как запросы и отчеты.
- *Многопользовательский контроль доступа*. Чтобы обеспечить целостность и согласованность данных, СУБД использует сложные алгоритмы, обеспечивающие одновременный доступ нескольких пользователей к БД, не нарушая ее целостности.

- *Резервное копирование и управление восстановлением*. Современные системы СУБД предоставляют специальные утилиты, которые позволяют администраторам БД выполнять процедуры резервного копирования и восстановления.
- *Управление целостностью данных*. СУБД продвигает и обеспечивает соблюдение правил целостности, уменьшает избыточность данных и увеличивает согласованность данных. Связи данных, хранящиеся в словаре данных, используются для обеспечения целостности данных.
- *Языки доступа к БД*. СУБД обеспечивает доступ к данным через язык запросов. *Язык запросов* – это непроцедурный язык, который позволяет пользователю указать, что должно быть сделано, не указав, как. Язык структурированных запросов (structured query language – *SQL*) является де-факто языком запросов и стандартом доступа к данным, поддерживаемым большинством поставщиков СУБД.
- *Интерфейсы связи с базой данных*. Современные СУБД принимают запросы пользователей через несколько разных сетевых сред. Например, СУБД может предоставлять доступ к БД через Интернет с помощью веб-браузеров.

Система базы данных позволяет решать гораздо более сложные задачи использования ресурсов данных, чем файловые системы хранилища данных. Но и у системы БД есть недостатки:

- *Увеличение расходов*. Системы БД требуют сложного аппаратного и программного обеспечения и высококвалифицированного персонала. Затраты на обучение, лицензирование и соответствие нормативным требованиям часто игнорируются при внедрении систем баз данных.
- *Сложность управления*. Системы БД содержат важные данные компании, доступ к которым осуществляется из разных точек, требования к безопасности должны тщательно соблюдаться.
- *Поддержание актуальности*. Чтобы увеличить эффективность системы БД, она должна быть в актуальном состоянии. Нужно своевременно обновлять и применять последние исправления для всех компонентов.
- *Зависимость от поставщика*. Учитывая размеры инвестиции в технологии и обучение персонала, предприятия неохотно меняют поставщиков БД. В результате клиенты ограничены в выборе компонентов системы БД.
- *Частые циклы обновления/замены*. Поставщики СУБД часто обновляют свои продукты, добавляя новые функциональные возможности. Новые версии могут требовать обновления оборудования и обучения персонала.

1.10. Итоги

- Данные состоят из необработанных фактов. Информация является результатом обработки данных. Точная, актуальная и своевременная информация является ключом к правильному принятию решений.
- Данные обычно хранятся в БД. Для создания БД и управления ее содержимым необходима система управления базами данных (СУБД). СУБД служит посредником между пользователем и БД. БД содержит данные и «данные о данных» – метаданные.
- Хорошо спроектированная БД облегчает управление данными и генерирует точную и ценную информацию. Плохо спроектированная БД может привести к плохому принятию решений, а плохое принятие решений может привести к краху организации.
- БД могут быть классифицированы в соответствии с количеством поддерживаемых пользователей, расположением и типом данных, специальностью и степенью структурированности данных.
- БД развивались из ручной, а затем компьютеризированной файловой системы. В файловой системе данные хранятся в независимых файлах, каждый из которых требует своих собственных программ управления данными. Хотя этот метод управления данными в значительной степени устарел, понимание его характеристик облегчает понимание структуры БД.
- Файловая система требует обширного программирования, вносить изменения в существующие структуры сложно, соблюдение безопасности трудно. Независимые файлы содержат избыточные данные, что приводит к проблемам структурной и информационной зависимости.
- СУБД были разработаны для устранения недостатков, присущих файловой системе. Она представляет БД пользователю в виде единого хранилища данных. Кроме того, СУБД обеспечивает целостность данных, устраняет избыточность и обеспечивает безопасность данных.

Лекция 2. Модели данных

2.1. Введение

Моделирование данных – это первый шаг в процессе проектирования БД, служащий мостом между объектами реального мира и компьютерной системой.

Архитекторы, программисты и пользователи видят данные по-разному. Разные представления об одних и тех же данных могут привести к проектам баз данных, которые не отражают фактическую деятельность организации, не позволяют удовлетворить потребности конечных пользователей и требования к эффективности данных. Чтобы избежать таких проблем, архитекторы БД должны получить точное описание свойства данных. Общение между архитекторами БД, программистами и пользователями должно быть частым и понятным. Моделирование данных проясняет такое взаимодействие, сводя сложности проектирования базы данных к более понятным абстракциям, которые определяют сущности, отношения и преобразования данных.

2.2. Модели данных и моделирование

При проектировании БД основное внимание уделяется тому, как структура будет использоваться для хранения и управления данными. *Моделирование данных* – первый шаг в проектировании базы данных, относится к процессу создания конкретной модели данных для определенной области. *Модель данных* – абстракция более сложного объекта или события реального мира. В среде БД модель данных представляет структуры данных и их характеристики, связи, ограничения, преобразования и другие конструкции с целью поддержки конкретной области.

Моделирование данных – это итеративный процесс. При правильном выполнении окончательная модель данных представляет собой проект со всеми инструкциями по созданию базы данных, которая будет отвечать всем требованиям пользователя. Проект БД носит содержит как текстовые описания на ясном форме, так и понятные, полезные диаграммы, изображающие основные элементы данных.

Традиционно архитекторы БД, чтобы разработать хорошую модель данных, полагались на здравый смысл. К сожалению, здравый смысл развивается после долгих проб и ошибок. Например, если каждый студент в классе должен создать модель данных для библиотеки, очень вероятно, что каждый из них придумает свою модель. Какая из них будет правильной? Простой ответ: «Та, которая отвечает всем требованиям конечного пользователя», и может быть более одного правильного решения! К счастью, разработчики баз данных используют существующие конструкции для моделирования данных и мощные инструменты проектирования баз данных, которые значительно уменьшают вероятность ошибок при моделировании баз данных.

Модели данных могут облегчить взаимодействие между архитектором, программистом приложений и пользователем. Хорошо разработанная модель данных может даже способствовать лучшему пониманию организации, для которой разрабатывается проект БД.

Важность моделирования данных невозможно переоценить. Данные представляют собой основную информацию, используемую системой. Приложения создаются для управления данными и помогают преобразовывать данные в информацию.

Учитывая расположение данных, форматирование и конкретных требованиях к отчетности, разработчики приложений имеют свои представления данных. Прикладные программисты переводят политику и процедуры компании из разных источников в соответствующие интерфейсы, отчеты и экраны запросов.

Модель данных является абстракцией, невозможно извлечь необходимые данные из модели данных.

Основными элементами модели данных являются сущности, атрибуты, связи (отношения) и ограничения. *Сущность* – это человек, место, вещь или событие, о которых будут собираться и храниться данные. Сущность представляет конкретный тип объекта в реальном мире, каждый экземпляр сущности уникален. Сущности могут быть физическими объектами, такими как клиенты или продукты, но также могут быть абстракциями, такими как маршруты полетов или музыкальные концерты.

Атрибут – является свойством объекта. Например, сущность «Студент» будет описываться такими атрибутами, как фамилия, имя, номер зачетной книжки, курс и группа.

Связь – описывает отношение между сущностями. Например, существует связь между студентами и кураторами, которую можно описать следующим образом: куратор может курировать много студентов, и каждый студент может курироваться одним куратором. Модели данных используют три типа связи: *один-ко-многим, многие-ко-многим и один-к-одному*. Архитекторы БД используют сокращенные обозначения 1:M или 1..*, M:N или *..* и 1:1 или 1..1 соответственно. Вместо M:N можно использовать «M:M». Следующие примеры иллюстрируют различия между этими тремя типами:

- *Связь один-ко-многим (1:M или 1..*)*. В стране находится много городов, но каждый город находится только в одной стране. Таким образом, страна («один») связана с городами («многие»). Поэтому архитекторы БД помечают связь «в СТРАНЕ находится ГОРОД» как 1:M. Имена сущностей часто пишутся с заглавной буквы, чтобы их легко идентифицировать. Аналогично, куратор («один») может курировать много студентов, но каждый студент («много») курируется только одним куратором. Отношение «КУРАТОР курирует СТУДЕНТА» также будет помечено как 1:M.
- *Связь много-ко-многим (M:N или *..*)*. Студент может изучать много дисциплин, и каждая дисциплина может быть изучена многими студентами. Архитекторы БД обозначают связь «СТУДЕНТ изучает ДИСЦИПЛИНУ» как M:N.
- *Связь один-к-одному (1:1 или 1..1)*. В университете каждый факультет управляется одним деканом. В свою очередь, каждый декан, который является сотрудником,

управляет только одним факультетом. Следовательно, отношение «СОТРУДНИК управляет ФАКУЛЬТЕТОМ» обозначено как 1:1.

Предыдущее обсуждение определило каждая связь в обоих направлениях; то есть отношения являются двунаправленными:

- Один КУРАТОР может курировать много СТУДЕНТОВ.
- Каждый из множества СТУДЕНТОВ курируется только одним КУРАТОРОМ.

Ограничение – это лимиты на данные. Ограничения важны, потому что они помогают обеспечить целостность данных. Ограничения обычно выражаются в виде правил:

- Заработная плата работника должна иметь значения от 6000 до 350000 рублей.
- Средний балл студента должен составлять от 2.00 до 5.00.
- В каждом классе должен быть один и только один учитель.

Для правильного определения сущностей, атрибутов, отношения и ограничения, нужно точное определение бизнес-правил для области, которую моделируют.

Для определения сущностей, атрибутов и отношений, которые будут использоваться для построения модели данных, архитекторы БД должны изучать типы существующих данных, как и когда они используются. Но такие данные и информация сами по себе не дают необходимого понимания всего процесса. С точки зрения БД, сбор данных становится значимым только тогда, когда он отражает правильно определенные бизнес-правила. *Бизнес-правило* – это краткое, точное и однозначное описание политики, процедуры или стандартов в конкретной организации. Бизнес-правила должны быть оформлены в письменном виде. Правильно написанные бизнес-правила используются для определения сущностей, атрибутов, связей и ограничений.

Основными источниками бизнес-правил являются руководители предприятий, политики, заведующие отделов и письменная документация, такая как процедуры, стандарты и руководства по эксплуатации. Самым быстрым методом для определения бизнес-правил, являются прямые интервью с пользователями. К сожалению, из-за различий в восприятии – пользователи иногда являются менее надежным источником. Например, электрик может полагать, что любой электрик может ремонтировать высоковольтную линию, когда на самом деле такую задачу могут выполнять только электрики с разрешением. Такое различие может показаться тривиальным, но оно может иметь серьезные последствия. Хотя пользователи вносят решающий вклад в разработку бизнес-правил, стоит проверить восприятие пользователя. Иногда собеседования с несколькими людьми, выполняющими одну и ту же работу, дают совершенно разные представления о свойствах работы. Задача архитектора БД – согласовать различия и проверить результаты сверки, чтобы убедиться, что бизнес-правила являются правильными и точными.

Процесс идентификации и документирования бизнес-правил важен для проектирования базы данных по нескольким причинам:

- Помогает стандартизировать представление о данных.
- Позволяет понять природу, роль и объем данных.

- Позволяет понимать бизнес-процессы.
- Позволяет разработать соответствующие правила и ограничения, участия в связях и создать точную модель данных.

Бизнес-правила устанавливают основу для правильной идентификации сущностей, атрибутов, отношений и ограничений. Как правило, для определения сущностей используется существительное, а связи между сущностями определяются с помощью глагола. Например, бизнес-правило «куратор может курировать много студентов» содержит два существительных (куратор и студент) и глагол (курировать), который связывает существительные. Из этого бизнес-правила можно сделать следующие выводы:

- Куратор и студент являются объектами, представляющими интерес, и должны быть представлены их соответствующими сущностями.
- Существует связь между куратором и студентом.

Чтобы правильно определить тип связи, нужно учитывать, что связь является двунаправленной. Например, бизнес-правило «куратор может курировать много студентов» дополняется бизнес-правилом «студент курируется только одним куратором». В этом случае отношение *один-ко-многим* (*1:M).* Куратор – сторона «1», а студент – сторона «много».

Чтобы правильно определить тип связи, нужно задать два вопроса:

- Сколько экземпляров B связано с одним экземпляром A?
- Сколько экземпляров A связано с одним экземпляром B?

Например, можно оценить отношения между учеником и классом, задав два вопроса:

- По сколько дисциплин может изучать один студент? Ответ: много дисциплин.
- Сколько студентов могут изучать одну дисциплину? Ответ: много студентов.

Следовательно, отношения между студентом и дисциплиной *многие-ко-многим (M:N).*

Имена сущностей должны описывать объекты из реального мира и использовать терминологию, знакомую пользователям. Атрибуты также должны описывать данные, представленные ими. Хорошей практикой также является добавление префикса имени атрибута к имени или сокращению объекта, в котором он встречается. Например, в сущности СТУДЕНТ средний бал может называться СТ_СРЕДНИЙ_БАЛ. СТ указывает, что атрибут является свойством для сущности СТУДЕНТ, а СРЕДНИЙ_БАЛ облегчает распознавание данных, которые будут содержаться в атрибуте. Использование соглашения об именах улучшит способность модели данных облегчить взаимодействие между разработчиком и пользователем. Правильное соглашение об именах имеет большое значение для самодокументирования.

2.3. Развитие моделей данных

Стремление к лучшему управлению данными привело к появлению нескольких моделей, которые пытаются устранить критические недостатки предыдущей модели. В таблице 2 прослеживается эволюция основных моделей данных.

Таблица 2

Эволюция основных моделей данных

Поколение	Время	Модель данных	Примеры	Комментарии
Первое	1960–1970-е годы	Файловая система	VMS / VSAM	Используется в основном в системах мэйнфреймов IBM. Управляемые записи, а не отношения.
Второе	1970-е годы	Иерархический и сетевой	IMS ADABAS IDS-II	Ранние системы баз данных. Навигационный доступ.
Третье	Середина 1970-х годов	Реляционный	DB2 Oracle MS SQL Server MySQL	Концептуальная простота. Моделирование сущностных отношений (ER) и поддержка реляционного моделирования данных.
Четвертое	Середина 1980-х годов	Объектно-ориентированный. Объектно/ реляционный (O/R)	Versant Objectivity/DB DB2 UDB Oracle 12c	Объектная/реляционная поддержка типов данных объектов. Поддержка Star Schema для хранилищ данных. Появляются веб-БД.
Пятое	Середина 1990-х годов	Гибридная XML СУБД	dbXML Tamino DB2 UDB Oracle 12c MS SQL Server	Поддержка неструктурированных данных. Модель O/R поддерживает XML-документы. Гибридная СУБД добавляет интерфейс объекта в реляционные базы данных. Поддержка больших баз данных (размер в терабайтах).
Новые модели: NoSQL	С 2000-х годов, до нашего времени	Хранилище ключей-значений. Хранилище столбцов	SimpleDB (Amazon) BigTable (Google) Кассандра (Apache) MongoDB Риак	Распределенная, легко масштабируемая. Высокая производительность, отказоустойчивость. Очень большое хранилище (петабайты). Подходит для разреженных данных. Собственный интерфейс прикладного программирования (API).

Иерархическая модель была разработана в 1960-х годах для управления большими объемами данных для сложных производственных проектов. Основная логическая структура модели представлена перевернутым деревом. Иерархическая структура содержит уровни или сегменты. Сегмент является эквивалентом типа записи файловой системы. Внутри иерархии

более высокий уровень воспринимается как родительский сегмент, а непосредственно под ним, дочерний. Иерархическая модель отображает набор отношений «один-ко-многим» (1:M) между родительским и его дочерними сегментами. У каждого родителя может быть много детей, но у каждого ребенка только один родитель.

Сетевая модель была создана для более эффективного представления сложных связей данных, чем иерархическая модель. В сетевой модели пользователь воспринимает сетевую базу данных как набор записей в отношениях 1:M. Однако, в отличие от иерархической модели, сетевая модель позволяет записи иметь более одного родителя. Хотя модель сетевой базы данных в настоящее время обычно не используется, определения стандартных концепций базы данных, появившиеся в сетевой модели, все еще используются современными моделями данных:

- *Схема* – это концептуальный проект всей БД.
- *Подсхема* – часть БД, доступная прикладным программам.
- *Язык манипулирования данными (DML)* – среда, в которой можно управлять данными.
- *Язык определения данных схемы (DDL)* – среда, в которой можно управлять метаданными.

Сетевая модель не смогла отвечать на растущие информационные потребности, так как любое структурное изменение могло бы привести к хаосу во всех прикладных программах. Из-за недостатков иерархической и сетевой моделей в 1980-х годах они были заменены реляционной моделью данных.

Реляционная модель была введена в 1970 г. Э.Ф.Коддом из IBM в его работе «Реляционная модель данных для больших общих банков данных» (Communications of ACM, июнь 1970 г., стр. 377–387). Реляционная модель представляет собой большой прорыв как для пользователей, так и для архитекторов. Его концептуальная простота заложила основу для настоящей революции в области баз данных.

Основой реляционной модели является математическая концепция, известная как отношение. Чтобы избежать сложности абстрактной математической теории, можно представить *отношение* (иногда называемое *таблицей*) как двумерную структуру, состоящую из пересекающихся строк и столбцов. Каждый ряд в отношении называется *кортежем.* Каждый столбец представляет *атрибут*. Реляционная модель также описывает точный набор конструкций манипулирования данными, основанных на математических концепциях.

В 1970 году работа Кодда считалась гениальной, но непрактичной. Компьютерам в то время не хватало возможностей для реализации реляционной модели. К счастью, мощность компьютеров и эффективность операционной системы выросла в геометрической прогрессии, а стоимость компьютеров уменьшалась. Сегодня даже ПК может запускать сложные программы реляционных баз данных, такие как Oracle, DB2, Microsoft SQL Server, MySQL и другие реляционные программы.

Модель реляционных данных реализуется с помощью очень сложной *системы управления реляционными базами данных (RDBMS – СУРДБ).* СУРБД выполняет те же

основные функции, которые предоставляются иерархическими и сетевыми системами СУБД, в дополнение к множеству других функций, облегчающих понимание и реализацию реляционной модели данных.

Наиболее важным преимуществом СУБД является ее способность скрывать сложности реляционной модели от пользователя. СУБД представляет реляционную базу данных как набор таблиц, в которых хранятся данные. Пользователь может манипулировать и запрашивать данные способом, который кажется интуитивно понятным и логичным.

Таблицы связаны друг с другом через общий атрибут (значение в столбце). Тип отношения (1:1, 1:M или M:N) часто отображается в реляционной схеме. *Реляционная диаграмма* – это представление сущностей реляционной базы данных, атрибутов внутри этих сущностей и отношений между этими сущностями.

Реляционная таблица хранит коллекцию связанных сущностей. В этом отношении таблица реляционной базы данных напоминает файл, но между таблицей и файлом есть существенное различие: таблица дает полные данные и структурную независимость, потому что это чисто логическая структура. То, как данные физически хранятся в базе данных, не имеет значения для пользователя или разработчика.

Еще одной причиной доминирования реляционной модели данных является ее мощный и гибкий язык запросов.

С точки зрения конечного пользователя любое приложение реляционной базы данных на основе SQL состоит из трех частей: пользовательский интерфейс, набор таблиц, хранящихся в базе данных, и «движок» SQL. Каждая из этих частей объясняется следующим образом:

- *Интерфейс конечного пользователя*. Интерфейс, автоматически генерируя код SQL, позволяет пользователю взаимодействовать с данными.
- *Набор таблиц, хранящихся в базе данных*. В реляционной БД все данные хранятся в таблицах. Каждая таблица независима. Строки в разных таблицах связаны общими значениями в общих атрибутах.
- *SQL-движок*. Механизм SQL является частью программного обеспечения СУБД и выполняет запросы данных. Пользователь использует SQL для создания структур таблиц и для доступа к данным и обслуживания таблиц.

Быстро растущие требования к транзакциям и информации создали потребность в более сложных структурах реализации базы данных и необходимость эффективных инструментов проектирования баз данных.

Сложные проектные работы требуют концептуальной простоты для получения успешных результатов. Разработчики баз данных предпочитают использовать графический инструмент, в котором изображаются объекты и их отношения. Широко принятым стандартом для моделирования данных является *модель сущность-связь (entity-relationship – ER),* или *ERM.*

Питер Чен (Peter Chen) впервые представил модель данных ER в 1976 году; графическое представление сущностей и их взаимосвязей в структуре базы данных быстро

стало популярным, поскольку оно дополняло понятия реляционной модели данных. Модели ER обычно представлены в *диаграмме взаимосвязей объектов (ERD)*, которая использует графические представления для моделирования компонентов базы данных.

Модель ER основана на следующих компонентах:

- *Сущность* – представлена в ERD прямоугольником. Имя сущности – существительное, написано в центре прямоугольника. Имя сущности обычно пишется заглавными буквами в именительном падеже и в единственном числе: СТУДЕНТ, а не СТУДЕНТЫ, и СОТРУДНИК, а не СОТРУДНИКИ. Обычно при применении ERD к реляционной модели сущность сопоставляется с таблицей. Каждая строка в реляционной таблице называется *экземпляром сущности* в модели ER.
- *Атрибуты*. Каждая сущность состоит из набора атрибутов, которые описывают конкретные свойства сущности. Например, сущность СОТРУДНИК будет иметь такие атрибуты, как страховой номер, фамилия и имя.
- *Связь* – описывает отношение между данными. Большинство связей описывают ассоциации между двумя объектами. В реляционной модели используются три типа связи: *один-ко-многим (1:M), много-ко-многим (M:N)* и *один-к-одному (1:1)*. Название отношения обычно является активным или пассивным глаголом. Например, КУРАТОР курирует много СТУДЕНТОВ, СТУДЕНТ изучает много ДИСЦИПЛИН, а ПРЕПОДАВАТЕЛЬ управляет КАФЕДРОЙ.

На рисунке 3 показаны различные типы отношений с использованием трех нотаций ER: исходная *нотация Чена*, *нотация Crow's Foot* и более новая *нотация UML Class Diagram*, которая является частью языка унифицированного моделирования.

Рис. 3. Нотации ER диаграмм

Левая сторона диаграммы ER показывает *нотацию Чена,* основанную на историческом документе Питера Чена. В этой нотации связи пишутся рядом с каждой сущностью. Связи представлены ромбом, соединенным с сущностями с помощью линий. Имя связи написано внутри ромба.

Середина рисунка иллюстрирует *нотацию Crow's Foot* (вороньи ноги). Название Crow's Foot получено из трезубого символа, используемого для обозначения «многих». Имя связи написано над линией.

В правой части рисунка показана *нотация UML Class Diagram*. Обратите внимание, что связь представлена с символами (1..1, 1..*). Кроме того, в нотации UML Class Diagram используются имена по обе стороны отношения.

На рисунке 3 сущности и связи показаны в горизонтальном формате, но они также могут быть вертикальными. Местоположение объекта и порядок его представления не имеют значения.

Простота ER-модели делает ее доминирующим инструментом моделирования и проектирования баз данных. Тем не менее, поиск лучших инструментов для моделирования данных продолжается, так как СУБД продолжает развиваться.

Сложные проблемы производства продемонстрировали необходимость в модели данных, которая более точно представляла бы объекты реального мира. В *объектно-ориентированной модели данных (ООМД)* и данные, и их связи содержатся в единой структуре, известной как *объект*. В свою очередь, ООМД является основой для *объектно-ориентированной системы управления базами данных (ООСУБД).*

ООМД отражает совершенно другой способ определения и использования сущностей. Как и сущность реляционной модели, объект описывается его фактическим содержанием. Но, в отличие от сущности, объект включает в себя информацию об отношениях между фактами внутри объекта, а также информацию о его отношениях с другими объектами. *ООМД* называется *моделью семантических данных,* потому что семантика указывает на значение.

ООМД основана на следующих компонентах:

- Объект – это абстракция сущности реального мира. Объект может считаться эквивалентным сущности модели ER.
- Атрибуты описывают свойства объекта. Например, объект СОТРУДНИК включает в себя атрибуты Имя, Фамилия и Дата рождения.
- Объекты, которые имеют сходные свойства, сгруппированы в классы. *Класс* – это набор похожих объектов с общей структурой (атрибутами) и поведением (методами). Класс отличается от набора сущностей тем, что он содержит процедуры, известные как методы. *Метод* класса представляет собой реальное действие, такое как поиск, изменение имени или печать адреса. Другими словами, методы являются эквивалентом процедур в традиционных языках программирования.
- Классы организованы в иерархию классов. *Иерархия классов* напоминает перевернутое дерево, в котором каждый класс имеет только одного родителя. Например, класс ПРЕПОДАВАТЕЛЬ и класс АССИСТЕНТ совместно используют родительский класс СОТРУДНИК.
- *Наследование* – это способность объекта в иерархии классов наследовать атрибуты и методы родительского класса. Например, два класса, ПРЕПОДАВАТЕЛЬ и

АССИСТЕНТ могут быть созданы как подклассы класса СОТРУДНИК. В этом случае ПРЕПОДАВАТЕЛЬ и АССИСТЕНТ будут наследовать все атрибуты и методы от класса СОТРУДНИК.

- Объектно-ориентированные модели данных обычно изображаются с использованием диаграмм классов *Unified Modeling Language (UML).* UML – это язык, основанный на концепции ОО, который описывает набор диаграмм и символов, которые можно использовать для графического моделирования системы. *Диаграммы классов UML* используются для представления данных и их взаимосвязей в языке моделирования UML.

Столкнувшись с необходимостью поддержки более сложных представлений данных, основные поставщики реляционной модели предлагают *расширенную модель реляционных данных (РМРД).* РМРД – поддерживают такие функции ОО, как объекты, расширяемые типы данных на основе классов и наследования. СУБД на основе РМРД часто описывается как *система управления объектными/реляционными базами данных (СУБД О/Р).*

Большинство современных реляционных баз данных можно классифицировать как объектные/реляционные. Успех СУБД О/Р можно объяснить концептуальной простотой модели, целостностью данных, простым в использовании языком запросов, высокой производительностью, высокой доступностью, безопасностью, масштабируемостью и расширяемостью. СУБД ОО пользуется популярностью на специализированных областях, таких как *Системы Автоматизированного Проектирование* (САПР – CAD/CAM), географические информационные системы (ГИС), телекоммуникации и мультимедиа, которые требуют поддержки более сложных объектов.

Использование сложных объектов получило импульс благодаря Интернет. В этой среде *расширенный язык разметки (XML)* стал стандартом для эффективного обмена структурированными, полуструктурированными и неструктурированными данными. В СУБД О/Р добавлена поддержка документов на основе XML в их реляционной структуре данных.

Хотя реляционные и объектно-реляционные базы данных отвечают большинству текущих потребностей в обработке данных, появилось новое поколение баз данных для решения некоторых весьма специфических проблем, с которыми сталкиваются некоторые организации эпохи Интернета.

Веб-данные в форме истории покупок, предпочтений клиентов, моделей поведения, данные социальных сетей, мобильные технологии, такие как смартфоны и планшеты, датчики GPS, системы RFID, датчики погоды, биомедицинские устройства, а также другие устройства, подключенные к Интернету и сотовой связи, являются источниками огромных объемов данных в разных форматах (текст, картинки, звук, видео и т.д.). Количество собираемых данных растет экспоненциально с каждым днем. Необходимость управления и использования этих данных привела к появлению явления, называемого «большие данные». Термин *«большие данные»* означает поиск новых и более совершенных способов управления большими объемами данных, и извлечения из них информации, одновременно обеспечивая высокую производительность и масштабируемость при разумных затратах.

Термин «большие данные» используется во многих различных сферах, от права до статистики, экономики и вычислительной техники. Аналитик из Gartner Group Дуглас Лейни (Douglas Laney) описал *основные характеристики БД Больших данных*: объем (volume), скорость (velocity) и разнообразие (variety), или *3V:*

- *Объем* относится к количеству данных. На протяжении многих лет данные о миллионах электронных операциях ежедневно хранились в БД. Этот постоянно растущий объем данных быстро достигал размера петабайта и продолжает расти.
- *Скорость* относится не только к скорости, с которой данные растут, но и к необходимости быстрой обработки этих данных для получения информации и анализа. Организациям необходимо не только хранить большие объемы данных, но и быстро обрабатывать их.
- *Разнообразие* относится к тому факту, что собираемые данные представлены в разных форматах. Большая часть этих данных поступает в форматах, не подходящих для обработки типичными операционными БД на основе реляционной модели.

3V иллюстрирует то, что объем данных, собираемых в базы данных, экспоненциально растет в размере и сложности. Традиционные реляционные базы данных предназначены для управления структурированными данными, но не очень подходят для управления и обработки объемов и типов данных, собираемых в современной бизнес-среде:

- Невозможно вписать неструктурированные данные из социальных сетей или сгенерированные датчиком данные в реляционную структуру строк и столбцов.
- Ежедневное добавление миллионов строк мульти-форматных данных неизбежно приведет к необходимости большего объема хранилища, вычислительной мощности и сложных инструментов анализа данных, которые часто недоступны в реляционной среде.
- Анализ данных на основе инструментов OLAP оказался очень успешным в реляционных средах с высоко структурированными данными. Однако добыча полезных данных в огромных объемах неструктурированных данных, собранных из веб-источников, требует другого подхода.

Для создания ценности из больших данных используются новые технологии больших данных – Hadoop, MapReduce и NoSQL.

- *Hadoop* – это высокоскоростное, отказоустойчивое распределенное хранилище и вычислительная среда на основе Java с открытым исходным кодом. Hadoop использует недорогое оборудование для создания кластеров из тысяч компьютерных узлов для хранения и обработки данных. Hadoop имеет несколько модулей, но двумя основными компонентами являются распределенная файловая система Hadoop (HDFS) и MapReduce.
- *Распределенная файловая система Hadoop (HDFS)* – это распределенная, отказоустойчивая система хранения файлов, предназначенная для управления

большими объемами данных с высокой скоростью. HDFS использует *пять типов узлов:* узел имени, в котором хранятся все метаданные о файловой системе; узел контрольной точки, который предназначен для проверки контрольных точек метаданных файловой системы; узел данных, в котором хранятся блоки данных фиксированного размера; клиентский узел, который действует как интерфейс между пользовательским приложением и HDFS; и трекер задач – подчиненный узел клиентского узла.

- *MapReduce* – это интерфейс прикладного программирования (API) с открытым исходным кодом, обеспечивающий быстрые услуги анализа данных. MapReduce распределяет обработку данных между тысячами узлов параллельно, работает со структурированными и неструктурированными данными. Фреймворк MapReduce предоставляет две основные функции: map и reduce. Функция *map* принимает задание и делит его на более мелкие единицы работы, а функция *reduce* собирает все выходные результаты, сгенерированные из узлов, и объединяет их в единый набор результатов.
- *NoSQL* – это крупномасштабная система распределенных баз данных, которая эффективно хранит структурированные и неструктурированные данные.

Технологии Hadoop обеспечивают основу для аналитики больших данных, в которой данные распределяются, реплицируются и обрабатываются параллельно. Hadoop представил новые способы хранения данных и управления ими, а технологии, связанные с Hadoop, породили системы баз данных нового поколения. Hadoop – распределенная модель хранения и обработки файлов.

Термин NoSQL используется для обозначения баз данных нового поколения, которые решают конкретные задачи эпохи больших данных и имеют следующие общие характеристики:

- Не основаны на реляционной модели и SQL, отсюда и название NoSQL.
- Поддерживают высокораспределенные архитектуры БД.
- Обеспечивают линейную масштабируемость, высокую доступность и отказоустойчивость.
- Поддерживают очень большие объемы разреженных данных.
- Ориентированы на производительность, а не на последовательность транзакций.

В отличие от реляционной модели, модель NoSQL представляет различные подходы к хранению и манипулированию данными. Наиболее распространенными из этих подходов являются хранилища значений ключей, базы данных документов, столбцовые и графовые базы данных.

Развитие СУБД всегда было обусловлено поиском новых способов моделирования и управления все более сложными данными реального мира. Исторический путь развития наиболее распространенных моделей данных показан на рисунке 4:

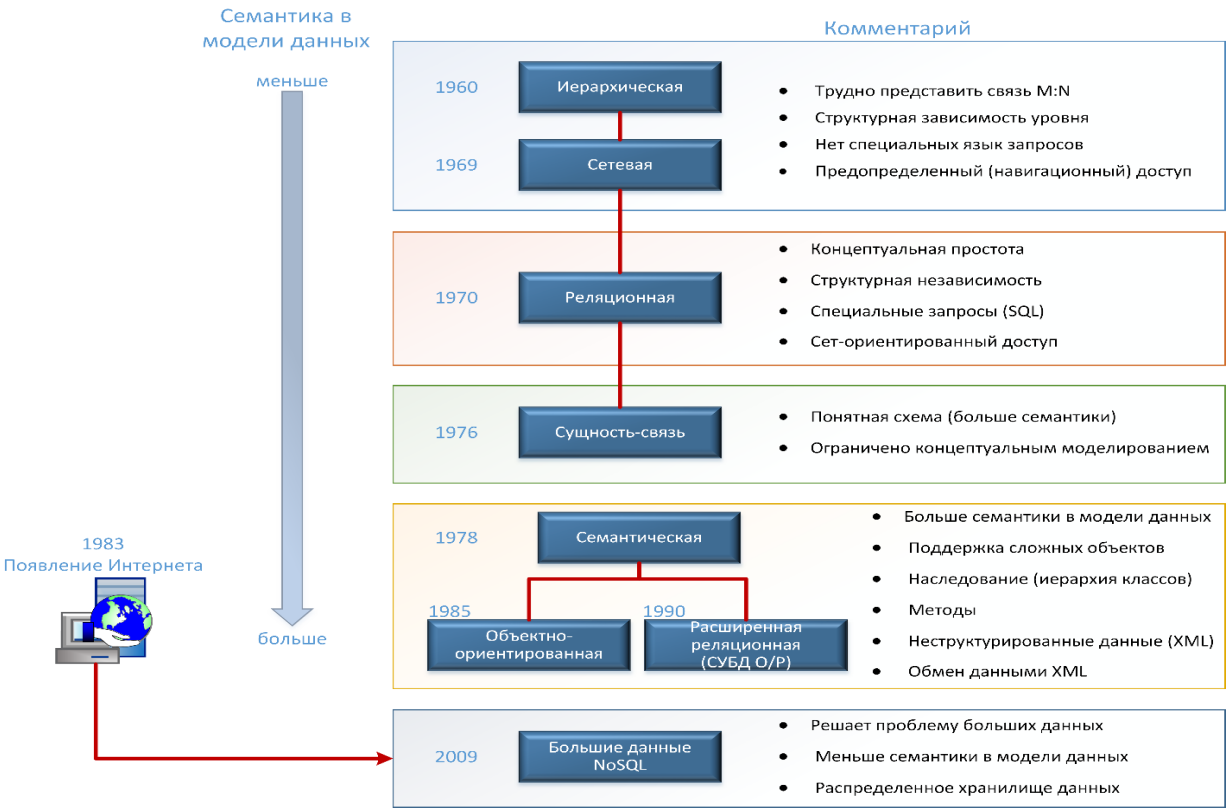


Рис. 4. Развитие моделей данных

2.4. Абстракция данных

Чтобы проиллюстрировать смысл абстракции данных, рассмотрим в качестве примера проектирование самолета. Инженер-проектировщик сначала чертит концепцию самолета, который будет создан. Затем разрабатывают детали, которые помогают перенести основную концепцию в структуру, которая будет произведена. Наконец, технические чертежи переводятся в производственные спецификации для использования на заводе. Процесс производства самолета начинается с высокого уровня абстракции и продолжается до все возрастающего уровня детализации. Процесс производства на заводе не может продолжаться, если детали не определены должным образом, а детали не могут существовать без базовой концептуальной основы.

Разработка БД происходит по тому же принципу. Архитектор БД начинает с абстрактного представления всей среды данных и добавляет детали по мере приближения проекта к реализации. Использование уровней абстракции также может быть очень полезным при интеграции нескольких представлений данных на разных уровнях организации.

В стандарте ГОСТ 34.320-96 определены три уровня абстракции данных: внешний, концептуальный и внутренний. Можно использовать эту платформу для лучшего понимания моделей баз данных.

Внешняя модель – представление пользователей о среде данных. Термин «пользователи» относится к людям, которые используют прикладные программы для манипулирования данными и генерирования информации. Представление внешней модели называется *внешней схемой*.

Концептуальная модель представляет собой общее представление корпоративной БД. То есть концептуальная модель объединяет все внешние представления в единое представление данных на уровне предприятии. Представление концептуальной модели называется *концептуальная схема*. Наиболее широко используемая концептуальная модель – модель ER. Концептуальная модель дает представление о среде данных (на макроуровне). Концептуальная модель не зависит от программного и аппаратного обеспечения. *Программная независимость* означает, что модель не зависит от программного обеспечения СУБД, использованного для реализации модели. *Аппаратная независимость* означает, что модель не зависит от аппаратного обеспечения, используемого при реализации модели. Следовательно, изменения в аппаратном обеспечении или программном обеспечении СУБД не окажут влияния на структуру базы данных на концептуальном уровне. Как правило, термин *логическое проектирование* относится к задаче создания концептуальной модели данных, которая может быть реализована в любой СУБД.

Внутренняя модель – это представление базы данных со стороны СУБД. Внутренняя модель требует от архитектора соответствия характеристик концептуальной модели и ограничений выбранной модели реализации. *Внутренняя схема* отображает конкретное представление внутренней модели с использованием конструкций базы данных, поддерживаемых выбранной базой данных.

Поскольку внутренняя модель зависит от конкретного программного обеспечения базы данных, она называется *программно-зависимой.* Поэтому для изменения программного обеспечения СУБД необходимо изменить внутреннюю модель, чтобы она соответствовала характеристикам и требованиям модели базы данных реализации. Возможность изменения внутренней модели, не затрагивая концептуальную модель называется *логической независимостью.* Внутренняя модель не зависит от оборудования, поскольку на нее не влияет тип компьютера, на котором установлено программное обеспечение. Поэтому изменение в устройствах хранения или даже изменение в операционных системах не повлияет на внутреннюю модель.

Физическая модель работает на самом низком уровне абстракции, описывая способ сохранения данных на носителях данных. Физическая модель требует определения как запоминающих устройств, так и методов доступа, необходимых для доступа к данным в этих устройствах, что делает его зависимым как от программного, так и от аппаратного обеспечения. Используемые структуры хранения зависят от программного обеспечения (СУБД и операционной системы) и от типа устройств хранения, с которыми может работать компьютер.

Хотя реляционная модель не требует от разработчика заботиться о физических характеристиках хранения данных, реализация реляционной модели может потребовать точной настройки физического уровня для повышения производительности.

Физическая модель зависит от СУБД, методов доступа к файлам и типов аппаратных устройств хранения, поддерживаемых операционной системой. Возможность изменить физическую модель, не затрагивая внутреннюю модель, называется *физической независимостью.*

2.5. Итоги

- Модель данных – это абстракция реального мира в среде данных. Основными компонентами моделирования данных являются сущности, атрибуты, связи и ограничения. Бизнес-правила используются для определения и идентификации основных компонентов моделирования в конкретной реальной среде.
- Иерархические и сетевые модели данных были ранними моделями, которые больше не используются, но некоторые концепции встречаются в современных моделях данных.
- Реляционная модель является текущим стандартом реализации базы данных. В реляционной модели данные хранятся в таблицах. Таблицы связаны друг с другом посредством общих значений в общих атрибутах. Модель сущность-связь (ER) является популярным графическим инструментом для моделирования данных, который дополняет реляционную модель.
- Объектно-ориентированная модель данных (ООМД) использует объекты в качестве базовой структуры моделирования. Как и сущность реляционной модели, объект описывается его фактическим содержанием. Однако, в отличие от сущности, объект также включает в себя информацию о связях между фактами, и с другими объектами, что придает его данным больше значения.
- Реляционная модель приняла много объектно-ориентированных (ОО) расширений, чтобы стать расширенной реляционной моделью данных (РРМД). Для реализации РРМД были разработаны системы управления объектными / реляционными базами данных (СУБД О/Р). На данный момент ООМД в основном используется в специализированных инженерных и научных приложениях, в то время как РРМД в первую очередь ориентирована на бизнес-приложения.
- Технологии больших данных, такие как Hadoop, MapReduce и NoSQL обеспечивают распределенную, отказоустойчивую и эффективную поддержку для анализа больших данных. NoSQL – это базы данных нового поколения, которые не используют реляционную модель и предназначены для удовлетворения специфических потребностей организаций больших данных. Базы данных NoSQL предлагают распределенные хранилища данных, которые обеспечивают высокую масштабируемость, доступность и отказоустойчивость, жертвуя согласованностью данных и перенося бремя поддержки отношений и целостности данных на программный код.
- Требования к моделированию данных являются функцией различных представлений данных и уровней абстракции данных. ГОСТ 34.320-96 описывает три уровня абстракции данных: внешний, концептуальный и внутренний. Четвертый и самый низкий уровень абстракции данных, называемый физическим уровнем, связан исключительно с физическими методами хранения.

Лекция 3. Реляционные базы данных

3.1. Введение

База данных хранит и управляет как данными, так и метаданными. СУБД управляет и контролирует доступ к данным и структуре базы данных. Расположение СУБД между приложением и базой данных устраняет большинство ограничений, присущих файловой системе. Результатом такой гибкости, однако, является гораздо более сложная физическая структура. Реляционная модель данных позволяет разработчику сосредоточиться на логическом представлении данных и их взаимосвязи, а не на деталях физического хранения.

Практическое значение логического представления заключается в том, что оно служит напоминанием о простой концепции хранения данных в файлах. Хотя использование таблицы, совершенно не похожее на использование файла, имеет преимущества структурной независимости и независимости данных. Таблица действительно напоминает файл с концептуальной точки зрения. Поскольку можно рассматривать связанные записи как хранящиеся в независимых таблицах, модель реляционной базы данных гораздо легче понять, чем иерархическую и сетевую модели. Логическая простота приводит к простым и эффективным методологиям проектирования баз данных.

3.2. Таблицы и их характеристики

Логическое представление реляционной базы данных облегчается созданием связи данных на основе логической конструкции, известной как отношение. Поскольку отношение является математическим понятием, пользователям гораздо проще представить отношение как таблицу.

Таблица – двумерная структура, состоящая из строк и столбцов. Таблицу также называют отношением, потому что создатель реляционной модели Э.Ф. Кодд использовал оба термина в качестве синонимов. Таблица содержит коллекцию связанных сущностей, то есть множество сущностей. Например, таблица СТУДЕНТ содержит множество экземпляров сущностей, каждый из которых представляет студента. По этой причине термины *множество сущностей* и *таблица* часто взаимозаменяемы.

Табличное представление данных позволяет легко определять связи сущностей, тем самым значительно упрощает задачу проектирования базы данных. Характеристики реляционной таблицы приведены в таблице 3.

Таблица 3

Характеристики реляционной таблицы

1	Таблица – двумерная структура, состоящая из строк и столбцов.
2	Каждая строка таблицы (кортеж) представляет собой отдельное вхождение сущности в множество.
3	Каждый столбец таблицы представляет атрибут и имеет собственное имя.

4	Каждое пересечение строки и столбца представляет одно значение данных.
5	Все значения в столбце должны соответствовать одному и тому же формату данных.
6	Каждый столбец имеет определенный диапазон значений, известный как домен атрибута.
7	Порядок строк и столбцов не имеет значения для СУБД.
8	Каждая таблица должна иметь атрибут или комбинацию атрибутов, которые однозначно определяют каждую строку.

Используя таблицу СТУДЕНТ, показанную на рисунке 5, можно сделать следующие выводы, соответствующие пунктам в таблице 3:

Таблица: СТУДЕНТ — База данных: НВГУ

СТ_КОД	СТ_ИМЯ	СТ_ФАМ	СТ_ОТЦ	СТ_ДР	СТ_ГРУП	СТ_КУРС	СТ_ПЕР	ПР_КОД
б-190701	Анатолий	Канаев	Арсениевич	11.03.2003	3902	2	0	9015
б-190702	Павел	Матвиенко	Потапович	15.05.2003	3902	2	0	9015
б-190703	Карл	Сальников	Трофимович	25.01.2003	3902	2	0	9015
б-190704	Владимир	Клинских	Богданович	30.12.2002	3902	2	0	9015
б-190706	Василиса	Сотова	Владимировна	06.11.2002	3902	2	0	9015
б-190707	Оксана	Журавлева	Павловна	31.10.2002	3902	2	0	9015
б-190708	Дарья	Кудрина	Филипповна	10.09.2003	3902	2	0	9015
б-190709	Виктория	Никитина	Леонидовна	22.05.2003	3902	2	0	9015
б-190710	Матвей	Шипеев	Егорович	21.03.2003	3902	2	1	9015
б-190745	Святослав	Балин	Матвеевич	28.06.2003	3902	2	1	9015

```
СТ_КОД      = Код студента
СТ_ИМЯ      = Имя студента
СТ_ФАМ      = Фамилия студента
СТ_ОТЦ      = Отчество студента
СТ_ДР       = Дата рождения студента
СТ_ГРУП     = Номер группы
СТ_КУРС     = Курс
СТ_ПЕР      = Переведен из другого ВУЗ-а
ПР_КОД      = Код куратора
```

Рис. 5. Таблица СТУДЕНТ

1. Таблица СТУДЕНТ – двумерная структура, состоящая из 10 строк (кортежей) и 9 столбцов (атрибутов).

2. Каждая строка в таблице СТУДЕНТ описывает одно вхождение сущности во множество сущностей. Например, строка 4 на рисунке 5 описывает студента по имени «Владимир Богданович Клинских». Учитывая содержимое таблицы, множество сущностей СТУДЕНТ включает в себя десять отдельных сущностей (строк) или студентов.

3. Каждый столбец представляет атрибут и имеет отдельное имя.

4. Все значения в столбце соответствуют характеристикам атрибута. Данные должны быть классифицированы в соответствии с их форматом и функцией. Хотя разные СУБД могут поддерживать разные типы данных, большинство из них поддерживают по крайней мере следующие:

а) *Числовой*. Можно использовать числовые данные для выполнения арифметических процедур.

b) *Символ*. Символьные данные, или текстовые данные могут содержать любой символ или символ, не предназначенный для математических операций.

c) *Дата*. Атрибуты даты содержат календарные даты, хранящиеся в специальном формате.

d) *Логический*. Логические данные могут иметь только истинные/ ложные (да/нет или 1/0) значения.

5. Диапазон допустимых значений столбца называется его доменом. Поскольку значения СТ_КУРС ограничены диапазоном 1–4 включительно, доменом является [1,4].

6. Порядок строк и столбцов не имеет значения для пользователя.

7. Каждая таблица должна иметь первичный ключ. *Первичный ключ (PK)* – это атрибут или комбинация атрибутов, которые однозначно идентифицируют данную строку. В этом случае СТ_КОД (код студента) является первичным ключом.

3.3. Ключи

В реляционной модели ключи используются для обеспечения идентификации каждой строки и для установления связей между таблицами. *Ключ* состоит из одного или нескольких атрибутов, которые идентифицируют другие атрибуты. Например, код студента идентифицирует все атрибуты студента, такие как имя и дата рождения.

Роль ключа основана на концепции определения. *Определение* – это состояние, в котором знание значения одного атрибута позволяет определить значение другого. Идея определения не уникальна для среды базы данных. Например, по формуле: *{выручка – себестоимость = прибыль}*, если даны выручка и себестоимость, можно определить прибыль. Если известны прибыль и выручка, можно определить себестоимость. Учитывая любые два значения, можно определить третье. Однако определение в среде базы данных обычно основывается не на формуле, а на отношениях между атрибутами.

Если посмотреть атрибуты таблицы СТУДЕНТ на рисунке 5, можно увидеть взаимосвязь между атрибутами. Если дано значение для СТ_КОД, то можно определить значение для СТ_ИМЯ, поскольку одно и только одно значение СТ_ИМЯ связано с заданным значением СТ_КОД. Определенная терминология и обозначения используются для описания связи на основе определения. Связь называется *функциональной зависимостью*, которая означает, что значение одного или нескольких атрибутов определяет значение одного или нескольких других атрибутов. Стандартная запись для представления зависимости между СТ_КОД и СТ_ИМЯ выглядит следующим образом:

СТ_КОД → СТ_ИМЯ

В этой функциональной зависимости атрибут, значение которого определяет другое, называется *определителем* или ключом. Атрибут, значение которого определяется другим атрибутом, называется *зависимым*. Используя эту терминологию, было бы правильно сказать, что СТ_КОД является определителем, а СТ_ИМЯ является зависимым. СТ_КОД функционально определяет СТ_ИМЯ, а СТ_ИМЯ функционально зависит от СТ_КОД. Функциональная зависимость может включать определитель, который содержит более одного

атрибута и несколько зависимых атрибутов. Обратитесь к таблице СТУДЕНТ для следующего примера:

СТ_КОД → (СТ_ИМЯ, СТ_ФАМ, СТ_ДР)

и

(СТ_ИМЯ, СТ_ФАМ, СТ_ДР, СТ_ГРУП) → (СТ_КУРС, СТ_ПЕР).

Определители, состоящие из более чем одного атрибута, требуют особого рассмотрения. Возможна функциональная зависимость, в которой определитель содержит атрибуты, не являющиеся необходимыми для отношения. Рассмотрим следующие две функциональные зависимости:

СТ_КОД → СТ_ДР

(СТ_КОД, СТ_ИМЯ) → СТ_ДР.

Во второй функциональной зависимости определитель включает СТ_ИМЯ, но этот атрибут не является обязательным для взаимосвязи. Функциональная зависимость действительна, поскольку при наличии пары значений для СТ_КОД и СТ_ИМЯ для СТ_ДР возможно только одно значение. Более конкретный термин – *полная функциональная зависимость* – используется для обозначения функциональных зависимостей, в которых весь набор атрибутов в определителе необходим для связи. Следовательно, зависимость, показанная в предыдущем примере, является функциональной зависимостью, но не является полной функциональной зависимостью.

Несколько разных типов ключей используется в реляционной модели.

Составной ключ – это ключ, состоящий из нескольких атрибутов. Атрибут, который является частью ключа, называется атрибутом ключа. Например,

СТ_КОД → СТ_ДР

(СТ_ИМЯ, СТ_ФАМ, СТ_ДР, СТ_ГРУП) → СТ_ПЕР.

В первой функциональной зависимости СТ_КОД является примером ключа, состоящего только из одного атрибута ключа. Во второй функциональной зависимости (СТ_ИМЯ, СТ_ФАМ, СТ_ДР, СТ_ГРУП) представляет собой составной ключ, состоящий из четырех ключевых атрибутов.

Суперключ – это ключ, который может однозначно идентифицировать любую строку в таблице. Другими словами, суперключ функционально определяет каждый атрибут в строке. В таблице СТУДЕНТ СТ_КОД – это суперключ, как и составные ключи (СТ_КОД, СТ_ИМЯ), (СТ_КОД, СТ_ИМЯ, СТ_ФАМ) и (СТ_КОД, СТ_ФАМ, СТ_ДР, СТ_КУРС). Фактически, поскольку СТ_КОД один является суперключом, любой составной ключ, в котором в качестве атрибута ключа указан СТ_КОД, также будет суперключом. Не все ключи являются суперключами. Например, НВГУ определяет номер группы на основе курса. Поэтому можно написать:

СТ_КУРС → СТ_ГРУП.

Однако номер группы конкретного студента не зависит от курса. Вполне возможно найти студента с несоответствующим номером группы.

Особый тип суперключа называется ключом-кандидатом. *Ключ-кандидат* – это минимальный суперключ, то есть суперключ без каких-либо ненужных атрибутов. Ключ-кандидат основан на полной функциональной зависимости. Например, СТ_КОД будет ключом-кандидатом. (СТ_КОД, СТ_ИМЯ) – это суперключ, но он не является ключом-кандидатом, поскольку СТ_ИМЯ может быть удален, а ключ все равно будет суперключом. Таблица может иметь много разных ключей-кандидатов. Если в таблицу СТУДЕНТ включать номера паспортов учащихся в виде СТ_ПАС, то это тоже будет ключ-кандидат. Ключи-кандидаты называются кандидатами, потому что они являются приемлемыми вариантами, которые архитектор БД выберет при определении первичного ключа. Первичный ключ – это ключ-кандидат, выбранный в качестве основного средства уникальной идентификации строк таблицы.

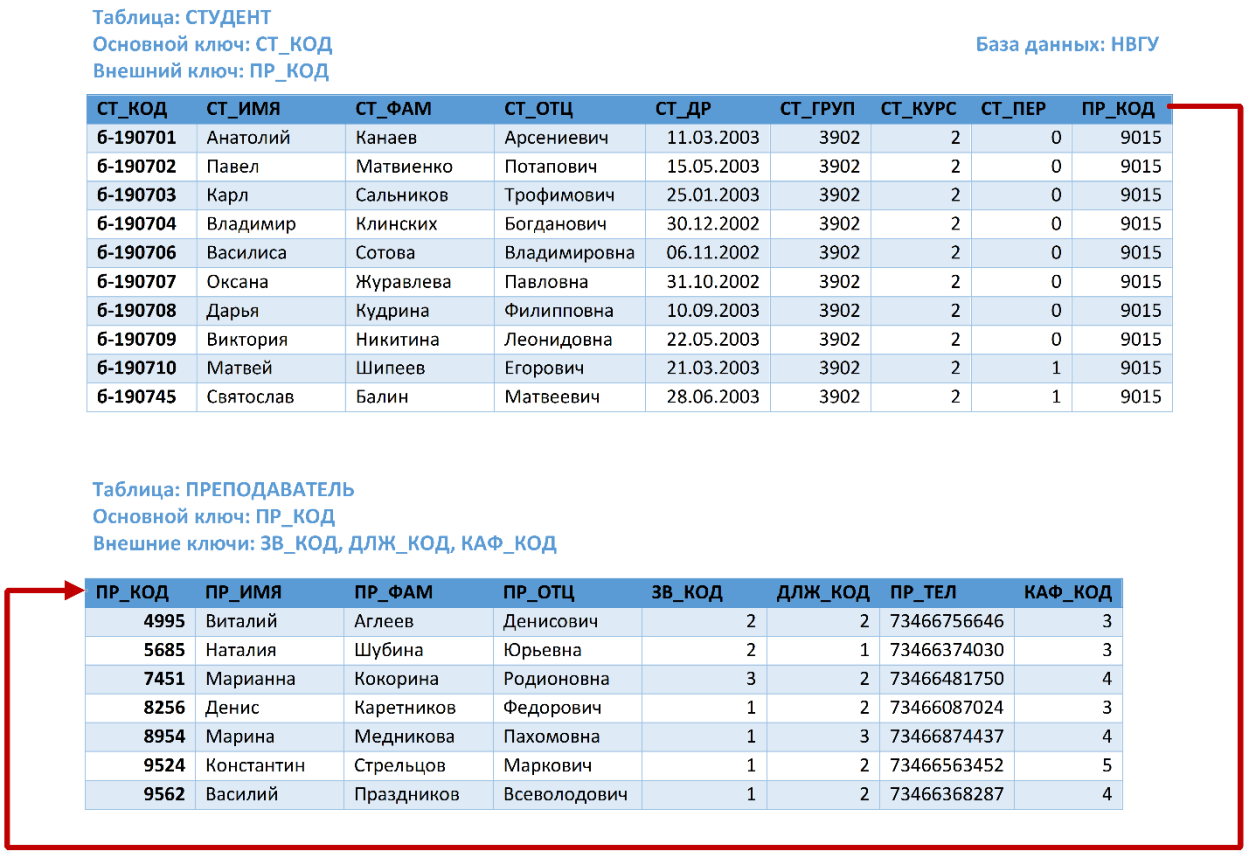
Целостность сущности – это условие, при котором каждая строка в таблице имеет свой уникальный идентификатор. Для обеспечения целостности объекта первичный ключ имеет два требования:

1. Все значения в первичном ключе должны быть уникальными;
2. Ни один атрибут ключа в первичном ключе не может быть пустым.

Пустые значения проблематичны в реляционной модели. *NULL* – это отсутствие какого-либо значения данных, и оно никогда не допускается ни в одной части первичного ключа. С теоретической точки зрения можно утверждать, что таблица, которая содержит NULL, вообще не является реляционной таблицей. Однако в практике некоторые NULL значения неизбежны. Например, не у всех студентов определены кураторы. Большое количество NULL-значений является признаком плохого дизайна. NULL может представлять любое из следующего:

- Неизвестное значение атрибута;
- Известное, но отсутствующее значение атрибута;
- Условие «неприменимо».

NULL-значения могут создавать проблемы при использовании агрегатных функций. NULL-значения могут создавать логические проблемы, когда реляционные таблицы связаны между собой.



СТ_КОД	СТ_ИМЯ	СТ_ФАМ	СТ_ОТЧ	СТ_ДР	СТ_ГРУП	СТ_КУРС	СТ_ПЕР	ПР_КОД
6-190701	Анатолий	Канаев	Арсениевич	11.03.2003	3902	2	0	9015
6-190702	Павел	Матвиенко	Потапович	15.05.2003	3902	2	0	9015
6-190703	Карл	Сальников	Трофимович	25.01.2003	3902	2	0	9015
6-190704	Владимир	Клинских	Богданович	30.12.2002	3902	2	0	9015
6-190706	Василиса	Сотова	Владимировна	06.11.2002	3902	2	0	9015
6-190707	Оксана	Журавлева	Павловна	31.10.2002	3902	2	0	9015
6-190708	Дарья	Кудрина	Филипповна	10.09.2003	3902	2	0	9015
6-190709	Виктория	Никитина	Леонидовна	22.05.2003	3902	2	0	9015
6-190710	Матвей	Шипеев	Егорович	21.03.2003	3902	2	1	9015
6-190745	Святослав	Балин	Матвеевич	28.06.2003	3902	2	1	9015

ПР_КОД	ПР_ИМЯ	ПР_ФАМ	ПР_ОТЧ	ЗВ_КОД	ДЛЖ_КОД	ПР_ТЕЛ	КАФ_КОД
4995	Виталий	Аглеев	Денисович	2	2	73466756646	3
5685	Наталия	Шубина	Юрьевна	2	1	73466374030	3
7451	Марианна	Кокорина	Родионовна	3	2	73466481750	4
8256	Денис	Каретников	Федорович	1	2	73466087024	3
8954	Марина	Медникова	Пахомовна	1	3	73466874437	4
9524	Константин	Стрельцов	Маркович	1	2	73466563452	5
9562	Василий	Праздников	Всеволодович	1	2	73466368287	4

Рис. 6. Пример реляционной БД

Помимо своей роли в предоставлении уникального идентификатора каждой строке таблицы, первичный ключ может играть дополнительную роль в управляемой избыточности, которая позволяет работать реляционной модели. Отличительной чертой реляционной модели является то, что отношения между таблицами реализуются с помощью общих атрибутов. Например, на рисунке 6 показаны таблицы СТУДЕНТ и ПРЕПОДАВАТЕЛЬ, связанные через общий атрибут ПР_КОД. В таблице СТУДЕНТ атрибут ПР_КОД называется внешним ключом. *Внешний ключ (foreign key – FK)* – первичный ключ одной таблицы, который был помещен в другую таблицу для создания общего атрибута. Одним из преимуществ использования правильного соглашения об именах для атрибутов таблицы является то, что можно легче идентифицировать внешние ключи. Например, поскольку в таблице ПРЕПОДАВАТЕЛЬ на рисунке 6 используется правильное соглашение об именах, можно идентифицировать три внешних ключа (ЗВ_КОД, ДЛЖ_КОД и КАФ_КОД), которые подразумевают наличие трех других таблиц в базе данных (ЗВАНИЕ, ДОЛЖНОСТЬ и КАФЕДРА), связанных с таблицей ПРЕПОДАВАТЕЛЬ.

В обеспечении целостности базы данных первичный и внешний ключи играют важную роль. Внешние ключи используются для обеспечения *ссылочной целостности* – условия, при котором каждая ссылка на экземпляр сущности является действительной. Другими словами, каждое значение внешнего ключа должно быть либо NULL, либо действительным значением в первичном ключе связанной таблицы.

3.4. Правила целостности

Правила целостности реляционной базы данных очень важны для хорошего проекта базы данных. Системы управления реляционными базами данных автоматически применяют правила целостности, но гораздо безопаснее убедиться, что проект БД соответствует правилам сущности и ссылочной целостности. Эти правила приведены в таблице 4.

Таблица 4

Правила целостности

Требование	Цель	Пример
Целостность сущности		
Все записи первичного ключа должны являться уникальными, никакая часть первичного ключа не может быть нулевой.	Каждая строка будет иметь уникальный идентификатор, и значения внешнего ключа должны правильно ссылаться на значения первичного ключа.	У каждого студента уникальный номер, и при этом он не может быть нулевым; следовательно, все студенты однозначно идентифицируются по номеру.
Ссылочная целостность		
Внешний ключ может иметь либо пустое значение, если он не является частью первичного ключа своей таблицы, либо значение, которое соответствует первичному ключу в таблице, с которой он связан. Каждое ненулевое значение внешнего ключа должно иметь ссылку на существующее значение первичного ключа.	Атрибут может не иметь соответствующего значения, но значение не пустое, оно должно совпадать с соответствующим первичным ключом. Применение правила ссылочной целостности делает невозможным удаление строки в одной таблице, первичный ключ которой имеет обязательные совпадающие значения внешнего ключа в другой таблице.	У студента еще может не быть назначенного куратора (номер), но нельзя поставить номер несуществующего куратора.

Чтобы избежать нулевых значений, некоторые разработчики используют специальные коды, известные как *флаги,* для указания отсутствия какого-либо значения.

Другими правилами целостности, которые применяются в реляционной модели, являются ограничения NOT NULL и UNIQUE. Ограничение NOT NULL гарантирует, что каждая строка в таблице имеет значение для этого столбца. Ограничение UNIQUE гарантирует отсутствие повторяющихся значений для этого столбца.

3.5. Реляционная алгебра

Данные не имеют ценности, если ими нельзя манипулировать для получения полезной информации. *Реляционная алгебра* определяет теоретический способ манипулирования содержимым таблицы с использованием реляционных операторов.

Реляционная модель основана на математических принципах, а манипулирование данными в базе данных может быть описано в математических терминах. Программистам в области баз данных не нужно писать математические формулы для работы с данными. Данные обрабатываются с использованием мощных языков, таких как SQL, которые скрывают основную математику. Однако для создания эффективных и результативных запросов, важно понимание основополагающих принципов.

Реляционные операторы имеют свойство *замыкания;* то есть результатом применения операторов реляционной алгебры на существующие таблицы являются новые отношения. Некоторые операторы являются фундаментальными, другие могут быть получены с использованием фундаментальных операторов.

SELECT/RESTRICT (выбор) – унарный оператор, использует только одну таблицу в качестве ввода. Возвращает значения для всех строк, найденных в таблице, которые удовлетворяют заданному условию. SELECT может использоваться для вывода списка всех строк или может выдавать только те строки, которые соответствуют указанному критерию. Другими словами, SELECT возвращает горизонтальное подмножество таблицы. SELECT не будет ограничивать возвращаемые атрибуты, поэтому все атрибуты таблицы будут включены в результат.

Формально SELECT обозначается строчной греческой буквой σ (сигма). За σ следует условие, которое пишется в индексе, а потом отношение в скобках. Например, чтобы выбрать все строки в таблице СТУДЕНТ, которые имеют значение «2» в атрибуте СТ_КУРС, нужно написать следующее:

$$\sigma_{\text{ст\_курс} = 2} (\text{студент}).$$

PROJECT (проецирование) – возвращает все значения для выбранных атрибутов. Это также унарный оператор, принимающий в качестве входных данных только одну таблицу. PROJECT возвращает только запрошенные атрибуты в том порядке, в котором они запрашиваются. Другими словами, PROJECT выдает вертикальное подмножество таблицы. PROJECT не ограничивает возвращаемые строки, поэтому все строки указанных атрибутов будут включены в результат.

Формально PROJECT обозначается греческой буквой π (пи). После π следует список атрибутов, которые пишутся в индексе, а потом отношение, указанное в скобках. Например, чтобы проецировать атрибуты СТ_ИМЯ и СТ_ФАМ в таблице СТУДЕНТ, нужно написать следующее:

$$\pi_{\text{ст\_имя, ст\_фам}} (\text{студент}).$$

Поскольку реляционные операторы имеют свойство замыкания, то есть они принимают отношения в качестве входных данных, и создают отношения в качестве выходных данных,

операторы можно комбинировать. Например, можно объединить два предыдущих оператора, чтобы найти имя и фамилию студента с кодом «б-190701»:

$$\pi_{\text{ст\_имя, ст\_фам}} (\sigma_{\text{ст\_код = б-190701}} (\text{студент})).$$

UNION (объединение) – объединяет все строки из двух таблиц, исключая повторяющиеся строки. Для использования в UNION таблицы должны иметь одинаковые атрибуты; другими словами, столбцы и домены должны быть совместимы. Когда две или более таблиц имеют одинаковое количество столбцов и их соответствующие столбцы используют одинаковые или совместимые домены, они называются *совместимыми с объединением.*

UNION обозначается символом U. Если отношения СТУДЕНТ и ПРЕПОДАВАТЕЛЬ совместимы с объединением, то объединение между ними будет обозначаться следующим образом:

студент U преподаватель.

INTERSECT (пересечение) – возвращает только те строки, которые есть в обеих таблицах. Как и в случае с UNION, таблицы должны быть совместимы с объединением для получения правильных результатов. Например, нельзя использовать INTERSECT, если один из атрибутов является числовым, а другой – символьным. Чтобы строки считались одинаковыми в обеих таблицах и появлялись в результате INTERSECT, все строки должны быть точными дубликатами.

INTERSECT обозначается символом ∩. Если отношения СТУДЕНТ и ПРЕПОДАВАТЕЛЬ совместимы с объединением, то пересечение между ними будет обозначаться следующим образом:

студент ∩ преподаватель.

DIFFERENCE (разность) – выдает все строки в одной таблице, которые не найдены в другой таблице; то есть вычитает одну таблицу из другой. Как и в случае с UNION, таблицы должны быть совместимы с объединением для получения правильных результатов.

DIFFERENCE обозначается символом минус (-). Если отношения СТУДЕНТ и ПРЕПОДАВАТЕЛЬ совместимы с объединением, тогда разность между ними будет записана следующим образом:

студент - преподаватель.

PRODUCT (умножение) – или декартово произведение возвращает все возможные пары строк из двух таблиц. Следовательно, если в одной таблице 6 строк, а в другой 3 строки, PRODUCT выдает список, состоящий из 6 × 3 = 18 строк.

PRODUCT обозначается символом умножения (×). Умножение отношений СТУДЕНТ и ПРЕПОДАВАТЕЛЬ будет записано следующим образом:

студент × преподаватель.

JOIN (соединение) – позволяет объединять информацию из двух или более таблиц. JOIN – реальная сила реляционной БД, позволяющая использовать независимые таблицы, связанные общими атрибутами. Таблицы СТУДЕНТ и ПРЕПОДАВАТЕЛЬ, показанные на рисунке 6, будут использованы для иллюстрации нескольких типов объединений.

Естественное объединение связывает таблицы, выбирая только строки с общими значениями в их общих атрибутах. Естественное объединение является результатом трехэтапного процесса:

1. Выполняется PRODUCT таблиц.
2. Выполняется SELECT над результатами шага 1, чтобы получить только те строки, для которых значения ПР_КОД равны. Общие столбцы называются *столбцами соединения*.
3. Выполняется PROJECT над результатами шага 2, чтобы получить одну копию каждого атрибута, тем самым устраняя дубликаты столбцов.

Окончательный результат естественного объединения приводит к таблице, которая не включает несопоставленные пары и предоставляет только совпадения.

Естественное соединение обычно упоминается как JOIN в формальных обработках. JOIN обозначается символом ⋈. Соединение отношений СТУДЕНТ и ПРЕПОДАВАТЕЛЬ будет иметь следующий вид:

студент ⋈ преподаватель.

JOIN не является фундаментальным оператором реляционной алгебры, он может быть получен из других.

Другая форма объединения, известная как *equijoin*, связывает таблицы на основе условия равенства, которое сравнивает указанные столбцы каждой таблицы. Результат не устраняет повторяющиеся столбцы, и условие или критерий, используемые для объединения таблиц, должны быть явно определены. Equijoin берет свое имя от оператора сравнения равенства (=), используемого в условии. Если используется любой другой оператор сравнения, соединение называется *тэта-соединением(θ)*.

Формально θ-соединение считается продолжением естественного соединения. θ-соединение обозначается добавлением θ-индекса после символа JOIN: $\bowtie_\theta$. equijoin – это особый тип θ-соединения.

Предыдущие объединения часто классифицируются как внутреннее объединение. *Внутреннее соединение* возвращает только совпадающие записи из таблиц, которые объединяются. Во *внешнем соединении* совпадающие пары будут сохранены, а любые несопоставленные значения в другой таблице останутся пустыми. Внешнее соединение не является противоположностью внутреннего соединения. Правильно думать о внешнем соединении как о «внутреннем соединении плюс». Внешнее соединение возвращает все сопоставленные записи, которые возвращает внутреннее соединение, плюс несопоставленные записи одной из таблиц. Если для таблиц СТУДЕНТ и ПРЕПОДАВАТЕЛЬ создается внешнее соединение, возможны два сценария:

• *Левое внешнее соединение* дает все строки в таблице СТУДЕНТ, включая те, которые не имеют совпадающего значения в таблице ПРЕПОДАВАТЕЛЬ.

• *Правое внешнее соединение* дает все строки в таблице ПРЕПОДАВАТЕЛЬ, включая те, которые не имеют совпадающих значений в таблице СТУДЕНТ.

DIVIDE (Разделить) – используется для получения отношения, которое состоит из унарных кортежей, включающих такие значения первого атрибута кортежей первого операнда, что множество значений второго атрибута (при фиксированном значении первого атрибута) включает множество значений второго операнда.

Оператор DIVIDE обозначается символом деления ÷. Учитывая два отношения R и S, разделение их будет записано как r ÷ s.

3.6. Словарь данных и системный каталог

Словарь данных предоставляет описание всех таблиц в базе данных, созданной архитектором БД. Таким образом, словарь данных содержит метаданные – все имена и характеристики атрибутов для каждой таблицы в системе.

Словарь данных иногда описывается как «база данных разработчика БД», потому что он записывает проектные решения о таблицах и их структурах.

Как и словарь данных, системный каталог содержит метаданные. *Системный каталог* может быть описан как подробный словарь системных данных, который описывает все объекты в базе данных, включая данные об именах таблиц, создателе и дате создания таблицы, количестве столбцов в каждой таблице, типе данных, соответствующих каждому столбцу, именах файлов индекса, индексе создателя, авторизованных пользователях и праве доступа.

Поскольку системный каталог содержит всю необходимую информацию словаря данных, термины системный каталог и словарь данных часто используются взаимозаменяемо. Современные СУБД обычно предоставляют только системный каталог, из которого может быть получена информация и словарь данных.

Системный каталог автоматически создает документацию базы данных. По мере добавления новых таблиц в базу данных эта документация также позволяет СУБД проверять и устранять омонимы и синонимы. *Омонимы* – это слова с одинаковым написанием и звучанием, но разные по значению – такие, как бор (сосновый лес и химический элемент), или ключ (открыватель и родник). В контексте базы данных слово «омоним» указывает на использование одного и того же имени для обозначения разных атрибутов. Например, можно использовать ПР_КОД для атрибута "код преподавателя" в таблице ПРЕПОДАВАТЕЛЬ и для атрибута "код предмета" в таблице ПРЕДМЕТ. Разработчики БД должны избегать омонимов; словарь данных очень полезен в этом отношении.

В контексте базы данных *синоним* является противоположностью омонима и указывает на использование разных имен для описания одного и того же атрибута. Например, дисциплина и предмет относятся к одной и той же сущности. Синонимов следует избегать всякий раз, когда это возможно.

3.7. Связи в реляционной базе данных

Связи классифицируются как один-к-одному (1:1), один-ко-многим (1:M) и многие-ко-многим (M:N или M:M).

- Связь 1:M является идеальной в реляционной модели. Следовательно, этот тип связи должен быть нормой в любом проекте реляционной БД.
- Связь 1:1 должна быть редкой в любом проекте реляционной БД.
- Связь M:N не может быть реализована в реляционной модели. Любую связь M:N можно заменить двумя связями 1:M.

Связь 1:M является нормой для реляционных баз данных. Чтобы увидеть, как моделируются и реализуются такие связи, рассмотрим пример КАРТИНА и ХУДОЖНИК, показанный на рисунке 7.

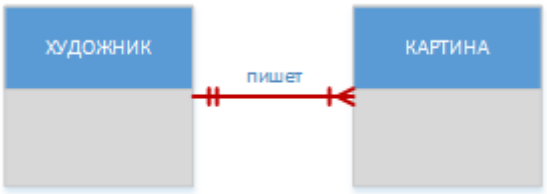


Рис. 7. Связь 1:M между куратором и студентом

Реализация данной модели показана на рисунке 8:

Таблица: ХУДОЖНИК
Основной ключ: ХУД_КОД
Внешний ключ: нет

База данных: Музей

ХУД_КОД	ХУД_ИМЯ	ХУД_ФАМ	ХУД_ОТЧ
256	Иван	Айвазовский	Константинович
259	Иван	Билибин	Яковлевич
261	Владимир	Видинеев	Николаевич
258	Петр	Кончаловский	Петрович
260	Сергей	Соломко	Сергеевич
257	Василий	Тропинин	Андреевич

Таблица: КАРТИНА
Основной ключ: КАР_КОД
Внешний ключ: ХУД_КОД

КАР_КОД	КАР_НАЗВ	ХУД_КОД
2565	В море остров был крутой, Непривольный, нежилой	260
2569	Кораблекрушение	256
2568	Ледяные горы в Антарктиде	256
2564	Он над ней жужжит, кружит	260
2567	Среди волн	256
2566	Черное море	256

Рис. 8. Связь 1:M – КАРТИНА и ХУДОЖНИК

При изучении содержимого таблицы ХУДОЖНИК и КАРТИНА на рисунке 8 нужно обратить внимание на следующие особенности:

- Каждая картина была создана одним и только одним художником, но каждый художник мог создать много картин. Например, художник с кодом 256 (И.К.Айвазовский) имеет четыре работы, хранящиеся в таблице КАРТИНА.

- Существует только одна строка в таблице ХУДОЖНИК для любой данной строки в таблице КАРТИНА, но может быть много строк в таблице КАРТИНА для любой данной строки в таблице ХУДОЖНИК.

Отношение «один-ко-многим» (1:M) легко реализуется в реляционной модели, если поместить первичный ключ стороны «1» в таблицу стороны «многих» в качестве внешнего ключа.

Сущность в связи 1:1 может быть связана только с одной сущностью. Например, преподаватель может возглавлять только один факультет, а факультет может иметь только одного декана. Таким образом, сущности ПРЕПОДАВАТЕЛЬ и ФАКУЛЬТЕТ демонстрируют связь 1:1. Конечно, не все преподаватели возглавляют, и не все обязаны возглавлять факультет. То есть связь между двумя сущностями не является обязательной. Однако на этом этапе обсуждения нужно сосредоточить свое внимание на связи 1:1. Связь 1:1 смоделирована на рисунке 9:

Рис. 9. Связь 1:1 между преподавателем и кафедрой

Реализация данной модели показана на рисунке 10:

Таблица: ПРЕПОДАВАТЕЛЬ
Основной ключ: ПР_КОД
Внешний ключ: КАФ_КОД

База данных: НВГУ

ПР_КОД	ПР_СПЕЦ	ПР_УЧС	ПР_ОТР	ПР_ДЛЖ	КАФ_КОД
12	Рисование, черчение, труд	доктор	педагогических наук	профессор	АДДИ
14	Культурно-просветительская работа	кандидат	педагогических наук	доцент	ППСО
15	Филология	кандидат	филологических наук	доцент	ФМК
16	Машины и аппараты пищевых производств	кандидат	технических наук	доцент	НД
19	история, английский язык	кандидат	исторических наук	доцент	ППСО
20	Филология	кандидат	филологических наук	доцент	ФМК
21	Промышленная теплоэнергетика	кандидат	технических наук	доцент	ЭНЕРГ
22	Филология	кандидат	филологических наук	ст. преподаватель	ФМК
23	Изобразительное искусство и черчение	кандидат	педагогических наук	доцент	АДДИ
25	Дошкольная педагогика и психология	кандидат	педагогических наук	доцент	ППСО
26	Лечебно-профилактическое дело	доктор	медицинских наук	профессор	ПОИР
27	история	кандидат	исторических наук	доцент	ИР
29	Автоматика и телемеханика	доктор	технических наук	профессор	ЭНЕРГ
30	Философия	доктор	философских наук	профессор	КФСН

Таблица: ФАКУЛЬТЕТ
Основной ключ: ФКЛ_КОД
Внешний ключ: ПР_КОД

ФКЛ_КОД	ФКЛ_НАЗВ	ФКЛ_СОКР	ПР_КОД
1	Гуманитарный факультет	ГФ	72
2	Факультет искусств и дизайна	ФИиД	209
3	Факультет педагогики и психологии	ФПиП	103
4	Факультет физической культуры и спорта	ФФКС	63
5	Факультет информационных технологий и математики	ФИТМ	301
6	Факультет экологии и инжиниринга	ФЭИ	96
7	Факультет дополнительного образования	ФДО	235

Рис. 10. Связь 1:M – ПРЕПОДАВАТЕЛЬ и ФАКУЛЬТЕТ

Изучая таблицы на рис. 10, нужно обратить внимание на несколько важных особенностей:

• Связь 1:1 «ПРЕПОДАВАТЕЛЬ возглавляет ФАКУЛЬТЕТ» реализуется при наличии внешнего ключа ПР_КОД в таблице ФАКУЛЬТЕТ. Связь 1:1 рассматривается как частный случай отношения 1:M, в котором сторона «многие» ограничена одним случаем.

• Таблица ПРЕПОДАВАТЕЛЬ содержит внешний ключ КАФ_КОД для реализации отношения 1:M «ПРЕПОДАВАТЕЛЬ относится к КАФЕДРЕ». Это хороший пример того, как сущность может участвовать в двух (и более) связях одновременно.

Связь «многие ко многим» (M:N) не поддерживается напрямую в реляционной среде, но может быть реализована путем создания новой сущности в отношениях 1:M с исходными сущностями. Связь M:N смоделирована на рисунке 11:

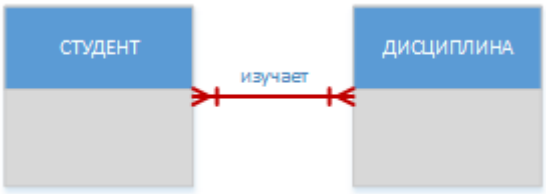


Рис. 11. Связь M:N между студентом и дисциплиной

Особенность диаграммы на рисунке 11 заключается в том, что каждую дисциплину может изучать много студентов, и каждый студент может изучать много дисциплин.

Связи M:N реализуются с помощью *составной сущности* (связующей сущности или ассоциативной сущности). Такая таблица используется для связывания таблиц, и включает в качестве внешних ключей первичные ключи таблиц, которые должны быть связаны. При определении первичного ключа составной таблицы у разработчика базы данных есть два основных параметра: использование комбинации этих внешних ключей или создание нового первичного ключа.

Таблица: СТУДЕНТ
Основной ключ: СТ_КОД
Внешний ключ: нет

СТ_КОД	СТ_ФАМ	СТ_ГРУП	СТ_КУРС
6-190701	Канаев	3902	2
6-190702	Матвиенко	3902	2
6-190703	Сальников	3902	2
6-190704	Клинских	3902	2
6-190706	Сотова	3902	2
6-190707	Журавлева	3902	2
6-190708	Кудрина	3902	2
6-190709	Никитина	3902	2
6-190710	Шипеев	3902	2
6-190745	Балин	3902	2

Таблица: ДИСЦИПЛИНА
Основной ключ: ДИС_КОД
Внешний ключ: нет

ДИС_КОД	ДИС_ИНД	ДИС_НАЗВ	ДИС_ЧАС
1	Б1.В.9	Web-проектирование	36
2	Б1.В.5	Архитектура информационных систем	36
3	Б1.Б.17	Базы данных	36
4	Б1.В.11	Визуальные среды программирования	36
5	Б1.В.10	Защита информации	27
10	Б1.В.9	Компьютерное моделирование	36
11	Б1.В.17	Компьютерные сети и интернет-технологии	27
13	Б1.Б.16	Методы разработки и оценки алгоритмов	27
15	Б1.В.8	Объектно-ориентированное программирование	36
16	Б1.Б.19	Операционные системы	36
17	Б1.В.13	Основы web-программирования	36

База данных: НВГУ

Таблица: РЕГИСТРАЦИЯ
Основной ключ: ДИС_КОД + СТ_КОД
Внешний ключ: ДИС_КОД, СТ_КОД

ДИС_КОД	СТ_КОД
3	6-190701
1	6-190703
2	6-190703
1	6-190704
4	6-190701
3	6-190702
4	6-190702
5	6-190701
2	6-190704
3	6-190703
4	6-190703
21	6-190709
24	6-190706
23	6-190707
26	6-190704
27	6-190703
27	6-190704
24	6-190707
25	6-190706
22	6-190709
23	6-190708
21	6-190710

Рис. 12. Связь M:N – СТУДЕНТ и ДИСЦИПЛИНА

Каждая сущность в ERM представлена таблицей. Поэтому нужно создать составную таблицу РЕГИСТРАЦИЯ, показанную на рисунке 12, чтобы связать таблицы ДИСЦИПЛИНА и СТУДЕНТ. В этом примере первичный ключ таблицы РЕГИСТРАЦИЯ является комбинацией ее внешних ключей ДИС_КОД и СТ_КОД. Однако разработчик мог бы решить создать новый первичный ключ с одним атрибутом, такой как РЕГ_КОД, используя другое значение строки для уникальной идентификации каждой строки.

Поскольку таблица РЕГИСТРАЦИЯ на рисунке 12 связывает две таблицы, ДИСЦИПЛИНА и СТУДЕНТ, она также называется таблицей связи. *Таблица связей* – это реализация составной сущности.

Помимо атрибутов связи, составная таблица РЕГИСТРАЦИЯ может содержать другие атрибуты, как оценка, дата оценки и т.д. Составная таблица может содержать любое количество атрибутов, которые разработчик хочет отслеживать.

3.8. Индексы

Индекс – это упорядоченное расположение, используемое для логического доступа к строкам в таблице. Индекс состоит из ключа индекса и набора указателей. Более формально, индекс – это упорядоченное расположение ключей и указателей. Каждый ключ указывает на местоположение данных, идентифицированных ключом.

Например, чтобы посмотреть все картины, созданные определенным художником в базе данных «Музей» без индекса, нужно прочитать каждую строку в таблице КАРТИНА и посмотреть, соответствует ли ХУД_КОД запрошенному художнику. Однако, если индексировать таблицу КАРТИНА и использовать ключ индекса ХУД_КОД, вам просто нужно найти нужный индекс и соответствующие указатели.

СУБД используют индексы для разных целей. Индекс может использоваться для более эффективного извлечения данных, но индексы также могут использоваться СУБД для получения данных, упорядоченных по определенным атрибутам. Например, создание индекса по фамилии клиента позволит получать данные о клиентах в алфавитном порядке по фамилии. Индексный ключ может состоять из одного или нескольких атрибутов.

Индексы играют важную роль в СУБД для реализации первичных ключей. Когда определен первичный ключ таблицы, СУБД автоматически создает уникальный индекс для столбцов первичного ключа. В *уникальном индексе*, как следует из его имени, ключ индекса может иметь только одно значение указателя (строку), связанное с ним. Таблица может иметь много индексов, но каждый индекс связан только с одной таблицей.

3.9. 12 правил Кодда

В 1985 году др. Э.Ф. Кодд опубликовал список из 12 правил для определения системы реляционных баз данных. Он опубликовал список, опасаясь, что многие поставщики рекламируют продукты как «реляционные», даже если эти продукты не соответствуют реляционным стандартам. Список доктора Кодда, показанный в Таблице 5, является ориентиром для того, какой должна быть действительно реляционная база данных. Но даже доминирующие поставщики баз данных не полностью поддерживают все 12 правил.

Таблица 5

12 правил Кодда

Правило	Название правила	Описание
1	Информационное правило	Вся информация в реляционной БД на логическом уровне должна быть явно представлена единственным способом: значениями в таблицах.
2	Гарантированный доступ	В реляционной БД каждое отдельное (атомарное) значение данных должно быть логически доступно с помощью комбинации имени таблицы, значения первичного ключа и имени столбца.
3	Систематическое лечение нулей	Значения NULL должны быть представлены и обработаны систематически, независимо от типа данных.
4	Динамический онлайн-каталог на основе реляционной модели	Метаданные должны храниться и управляться как обычные данные, то есть в таблицах в БД; должны быть доступны авторизованным пользователям, использующим стандартный язык реляционных баз данных.
5	Комплексный подъязык данных	Реляционная база данных может поддерживать много языков; однако он должен поддерживать один декларативный язык для определения данных и представления, манипулирования данными (интерактивных и программных), ограничения целостности, авторизации и управления транзакциями (начало, принятие и откат).
6	Возможность изменения представлений	Каждое представление должно поддерживать все операции манипулирования данными, которые поддерживают реляционные таблицы: операции выборки, вставки, изменения и удаления данных.
7	Высокоуровневая вставка, обновление и удаление	База данных должна поддерживать вставки, обновления и удаления на уровне набора.
8	Физическая независимость данных	Приложения не должны зависеть от используемых способов хранения данных на носителях, от аппаратного обеспечения компьютеров, на которых находится реляционная база данных.
9	Независимость логических данных	Представление данных в приложении не должно зависеть от структуры реляционных таблиц. Если в процессе нормализации одна реляционная таблица разделяется на две, представление должно обеспечить объединение этих данных, чтобы изменение структуры реляционных таблиц не сказывалось на работе приложений.
10	Независимость контроля целостности	Все ограничения реляционной целостности должны быть определены на реляционном языке и храниться в системном каталоге, а не на уровне приложения.
11	Независимость от расположения	Конечные пользователи и прикладные программы не знают и не зависят от расположения данных (распределенных и локальных баз данных).

Правило	Название правила	Описание
12	Согласование языковых уровней	Если система поддерживает низкоуровневый доступ к данным, пользователям нельзя разрешать обходить правила целостности базы данных.
13	Правило ноль	Все предыдущие правила основаны на понятии, что для того, чтобы считаться реляционной, БД должна использовать для управления исключительно свои реляционные возможности.

3.10. Итоги

- Таблицы являются основными строительными блоками реляционной БД. Набор сущности сохраняется в таблице. Реляционная таблица состоит из пересекающихся строк (кортежей) и столбцов. Каждая строка представляет отдельную сущность, а каждый столбец представляет характеристики (атрибуты) сущностей.
- Ключи определяют функциональные зависимости; то есть другие атрибуты зависят от ключа и поэтому могут быть найдены, если известно значение ключа. Ключ может быть классифицирован как суперключ, ключ-кандидат, первичный ключ, вторичный ключ или внешний ключ.
- Каждая строка таблицы должна иметь первичный ключ. Первичный ключ – атрибут или комбинация атрибутов, которые однозначно идентифицируют все оставшиеся атрибуты в данной строке. Поскольку первичный ключ должен быть уникальным, пустые значения не допускаются.
- Таблицы могут быть связаны общими атрибутами. Таким образом, первичный ключ одной таблицы может отображаться как внешний ключ в другой таблице, с которой он связан. Ссылочная целостность диктует, что внешний ключ должен содержать значения, которые соответствуют первичному ключу в связанной таблице или должны содержать нули.
- Реляционная модель поддерживает несколько функций реляционной алгебры, в том числе SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT и DIVIDE. Понимание основных математических форм этих функций дает более широкое понимание вариантов манипулирования данными.
- Проектирование реляционных БД начинается с определения соответствующих сущностей и их атрибутов, а затем связей между сущностями. Связи (1:1, 1:M и M:N) могут быть представлены с использованием ERD.

Лекция 4. Модель сущность-связь

4.1. Введение

Прежде, чем приступать к созданию системы автоматизированной обработки информации, разработчик должен сформировать понятия о предметах, фактах и событиях, которыми будет оперировать данная система. Для того, чтобы привести эти понятия к модели данных, необходимо заменить их информационными представлениями. Моделирование данных – это первый шаг в процессе проектирования базы данных, служащий мостом между объектами реального мира и моделью базы данных в компьютере. Поэтому важность деталей моделирования данных, выраженных графически через диаграммы взаимосвязей объектов (ERD), невозможно переоценить.

Концептуальные модели, такие как ERM, могут быть использованы для понимания и разработки требований к данным. ERM не зависит от типа базы данных. Концептуальные модели используются в концептуальном проектировании баз данных, в то время как реляционные модели используются в логическом проектировании баз данных.

4.2. Модель сущность-связь

Модель сущность-связь (ERM) представляет концептуальную базу данных с точки зрения пользователя. Она отображает основные компоненты БД: сущности, атрибуты и связи. Поскольку сущность представляет объект реального мира, слова сущность и объект часто используются взаимозаменяемо.

Сущность – это объект, представляющий интерес для пользователя. Сущность в ERM соответствует таблице. ERM ссылается на строку таблицы как экземпляр сущности. В нотациях Чена, Crow's Foot и UML сущность представлена прямоугольником, содержащим имя сущности. Имя сущности, существительное, обычно пишется заглавными буквами.

Атрибуты являются свойствами сущностей. Например, сущность СТУДЕНТ включает в себя атрибуты СТ_ИМЯ, СТ_ФАМ, СТ_ДР и многие другие. В нотации Чена атрибуты представлены овалами и связаны с сущностью линией. Каждый овал содержит имя атрибута, который он представляет. В нотации Crow's Foot атрибуты записываются в поле атрибутов под названием сущности. Поскольку нотация Чена занимает больше места, разработчики БД используют нотацию Crow's Foot.

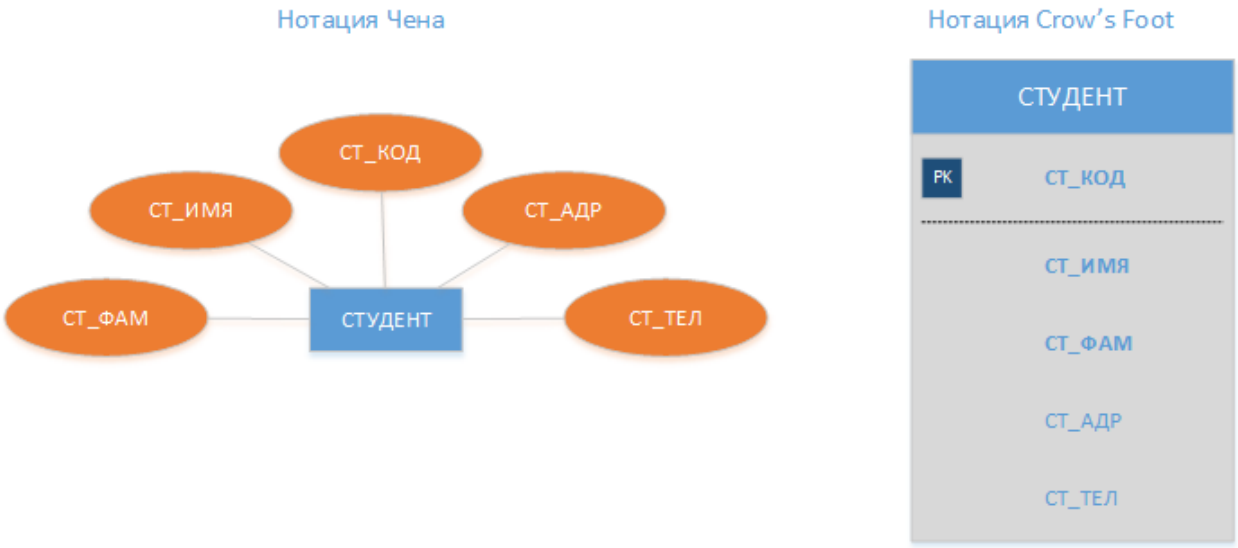


Рис. 13. Сущность СТУДЕНТ: нотации Чена и Crow's Foot

Обязательные и необязательные атрибуты. *Обязательный атрибут* должен иметь значение; другими словами, его нельзя оставлять пустым. Например, СТ_ИМЯ и СТ_ФАМ требуют ввода данных, поскольку предполагается, что все студенты имеют имя и фамилию. Такие атрибуты выделяются жирным шрифтом. Однако у студентов может не быть номера телефона и адреса электронной почты. *Необязательный атрибут* – это атрибут, который не требует значения; следовательно, его можно оставить пустым.

Домены. Атрибуты имеют домен. *Домен* – это набор возможных значений для данного атрибута. Например, домен для атрибута среднего балла записывается (2, 5), поскольку минимально возможное значение равно 2, а максимально возможное значение равно 5. Домен для полового атрибута состоит из двух возможностей: М или Ж. Домен для атрибута даты поступления состоит из всех дат, которые соответствуют диапазону (например, от даты основания университета до текущей даты). Атрибуты могут совместно использовать домен. Например, адрес студента и адрес преподавателя совместно используют один и тот же домен всех возможных адресов.

Идентификаторы. ERM использует *идентификаторы* – один или несколько атрибутов, которые уникально идентифицируют каждый экземпляр сущности. В реляционной модели сущности отображаются в виде таблицы, а идентификатор сущности отображается как первичный ключ таблицы (PK). Идентификаторы подчеркнуты в ERD.

Составные идентификаторы. Идентификатор часто состоит из одного атрибута. Но можно использовать *составной идентификатор* – первичный ключ, состоящий из нескольких атрибутов. Например, администратор базы данных может решить идентифицировать каждый экземпляр сущности СТУДЕНТ, используя составной первичный ключ СТ_КОД и СТ_КУРС. Комбинация СТ_КОД и СТ_КУРС является подходящим ключом-кандидатом.

Составные и простые атрибуты. Атрибуты классифицируются на простые и составные. *Составной атрибут* – атрибут, который может быть разделен для получения дополнительных атрибутов. Например, атрибут СТ_АДРЕС можно разделить на улицу, город,

область и почтовый индекс. Аналогично, атрибут СТ_ТЕЛ можно разделить на код города и номер абонента. *Простой атрибут* – атрибут, который нельзя разделить. Например, возраст, пол и семейное положение – простые атрибуты. Для упрощения сложных запросов целесообразно заменить составные атрибуты на простые.

Однозначные атрибуты. Однозначный атрибут – атрибут, который может иметь только одно значение. Например, у студента может быть только один номер зачетной книжки. Однозначный атрибут не обязательно является простым атрибутом. Номер группы (например, 3912) является однозначным, но он является составным атрибутом, поскольку его можно подразделить на факультет (3), год поступления (9), направление (2) и подгруппу (1).

Многозначные атрибуты. Многозначный атрибут – атрибут, который может иметь много значений. Например, у человека может быть несколько телефонов, каждый со своим номером. Точно так же цвет автомобиля может быть разделен на множество цветов для крыши, кузова и отделки. В нотации Чена многозначные атрибуты показаны двойной линией, соединяющей атрибут с сущностью. Нотация Crow's Foot не идентифицирует многозначные атрибуты.

Реализация многозначных атрибутов. В реляционной таблице каждое пересечение столбцов и строк представляет одно значение данных. Если существуют многозначные атрибуты, разработчик должен выбрать один из двух возможных вариантов действий:

1. В исходной сущности создать новые атрибуты, по одному для каждого компонента исходного многозначного атрибута. Например, атрибут СТ_АДР сущности СТУДЕНТ может быть разделен на новые атрибуты СТ_ГРД, СТ_УЛ и СТ_ОБЛ. Это допустимо, если у каждой сущности одинаковое количество значений для многозначного атрибута.
2. Создать новую сущность, состоящую из компонентов исходного многозначного атрибута. Она позволит разработчику определять значение для разных частей адреса. Потом новая сущность АДРЕС связана с исходной сущностью СТУДЕНТ связью 1:М.

Производные атрибуты. Производный атрибут – атрибут, значение которого вычисляется из других атрибутов. Он не должен физически храниться в базе данных; вместо этого его можно получить с помощью алгоритма. Например, возраст студента – СТ_ВОЗ, может быть найден путем вычисления разницы между текущей датой и СТ_ДР. В нотации Чена производный атрибут соединяется со сущностью пунктирной линией. В нотации Crow's Foot отсутствует метод, позволяющий отличить производный атрибут от других атрибутов.

Связь – это ассоциация, установленная между несколькими сущностями. Имя связи является активным или пассивным глаголом; например, СТУДЕНТ изучает ДИСЦИПЛИНУ, ПРЕПОДАВАТЕЛЬ преподает ДИСЦИПЛИНУ, КАФЕДРА возглавляется ПРЕПОДАВАТЕЛЕМ, а ПОЕЗД управляется МАШИНИСТОМ.

Связь между сущностями всегда действуют в обоих направлениях. Чтобы определить тип связи между сущностями КАРТИНА и ХУДОЖНИК, нужно указать, что:

- ХУДОЖНИК может рисовать много КАРТИН.
- Каждая КАРТИНА рисуется одним ХУДОЖНИКОМ.

Поскольку известны оба направления связи между ХУДОЖНИКОМ и КАРТИНОЙ, легко увидеть, что это связь типа 1:M.

Трудно установить тип связи, если известна только одна сторона. Например:

КАФЕДРОЙ заведует один ПРЕПОДАВАТЕЛЬ.

Неизвестно, являются ли эти отношения 1:1 или 1:M. Поэтому следует задать вопрос «Может ли преподаватель управлять более чем одной кафедрой?». Если ответ «да», связь имеет тип 1:M, а вторая часть пишется как:

ПРЕПОДАВАТЕЛЬ может заведовать многими КАФЕДРАМИ.

Если преподаватель не может заведовать более чем одной кафедрой, связь имеет тип 1:1, а вторая часть пишется как:

ПРЕПОДАВАТЕЛЬ может заведовать только одной КАФЕДРОЙ.

Мощность выражает минимальное и максимальное количество вхождений сущности в связь. В ERD количество элементов указывается путем размещения соответствующих чисел рядом с сущностями в формате (x, y). Первое значение представляет минимальное, а второе – максимальное количество связанных объектов. Большинство разработчиков БД, использующие нотацию Crow’s Foot, не отображают конкретные мощности на самой диаграмме ER, потому что конкретные ограничения не могут быть реализованы через структуру БД. Соответственно, некоторые инструменты моделирования Crow's Foot не печатают числовой диапазон мощности на диаграмме.

Знание минимального и максимального количества вхождений сущностей очень полезно на уровне прикладного программного обеспечения. Например, НВГУ требует, что в каждом классе обучалось не менее 5 студентов. Если аудитория может вместить только 30 студентов, прикладное программное обеспечение должно использовать это количество элементов для ограничения количества. Однако, СУБД не может обрабатывать реализацию мощностей на уровне таблиц – эта возможность предоставляется прикладным программным обеспечением.

Размещение мощностей в диаграмме ER является условным. Нотация Чена помещает мощности на стороне связанной сущности. Диаграммы Crow's Foot и UML располагают мощности рядом с сущностью, к которой они применяются.

Если существование сущности зависит от существования другой сущности, то первая называется *зависимой сущностью*. В терминах реализации сущность зависит от существования, если у него есть обязательный внешний ключ, который не может быть нулевым. Например, если работник хочет предъявить одного или более иждивенцев в целях удержания налогов, уместным будет соотношение «у РАБОТНИКА есть ИЖДИВЕНЕЦ». В этом случае зависимая сущность ИЖДИВЕНЕЦ зависит от существования сущности РАБОТНИК, поскольку она не может существовать отдельно от него в БД.

Если сущность может существовать отдельно от всех связанных с ней сущностей, то она *независима от существования* и называется *сильной* или *регулярной сущностью*. Например, предположим, что предприятие использует детали для производства своей продукции. Кроме того, предположим, что некоторые из этих деталей производятся

собственными силами, а другие закупаются у поставщиков. В этом сценарии вполне возможно, чтобы ДЕТАЛЬ существовала независимо от ПОСТАВЩИКА в связи «ДЕТАЛЬ поставляется ПОСТАВЩИКОМ», поскольку, некоторые детали не имеют поставщика. Таким образом, ДЕТАЛЬ не зависит от существования ПОСТАВЩИКА.

Концепция *силы связи* основана на том, как определяется первичный ключ связанного объекта. Для реализации связи первичный ключ одного объекта (родительский объект, обычно на стороне «1» связи 1:M») отображается как внешний ключ в связанном объекте (дочерний объект, в основном, объект на стороне «M» в отношениях 1:M). Иногда внешний ключ также является компонентом первичного ключа в связанном объекте.

Слабая связь существует, если первичный ключ связанной сущности не содержит компонент первичного ключа родительской сущности.

Сильная связь существует, когда первичный ключ связанной сущности содержит компонент первичного ключа родительской сущности.

Порядок создания и заполнения таблиц очень важен. Сначала нужно заполнить данные со стороны «1» в отношении 1:M, чтобы избежать возможности ошибок ссылочной целостности, независимо от того, являются ли эти отношения слабыми или сильными.

Участие в связях может быть обязательным или необязательным. Связь работает в двух направлениях. Если ХУДОЖНИК рисует КАРТИНУ, то по определению КАРТИНА относится к ХУДОЖНИКУ. Из-за двунаправленной природы связи необходимо определить связность от КАРТИНЫ к ХУДОЖНИКУ и связность от ХУДОЖНИКА к КАРТИНЕ.

Необязательное участие означает, что одно вхождение сущности не требует соответствующего вхождения каждой сущности в конкретную связь. Например, в связи «ПРЕПОДАВАТЕЛЬ заведует КАФЕДРОЙ», не все преподаватели заведуют кафедрой. В нотации Crow's Foot необязательное участие показываются путем рисования маленького круга (O) на стороне необязательной сущности. Наличие необязательной сущности указывает, что ее минимальное количество равно 0.

Обязательное участие означает, что вхождение одного объекта требует соответствующего вхождения объекта в определенных отношениях. Если символ необязательности не указан, предполагается, что сущность существует в обязательных отношениях. Обязательное участие отображается в виде небольшой метки на линии связи. Наличие обязательного отношения указывает, что минимальное количество элементов равно по крайней мере 1.

Таблица 6

Символы Crow's Foot

Символ	Мощность	Описание
	(0, N)	0 или много; сторона «много» не обязательна
	(1, N)	1 или много; сторона «много» обязательна
	(1, 1)	1 и только 1; сторона «1» обязательна
	(0, 1)	0 или 1; сторона 1 является необязательной

Степень связи указывает количество сущностей, ассоциированных со связью. *Унарная связь* существует, когда связь существует внутри одной сущности. *Бинарная связь* существует, когда две разные сущности связаны. *Тернарная связь* существует, когда три сущности связаны. Хотя существуют и более высокие степени, они редки и не имеют конкретных названий. (Например, ассоциация четырех объектов описывается просто как связь с четырьмя степенями). Рисунок 14 показывает эти типы степеней связи.

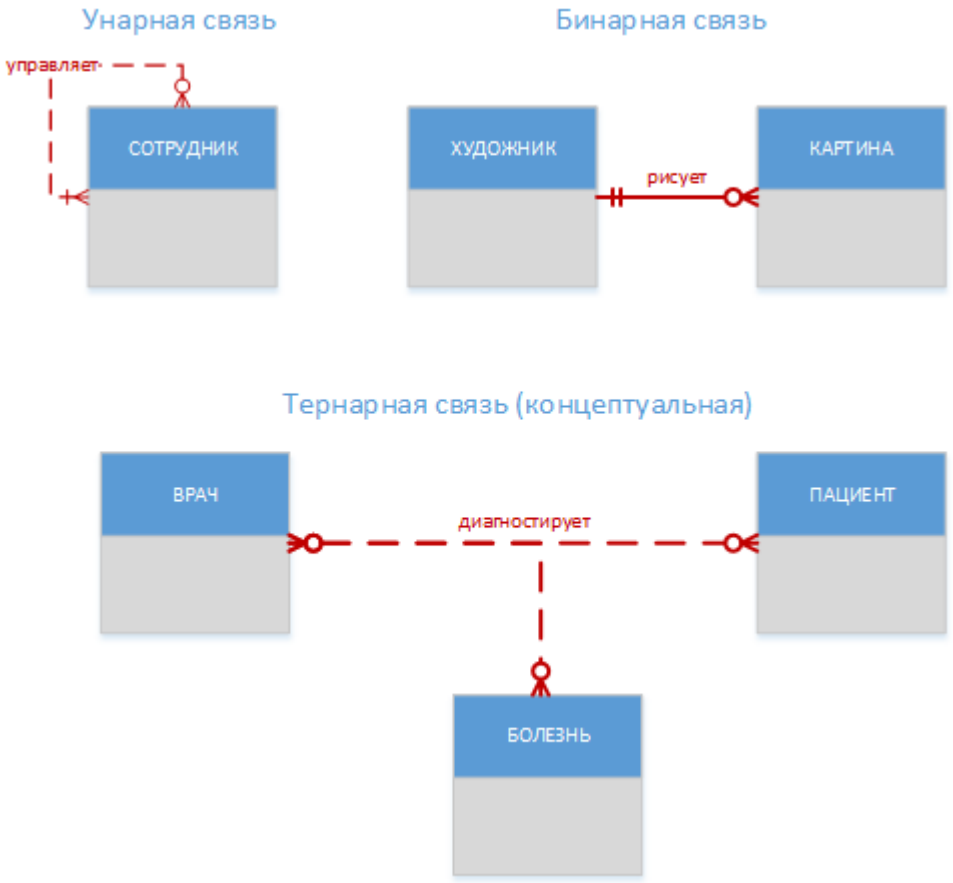


Рис. 14. Типы связей

Унарная связь. В случае унарных связей, показанных на рисунке 14, сотрудник является начальником одного или нескольких сотрудников в этой организации. В этом случае наличие связи «управляет» означает, СОТРУДНИК имеет связь с самим собой. Такие связи известны как рекурсивные.

Бинарная связь. Бинарные связи существуют, когда две сущности связаны. Бинарные связи являются наиболее распространенными. Большинство связей высшего порядка (тернарные и более высокие) по возможности разлагаются на эквивалентные бинарные связи. На рисунке 14 «ХУДОЖНИК рисует одну или несколько КАРТИН» представляет бинарную связь.

Тернарная связь. Хотя большинство связей являются бинарными, использование тернарных и более высоких степеней связей дает разработчику некоторую свободу в отношении семантики проблемы. Тернарная связь подразумевает связь между тремя разными объектами.

Связь M:N является допустимой конструкцией на концептуальном уровне и поэтому часто встречается в процессе моделирования ER. Однако ее реализация, особенно в реляционной модели, требует использования ассоциативной сущности. Ассоциативная сущность находится в связи 1:M с родительскими сущностями и состоит из атрибутов первичного ключа каждого родителя. Кроме того, ассоциативная сущность может иметь свои собственные атрибуты. При использовании обозначения Crow's Foot ассоциативная сущность определяется как сильная связь.

4.3. Расширенная модель сущность-связи

Поскольку сложность моделируемых структур данных возросла, а требования к прикладному программному обеспечению стали более строгими, то и необходимость сбора большего количества информации в модели данных возросла. *Расширенная модель сущность-связи (EERM)* является результатом добавления большего количества семантических конструкций в исходную модель ER. Диаграмма расширенной модели сущность-связи называется *диаграммой EER (EERD)*.

Сотрудники предприятий обладают разными навыками и специальными квалификациями. Разработчики БД должны найти способы группировки сотрудников. Например, университет может группировать сотрудников как преподавателей, административных работников и внешних совместителей. Группировка сотрудников по различным типам дает два важных преимущества:

- Избегать ненужных NULL значений в атрибутах, когда некоторые сотрудники имеют характеристики, остальные – нет.
- Разрешить определенному типу сотрудника участвовать в связях, которые являются уникальными для этого типа сотрудника.

Например, в авиакомпании, в которой работают пилоты, механики, секретари, бухгалтеры, менеджеры баз данных и многие другие сотрудники, пилоты имеют одинаковые атрибуты с другими сотрудниками, такими как фамилия (СТР_ФАМ) и дата найма (СТР_ДТ_НАЙМ). Но, у пилотов есть другие атрибуты, которых нет у других сотрудников. Например, пилоты должны соответствовать особым требованиям, таким как ограничение летного времени, проверка полета и периодическое обучение. Если всех сотрудников сохранить в одной сущности СОТРУДНИК, тогда было бы много NULL значений или пришлось бы создавать много ненужных фиктивных записей.

Исходя из предыдущего обсуждения, сущность ПИЛОТ имеет только атрибуты, уникальные для пилотов, и сущность СОТРУДНИК имеет атрибуты, общие для всех сотрудников. Основываясь на этой иерархии, можно сделать вывод, что ПИЛОТ является подтипом СОТРУДНИКА, а СОТРУДНИК является супертипом ПИЛОТА. В терминах моделирования *супертип сущности* – это общий тип сущности, который связан с одним или несколькими *подтипами сущности*. Супертип сущности содержит общие характеристики, и каждый из подтипов сущности содержит свои собственные уникальные характеристики.

Два критерия помогают разработчику определить, когда использовать подтипы и супертипы:

- С точки зрения пользователя должны быть разные типы сущностей.
- Каждый из типов экземпляров должен иметь один или несколько атрибутов, уникальных для этого типа.

В предыдущем примере, поскольку пилоты отвечают обоим критериям того, чтобы быть специальным видом сотрудников и иметь уникальные атрибуты, которыми не обладают другие сотрудники, целесообразно создать ПИЛОТ как подтип СОТРУДНИКА.

Супертипы и подтипы сущностей организованы в *иерархию специализаций*, которая отображает расположение родительских сущностей и дочерних сущностей. Связь, отображаемая в иерархии специализации, иногда описывается в терминах отношений «является». Например, пилот является работником, механик является работником, бухгалтер является работником. В иерархии специализации подтип может существовать только в контексте супертипа, и каждый подтип может иметь только один супертип, с которым он непосредственно связан. Однако иерархия специализации может иметь много уровней отношений супертипа или подтипа, то есть может быть иерархия специализации, в которой супертип имеет много подтипов. В свою очередь, один из подтипов может стать супертипом в отношении к другим подтипам более низкого уровня.

Иерархии специализации позволяют модели данных захватывать дополнительный смысл в ERD. Иерархия специализации предоставляет средства для:

- Поддержки атрибутов наследования.
- Определения специального атрибута супертипа, известного как дискриминатор подтипа.
- Определения непересекающихся или перекрывающихся ограничений, и полных или частичных ограничений.

Свойство *наследования* позволяет подтипу сущности наследовать атрибуты и связи супертипа. Все подтипы сущностей наследуют атрибут первичного ключа от своего супертипа. На уровне реализации супертип и его подтипы в иерархии специализации, поддерживают соотношение 1:1.

Подтипы сущностей наследуют все связи, в которых участвует сущность супертипа. В иерархиях специализации с несколькими уровнями супертипа и подтипов, подтип более низкого уровня наследует все атрибуты и связи от всех своих супертипов верхнего уровня. Подтипы могут иметь собственные связи.

Дискриминатор подтипа – это атрибут у сущности супертипа, который определяет, к какому подтипу относится возникновение супертипа. Например, для сотрудников дискриминатором подтипа является тип сотрудника.

Супертип сущности может иметь непересекающиеся или перекрывающиеся подтипы сущности. В авиационном примере служащий может быть пилотом, механиком или бухгалтером. Если бизнес-правило предписывает, что сотрудник не может принадлежать более чем одному подтипу одновременно, тогда работник не может быть пилотом и

механиком одновременно. *Непересекающиеся подтипы*, также известные как *неперекрывающиеся подтипы*, являются подтипами, которые содержат уникальное подмножество набора сущностей супертипа; другими словами, каждый экземпляр сущности супертипа может появляться только в одном из подтипов.

Если бизнес-правило указывает, что сотрудники могут иметь несколько классификаций, супертип СОТРУДНИК может содержать перекрывающиеся подтипы классификации должностей. *Перекрывающиеся подтипы* – это подтипы, которые содержат неуникальные подмножества набора сущностей супертипа; то есть каждый экземпляр сущности супертипа может появляться в нескольких подтипах. Например, в университете человек может быть сотрудником, студентом или и тем, и другим. В свою очередь, сотрудник может быть преподавателем, а также администратором. Поскольку сотрудник также может быть студентом, СТУДЕНТ и СОТРУДНИК являются перекрывающимися подтипами супертипа ЛИЦО, точно так же, как ПРЕПОДАВАТЕЛЬ и АДМИНИСТРАТОР являются перекрывающимися подтипами супертипа СОТРУДНИК.

Реализация непересекающихся подтипов основана на значении атрибута дискриминатора подтипа в супертипе. Однако реализация перекрывающихся подтипов требует использования одного атрибута дискриминатора для каждого подтипа.

Ограничение завершенности определяет, должен ли каждый экземпляр супертипа сущности также быть членом хотя бы одного подтипа. Ограничение полноты может быть частичным или полным. *Частичная завершенность* означает, что не каждое вхождение супертипа является членом подтипа; некоторые вхождения супертипа не могут быть членами какого-либо подтипа. *Полная завершенность* означает, что каждое вхождение супертипа должно быть членом хотя бы одного подтипа.

Можно использовать разные подходы для разработки супертипов и подтипов сущностей. Например, сначала можно определить сущность, а затем определить все подтипы сущности на основе их отличительных свойств. Также можно начать с определения нескольких типов сущностей, а затем извлечь общие характеристики этих сущностей, чтобы создать сущность супертипа более высокого уровня.

Специализация – это нисходящий процесс определения специфических подтипов из супертипа сущности. Специализация основана на группировании уникальных характеристик и связи подтипов. *Обобщение* – это восходящий процесс определения общего супертипа из подтипов. Обобщение основано на группировании общих атрибутов и связей подтипов.

Разработка диаграммы ER влечет за собой обнаружение, возможно сотен типов сущностей и их связей. Начальная ERD содержит несколько сущностей. Когда проект приближается к завершению, ERD будет содержать сотни сущностей и связей, которые заполняют диаграмму до такой степени, что она становится нечитаемой и неэффективной в качестве инструмента коммуникации. В этих случаях можно использовать кластеры объектов, чтобы минимизировать количество объектов, отображаемых в ERD.

Кластер сущностей – это «виртуальный» тип сущности, используемый для представления нескольких сущностей и связей в ERD. Кластер сущностей формируется путем

объединения нескольких взаимосвязанных сущностей в одну абстрактную сущность. Он не является сущностью в окончательном ERD. Это временная сущность, используемая для представления нескольких сущностей и взаимосвязей с целью упрощения ERD и, следовательно, повышения его читабельности.

4.4. Выбор первичных ключей

Первичный ключ наиболее важный атрибут сущности. Он уникальным образом идентифицирует каждый экземпляр сущности. Функция первичного ключа – гарантировать целостность объекта. Кроме того, первичные ключи и внешние ключи работают вместе для реализации связей в реляционной модели. Следовательно, важность правильного выбора первичного ключа напрямую влияет на эффективность реализации БД.

Концепция уникального идентификатора встречается и в реальном мире. Например, номера счетов-фактур для определения конкретных счетов-фактур и номера счетов для идентификации кредитных карт. Эти примеры иллюстрируют естественные идентификаторы или ключи. *Естественный ключ* или *естественный идентификатор* – общепринятый идентификатор реального мира, используемый для уникальной идентификации сущностей реального мира. Естественный ключ знаком пользователям и является частью их повседневной деловой лексики. Если сущность имеет естественный идентификатор, разработчик использует его в качестве первичного ключа.

Первичный ключ – это атрибут или комбинация атрибутов, которые однозначно идентифицируют экземпляры сущности во множестве сущностей. Функция первичного ключа – гарантировать целостность объекта, а не «описывать» объект.

Первичные ключи и внешние ключи используются для реализации связей между сущностями. В реальном мире пользователи идентифицируют сущности на основе свойств, которые они знают. Таблица 7 демонстрирует желательные свойства первичного ключа.

Таблица 7

Свойства первичного ключа

Свойства первичного ключа	Объяснение
Уникальное значение	PK должен однозначно идентифицировать каждый экземпляр сущности. Первичный ключ должен иметь возможность гарантировать уникальные значения. Он не может содержать NULL значения.
Не интеллектуальный	PK не должен иметь встроенного семантического значения, кроме как для уникальной идентификации каждого экземпляра объекта. Атрибут со встроенным семантическим значением, вероятно, лучше использовать в качестве описательной характеристики объекта, чем в качестве идентификатора.
Без изменений со временем	Если атрибут имеет семантическое значение, он может подвергаться обновлениям, поэтому имена не являются хорошими первичными ключами. Если первичный ключ подлежит изменению, значения

Свойства первичного ключа	Объяснение
	внешнего ключа должны быть обновлены, что увеличивает нагрузку на базу данных. PK должен быть постоянным и неизменным.
Предпочтительно один атрибут	Первичный ключ должен иметь минимальное количество возможных атрибутов. Первичные ключи с одним атрибутом желательны, но не обязательны. Первичные ключи с одним атрибутом упрощают реализацию внешних ключей. Наличие первичных ключей с несколькими атрибутами может привести к росту первичных ключей связанных объектов за счет возможного добавления многих атрибутов, что увеличивает нагрузку на базу данных и делает приложения более громоздким.
Предпочтительно числовой	Уникальные значения могут лучше управляться, когда они числовые, потому что база данных может использовать внутренние подпрограммы для реализации атрибута счетного стиля, который автоматически увеличивает значения при добавлении каждой новой строки. Фактически, большинство систем баз данных включает возможность использования специальных конструкций, таких как Autonumber в Microsoft Access, sequence в Oracle или identity в MS SQL Server, для поддержки автоматических атрибутов первичного ключа.
Безопасность-совместимый	Выбранный первичный ключ не должен состоять из каких-либо атрибутов, которые могут рассматриваться как угроза или нарушение безопасности. Например, использование номера паспорта в качестве PK в таблице СОТРУДНИК недопустимо.

Первичный ключ должен использовать минимально возможное количество атрибутов. Составные первичные ключи полезны в двух случаях:

- В качестве идентификаторов ассоциативных сущностей, в которых каждая комбинация первичных ключей допускается только один раз в отношении M:N.
- Как идентификаторы слабых сущностей, в которых слабая сущность имеет сильные идентифицирующие отношения с родителем.

В многих случаях первичный ключ не существует в реальном мире, или существующий естественный ключ не может быть подходящим первичным ключом. В этих случаях стандартная практика – создание суррогатного ключа. *Суррогатный ключ* – первичный ключ, созданный разработчиком для упрощения идентификации экземпляров сущности. Суррогатный ключ не имеет значения с точки зрения пользователя. Одним из практических преимуществ суррогатного ключа является то, что СУБД может генерировать его значения, чтобы всегда предоставлять уникальные значения.

Суррогатные первичные ключи являются общепринятой практикой в современных сложных средах передачи данных. Они особенно полезны, когда нет естественного ключа, когда выбранный ключ-кандидат имеет встроенное семантическое содержимое или слишком длинный. Существует компромисс: если использовать суррогатный ключ, нужно убедиться,

что ключ-кандидат рассматриваемой сущности работает должным образом, используя ограничения «уникальный индекс» и «не ноль».

4.5. Проблемы проектирования базы данных

Разработчики БД часто должны идти на компромиссы, вызванные противоречивыми целями, такими как соблюдение стандартов проектирования, скорость обработки и требования к информации.

- *Стандарты проектирования*. Проект БД должен соответствовать стандартам. Такие стандарты помогают в разработке логических структур, которые сводят к минимуму избыточность данных, тем самым сводя к минимуму вероятность возникновения аномалий данных. Стандарты проектирования позволяют работать с определенными компонентами и оценивать взаимодействие их с определенной точностью.
- *Скорость обработки*. Во многих предприятиях, особенно в тех, которые генерируют большое количество операций, высокая скорость обработки часто является главным приоритетом при проектировании базы данных.
- *Информационные требования*. Поиск своевременной информации может быть в центре внимания при разработке базы данных. Сложные информационные требования могут диктовать преобразования данных, и они могут увеличить количество объектов и атрибутов в проекте.

Проект, который отвечает всем логическим требованиям и правилам проектирования, является важной целью. Тем не менее, если этот совершенный проект не отвечает требованиям к скорости и информации, предъявляемым заказчиком, разработчик не будет выполнять надлежащую работу с точки зрения пользователя. Компромиссы – это факт жизни в реальном мире проектирования баз данных.

Разработчик должен учитывать требования пользователя, такие, как производительность, безопасность, общий доступ и целостность данных, и убедиться, что доступны все параметры обновления, поиска и удаления. Дизайн не имеет особой ценности, если конечный продукт не может выполнить все указанные требования к запросам и отчетам. Даже самый лучший процесс проектирования требует дальнейших изменений в соответствии с эксплуатационными требованиями.

4.5. Итоги

- ERM использует ERD для представления концептуальной базы данных с точки зрения пользователя. Основными компонентами ERM являются сущности, отношения и атрибуты. ERD включает в себя нотации связности и мощности, а также может показывать силу связей, участие в связях (необязательное или обязательное) и степень связей (унарные, бинарные или тернарные).
- Связь описывает классификацию ассоциаций (1:1, 1:M или M:N). Количество элементов выражает конкретное число вхождений сущности. Связности и мощности обычно основаны на бизнес-правилах.

 В ERM отношение M:N действует на концептуальном уровне. При реализации ERM в реляционной базе данных отношение M:N должно быть сопоставлено с набором связей 1:M через ассоциативную сущность.

 Разработчики баз данных, независимо от того, насколько хорошо они могут создавать проекты, соответствующие всем применимым соглашениям о моделировании, часто вынуждены идти на компромиссы. Эти компромиссы необходимы, когда пользователи предъявляют жизненно важные требования к скорости обработки и информации. Разработчики БД должны использовать свое профессиональное суждение, чтобы определить, как и в какой степени соглашения о моделировании могут быть изменены.

 Модель расширенного отношения объекта (EER) добавляет семантику к модели ER через супертипы, подтипы и кластеры объекта. Супертип сущности – общий тип сущностей, который связан с одним или несколькими подтипами.

 Иерархия специализации отображает расположение и связи между супертипами и подтипами сущностей. Наследование означает, что подтип сущности наследует атрибуты и связи супертипа. Подтипы могут быть непересекающимися или перекрывающимися. Дискриминатор подтипа используется для определения того, к какому подтипу относится экземпляр супертипа. Подтипы могут демонстрировать частичную или полную завершенность. Существует два основных подхода к разработке иерархии супертипов и подтипов сущностей: специализация и обобщение.

 Кластер объектов – это «виртуальный» тип объекта, используемый для представления нескольких сущностей и связей в ERD. Кластер сущностей формируется путем объединения нескольких взаимосвязанных сущностей и связей в одну абстрактную сущность.

 Естественные ключи – существуют в реальном мире. Естественные ключи иногда делают хорошие первичные ключи, но не всегда. Первичные ключи должны иметь уникальные значения, они не должны быть интеллектуальными, они не должны изменяться со временем, и они предпочтительно являются числовыми, и состоят из одного атрибута.

 Составные ключи полезны для представления отношений M:N и слабых (строго идентифицирующих) сущностей.

 Суррогатные первичные ключи полезны, когда нет естественного ключа, который создает подходящий первичный ключ, когда первичный ключ является составным первичным ключом с несколькими типами данных или слишком длинный, чтобы его можно было использовать.

Лекция 5. Нормализация таблиц

5.1. Введение

Хороший дизайн базы данных должен соответствовать хорошей структуре таблиц. Чтобы достичь этой цели, нужно разрабатывать хорошие структуры таблиц для контроля избыточности данных, который позволит избежать аномалий данных. Процесс, который дает такие желательные результаты, известен как нормализация.

Наличие хорошего программного обеспечения для реляционных БД недостаточно, чтобы избежать избыточности данных. Если таблицы БД спроектированы так, как файлы в файловой системе, система управления реляционными базами данных (СУРБД) никогда не сможет продемонстрировать свои превосходные возможности обработки данных.

Таблица является основным строительным блоком проектирования БД. Следовательно, структура таблицы представляет большой интерес. В идеале процесс проектирования БД дает хорошие структуры таблиц. Тем не менее, можно создавать плохие структуры таблиц даже при хорошем дизайне БД.

Нормализация – процесс оценки и исправления структур таблиц с целью минимизации избыточности данных, тем самым снижая вероятность аномалий данных. Процесс нормализации включает назначение атрибутов таблицам на основе концепций определения и функциональной зависимости.

Нормализация работает через ряд этапов, называемых нормальными формами. Первые три стадии описаны как первая нормальная форма (1НФ), вторая нормальная форма (2НФ) и третья нормальная форма (3НФ). Со структурной точки зрения 2НФ лучше, чем 1НФ, а 3НФ лучше, чем 2НФ. Для большинства целей в проектировании бизнес-БД 3НФ – достаточный уровень.

Хотя нормализация является очень важным компонентом при проектировании базы данных, не следует полагать, что самый высокий уровень нормализации всегда является наиболее желательным. Как правило, чем выше нормальная форма, тем больше реляционных операций соединения, и соответственно, больше ресурсов необходимо для получения ответа на запрос. Успешный дизайн также должен учитывать спрос пользователя на высокую производительность. Поэтому иногда потребуется денормализовать некоторые части структуры БД. *Денормализация* производит более низкую нормальную форму; то есть 3НФ будет преобразована во 2НФ. Повышение производительности за счет денормализации соблюдается с большой избыточностью данных.

5.2. Важность нормализации

Нормализация обычно используется вместе с моделированием связей сущностей. Разработчики БД используют нормализацию в двух ситуациях. При проектировании новой структуры БД, разработчик БД сначала создает модель данных. После того, как исходный проект завершен, разработчик проводит анализ связей между атрибутами в каждой сущности

и определяет, можно ли улучшить структуру посредством нормализации. Также от разработчиков БД часто требуют изменить существующие структуры данных, которые могут быть в форме текстовых файлов, электронных таблиц или более старых структур БД. Опять же, анализируя связи между атрибутами или полями в структуре данных, разработчик БД может использовать процесс нормализации, чтобы улучшить существующую структуру данных и создать соответствующий проект БД. Независимо от задачи, процесс нормализации одинаков.

Основной целью нормализации является устранение аномалий данных, путем устранения ненужных или нежелательных избыточностей. Для обеспечения заявленных целей проектирования БД, нормализация использует концепцию функциональных зависимостей, чтобы определить, какой атрибут (или набор атрибутов) определяет другие атрибуты.

Чтобы получить лучшее представление о процессе нормализации, можно взять в качестве примера упрощенную отчетность для АСУ предприятия, которая управляет несколькими проектами. Каждый проект имеет свой собственный номер проекта, имя, назначенных сотрудников и так далее. У каждого сотрудника есть номер, имя и должность, например, инженер или системный администратор.

Предприятие выставляет счет своим клиентам, учитывая часы, потраченные на каждый контракт. Почасовая оплата зависит от классификации должностей сотрудника. Периодически создается отчет, который содержит данные для каждого проекта в сводном формате, в виде таблицы 8.

Таблица 8

Примерный отчет

№	Название проекта	№ сотрудни-ка	ФИО сотрудника	Должность сотрудника	Стоимость 1 часа	Кол-во часов	Итого
2541	Ун-Еган	8951	К.С.Коленко	Программист	650,00 ₽	18,5	12 025,00 ₽
		8547	М.Д.Чайка	Разработчик БД	1 200,00 ₽	14,3	17 160,00 ₽
		8745	Ф.Э.Бухаров	Проектный менеджер	930,00 ₽	25,0	23 250,00 ₽
		8685	Е.Н.Пахомов	Системный аналитик	870,00 ₽	20,3	17 661,00 ₽
		8012	М.Г.Иванов	Системный администратор	450,00 ₽	16,0	7 200,00 ₽
Итого за проект							**77 296,00 ₽**
2542	КНС-432	8624	К.А.Большаков	Программист	650,00 ₽	23,1	15 015,00 ₽
		8912	Т.Н.Козин	Водитель	320,00 ₽	15,1	4 832,00 ₽
		8745	Ф.Э.Бухаров	Проектный менеджер	930,00 ₽	38,8	36 084,00 ₽
		8685	О.Е.Шигаева	Разработчик БД	1 200,00 ₽	31,3	37 560,00 ₽
		8475	Э.К.Чуприна	Системный аналитик	870,00 ₽	48,1	41 847,00 ₽
Итого за проект							**135 338,00 ₽**

№	Название проекта	№ сотрудни-ка	ФИО сотрудника	Должность сотрудника	Стоимость 1 часа	Кол-во часов	Итого
2543	ДНС-21	8951	Р.К.Бобылев	Программист	650,00 ₽	47,2	30 680,00 ₽
		8547	М.Д.Чайка	Разработчик БД	1 200,00 ₽	13,7	16 440,00 ₽
		8456	Т.П.Стаин	Проектный менеджер	930,00 ₽	44,2	41 106,00 ₽
		8685	В.К.Сытников	Системный аналитик	870,00 ₽	45,9	39 933,00 ₽
		8012	Б.Ф.Грызунов	Системный администратор	450,00 ₽	40,1	18 045,00 ₽
Итого за проект							**146 204,00 ₽**
2544	КНС-528	8952	К.С.Коленко	Программист	650,00 ₽	20,0	13 000,00 ₽
		8912	Т.Н.Козин	Водитель	320,00 ₽	38,0	12 160,00 ₽
		8745	Ф.Э.Бухаров	Проектный менеджер	930,00 ₽	24,6	22 878,00 ₽
		8685	О.Е.Шигаева	Разработчик БД	1 200,00 ₽	44,4	53 280,00 ₽
		8547	М.Д.Чайка	Разработчик БД	1 200,00 ₽	32,0	38 400,00 ₽
		8475	Э.К.Чуприна	Системный аналитик	870,00 ₽	50,0	43 500,00 ₽
		8012	М.Г.Иванов	Системный администратор	450,00 ₽	42,1	18 945,00 ₽
Итого за проект							**202 163,00 ₽**
ИТОГО							**561 001,00 ₽**

В этом случае разработчику поручается создать БД для поддержки этой отчетности. Сначала нужно сосредоточиться на основных данных, необходимых для составления отчета. Итоги, промежуточные итоги и общие итоги являются вычисляемыми данными. Как только начальный проект завершен, разработчик может принять проектные решения о том, какие вычисляемые данные хранить и какие вычислять при необходимости. В этом случае основные данные показаны на рисунке 15.

Таблица: ОТЧЕТ

База данных: АСУ

ПРО_КОД	ПРО_НАЗВ	СОТР_КОД	СОТР_ФИО	ДЛЖ_НАЗВ	РАБ_СТМ	РАБ_КОЛ
2541	Ун-Еган	8951, 8547, 8745, 8685, 8012	К.С.Коленко, М.Д.Чайка, Ф.Э.Бухаров, Е.Н.Пахомов, М.Г.Иванов	Программист, Разработчик БД, Проектный менеджер, Системный аналитик, Системный администратор	650, 1200, 930, 870, 450	18.5, 14.3, 25.0, 20.3, 16.0
2542	КНС-432	8624, 8912, 8745, 8685, 8475	К.А.Большаков, Т.Н.Козин, Ф.Э.Бухаров, О.Е.Шигаева, Э.К.Чуприна	Программист, Водитель, Проектный менеджер, Разработчик БД, Системный аналитик	650, 320, 930, 1200, 870	23.1, 15.1, 38.8, 31.3, 48.1
2543	ДНС-21	8951, 8547, 8456, 8685, 8012	Р.К.Бобылев, М.Д.Чайка, Т.П.Стаин, В.К.Сытников, Б.Ф.Грызунов	Программист, Разработчик БД, Проектный менеджер, Системный аналитик, Системный администратор	650, 1200, 930, 870, 450	47.2, 13.7, 44.2, 45.9, 40.1
2544	КНС-528	8952, 8912, 8745, 8685, 8547, 8475, 8012	К.С.Коленко, Т.Н.Козин, Ф.Э.Бухаров, О.Е.Шигаева, М.Д.Чайка, Э.К.Чуприна, М.Г.Иванов	Программист, Водитель, Проектный менеджер, Разработчик БД, Разработчик БД, Системный аналитик, Системный администратор	650, 320, 930, 1200, 1200, 870, 450	20.0, 38.0, 24.6, 44.4, 32.0, 50.0, 42.1

Рис. 15. Основные данные для отчетности

Базовые данные на рисунке 15 организованы вокруг проектов так же, как был организован отчет, при этом каждый проект имеет одну строку для представления данных,

связанных с этим проектом. Исходные данные показывают, что проекту назначено несколько сотрудников. Обратите внимание, что данные на рисунке 15 являются *ненормализованными* данными, что отражается в существовании нескольких многозначных элементов данных (СОТР_КОД, СОТР_ФИО, ДЛЖ_НАЗВ, РАБ_СТМ, РАБ_КОЛ).

Структура данных, изображенная на рисунке 15, не соответствует требованиям к реляционным таблицам, и поэтому не подходит для правильной обработки данных. Она имеет следующие недостатки:

- Структура данных допускает несоответствие данных. Например, значение ДЛЖ_НАЗВ «Разработчик БД» может быть введен как «Разр.БД» или даже «РБД». Структура позволила бы О.Е.Шигаевой и М.Д.Чайке в проекте КНС-528 взимать разные ставки, даже если у них одинаковая классификация должностей.
- Структура данных содержит несколько многозначных атрибутов, которые очень усложняют задачи управления данными. Поскольку все сотрудники, работающие над проектом, находятся в одной ячейке, трудно идентифицировать каждого сотрудника индивидуально, а для базы данных ответить на такие вопросы, как «Сколько сотрудников работает над проектом ДНС-21?».
- Данные о сотрудниках в таблице избыточны, поскольку сотрудники могут работать над несколькими проектами. Добавление, обновление и удаление данных, вероятно, будет очень громоздким с использованием этой структуры. Например, изменение классификации должностей для Ф.Э.Бухарова потребует обновления как минимум трех строк.

Очевидно, что эта структура приводит к несоответствиям данных. Отчет может давать разные результаты в зависимости от того, какая аномалия данных произошла. Аномалии отчетности вызывают множество проблем у менеджеров – не могут быть исправлены с помощью прикладного программирования.

Проблемы целостности, избыточности и несогласованности данных должны быть решены во время проектирования БД.

5.3. Процесс нормализации

Целью нормализации является обеспечение того, чтобы каждая таблица соответствовала концепции правильно сформированных связей. Таблицы после нормализации должны иметь следующие характеристики:

- Каждая таблица представляет отдельную сущность. Например, таблица ДИСЦИПЛИНА будет содержать только данные, которые непосредственно относятся к дисциплинам. Аналогично, таблица СТУДЕНТ будет содержать только данные об учениках.
- Каждое пересечение строки/столбца содержит только одно значение, а не группу значений.

- Ни один элемент данных не будет излишне сохранен в более чем одной таблице. Данные обновляются только в одном месте.
- Все неключевые атрибуты в таблице зависят только от первичного ключа. Данные уникально идентифицируются по значению первичного ключа.
- Каждая таблица не имеет аномалий вставки, обновления или удаления, что обеспечивает целостность и согласованность данных.

Для достижения этих целей процесс нормализации проводится через шаги, которые приводят к последовательно более высоким нормальным формам. Наиболее распространенные нормальные формы и их основные характеристики перечислены в таблице 9.

Таблица 9

Нормальные формы

Нормальная форма	Описание
Первая нормальная форма (1НФ)	Таблица находится в 1НФ, если все ее атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.
Вторая нормальная форма (2НФ)	Таблица находится во 2НФ, если она находится в 1НФ и каждый неключевой атрибут зависит от Первичного Ключа (ПК).
Третья нормальная форма (3НФ)	Таблица находится в 3НФ, когда находится во 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.
Нормальная форма Бойса-Кодда (НФБК)	Таблица находится в НФБК, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта.
Четвертая нормальная форма (4НФ)	Таблица находится в 4НФ, если она находится в НФБК и все нетривиальные многозначные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей.
Пятая нормальная форма (5НФ)	Таблица находится в 5НФ, если она находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами.

В обсуждении нормализации концепция ключей является основной темой. Ключ-кандидат является минимальным (неприводимым) суперключем. Первичный ключ – ключ-кандидат, выбранный в качестве основного, используемого для идентификации строк в таблице. Хотя нормализация обычно представляется с точки зрения ключей-кандидатов, это обсуждение предполагается для простоты, что каждая таблица имеет только один ключ-кандидат; следовательно, этот ключ-кандидат является первичным ключом.

С точки зрения разработчика данных, цель нормализации – обеспечить, чтобы все таблицы были как минимум в 3НФ. Существуют нормальные формы даже более высокого уровня. Однако нормальные формы, такие как пятая нормальная форма (5НФ) и нормальная форма ключа домена (КДНФ), вряд ли встретятся на практике, и представляют в основном теоретический интерес. Такие более высокие нормальные формы обычно увеличивают количество таблиц и связей, что снижает производительность, не добавляя никакой ценности в устранении избыточности данных. Некоторые очень специализированные приложения, такие как статистические исследования, могут требовать нормализации за пределами 4НФ, но эти приложения выходят за рамки большинства бизнес-операций.

Прежде чем обрисовать в общих чертах процесс нормализации, рекомендуется рассмотреть концепции определения и функциональной зависимости. В таблице 10 обобщены основные концепции.

Таблица 10

Концепция функциональной зависимости

Концепция	Определение
Функциональная зависимость	Атрибут ***B*** полностью функционально зависит от атрибута ***A***, если каждое значение ***A*** определяет одно и только одно значение ***B***. Пример: ПРО_КОД → ПРО_НАЗВ (читается как ПРО_КОД функционально определяет ПРО_НАЗВ) В этом случае атрибут ПРО_КОД является определителем, а атрибут ПРО_НАЗВ зависимым.
Функциональная зависимость (обобщенное определение)	Атрибут ***A*** определяет атрибут ***B*** (то есть ***B*** функционально зависит от ***A***), если все (обобщенное определение) строки в таблице, которые совпадают по значению для атрибута ***A***, также совпадают по значению для атрибута ***B***.
Полностью функциональная зависимость (составной ключ)	Если атрибут ***B*** функционально зависит от составного ключа ***A***, но не от какого-либо подмножества этого составного ключа, атрибут ***B*** полностью функционально зависит от ***A***.

Крайне важно понимать эти концепции, поскольку они используются для получения набора функциональных зависимостей для данной таблицы. Процесс нормализации работает по одной таблице за раз, выявляя зависимости и нормализуя ее. Нормализация начинается с определения зависимостей данной таблицы и постепенного разбиения ее на набор новых таблиц на основе зависимостей.

Два типа функциональных зависимостей представляют особый интерес при нормализации – частичные и транзитивные зависимости. *Частичная зависимость* существует, когда существует функциональная зависимость, в которой определитель является только частью первичного ключа. Например, если ***(A, B) → (C, D)***, ***B → C*** и ***(A, B)*** является первичным ключом, то функциональная зависимость ***B → C*** является частичной

зависимостью, поскольку требуется только частью первичного ключа (***B***) определить значение ***C***. Частичные зависимости, как правило, просты и их легко идентифицировать.

Транзитивная зависимость существует, когда существуют функциональные зависимости, такие как $X \rightarrow Y$, $Y \rightarrow Z$ и ***X*** является первичным ключом. В этом случае зависимость $X \rightarrow Z$ является транзитивной зависимостью, поскольку ***X*** определяет значение ***Z*** через ***Y***. В отличие от частичных зависимостей, транзитивные зависимости труднее идентифицировать. Но существует эффективный способ выявления транзитивных зависимостей: они возникают только в том случае, если среди неключевых атрибутов существует функциональная зависимость. В предыдущем примере фактической транзитивной зависимостью является $X \rightarrow Z$. Однако зависимость $Y \rightarrow Z$ сигнализирует о существовании транзитивной зависимости. Для решения проблем, связанных с транзитивными зависимостями, изменения в структуре таблицы делаются на основе функциональной зависимости, которая является признаком существования транзитивной зависимости.

Реляционная модель рассматривает данные как часть таблицы или коллекции таблиц, в которых должны быть идентифицированы все ключевые значения. На рисунке 15 содержатся так называемые повторяющиеся группы. *Повторяющаяся группа* получает свое имя из того факта, что для любого вхождения атрибута ключа может существовать группа из нескольких записей одного или нескольких типов. На рисунке 15 каждое вхождение одного кода проекта (ПРО_КОД) может ссылаться на группу связанных данных в столбце кода сотрудника, имя сотрудника, классификация работы и оплата за час. Например, проект ДНС-21 (ПРО_КОД = 2543) содержит пять значений для каждого из этих атрибутов.

Нормализация структуры таблицы уменьшит избыточность данных. Если повторяющиеся группы существуют, их необходимо устранить, убедившись, что каждая строка определяет один экземпляр сущности и что на каждом пересечении строки-столбца находится только одно значение. Зависимости должны быть идентифицированы для диагностики нормальной формы. Идентификация нормальной формы позволяет узнать место нахождения в процессе нормализации. Нормализация начинается с простой трехшаговой процедуры.

Шаг 1. Устранение повторяющихся групп. Нужно начать с представления данных в табличном формате, где каждая ячейка имеет одно значение и нет повторяющихся групп. Чтобы исключить повторяющиеся группы, нужно изменить таблицу с фокуса проекта на фокус назначения, то есть создать отдельные строки для каждого сотрудника, назначенного каждому проекту, преобразовав многозначные атрибуты в однозначные атрибуты. Это изменение преобразует таблицу на рисунке 15 в 1НФ, как показано на рисунке 16.

Таблица: ОТЧЕТ База данных: АСУ

ПРО_КОД	ПРО_НАЗВ	СОТР_КОД	СОТР_ФИО	ДЛЖ_НАЗВ	РАБ_СТМ	РАБ_КОЛ
2541	Ун-Еган	8951	К.С.Коленко	Программист	650,00 ₽	18,5
2541	Ун-Еган	8547	М.Д.Чайка	Разработчик БД	1 200,00 ₽	14,3
2541	Ун-Еган	8745	Ф.Э.Бухаров	Проектный менеджер	930,00 ₽	25,0
2541	Ун-Еган	8685	Е.Н.Пахомов	Системный аналитик	870,00 ₽	20,3
2541	Ун-Еган	8012	М.Г.Иванов	Системный администратор	450,00 ₽	16,0
2542	КНС-432	8624	К.А.Большаков	Программист	650,00 ₽	23,1
2542	КНС-432	8912	Т.Н.Козин	Водитель	320,00 ₽	15,1
2542	КНС-432	8745	Ф.Э.Бухаров	Проектный менеджер	930,00 ₽	38,8
2542	КНС-432	8685	О.Е.Шигаева	Разработчик БД	1 200,00 ₽	31,3
2542	КНС-432	8475	Э.К.Чуприна	Системный аналитик	870,00 ₽	48,1
2543	ДНС-21	8951	Р.К.Бобылев	Программист	650,00 ₽	47,2
2543	ДНС-21	8547	М.Д.Чайка	Разработчик БД	1 200,00 ₽	13,7
2543	ДНС-21	8456	Т.П.Стаин	Проектный менеджер	930,00 ₽	44,2
2543	ДНС-21	8685	В.К.Сытников	Системный аналитик	870,00 ₽	45,9
2543	ДНС-21	8012	Б.Ф.Грызунов	Системный администратор	450,00 ₽	40,1
2544	КНС-528	8952	К.С.Коленко	Программист	650,00 ₽	20,0
2544	КНС-528	8912	Т.Н.Козин	Водитель	320,00 ₽	38,0
2544	КНС-528	8745	Ф.Э.Бухаров	Проектный менеджер	930,00 ₽	24,6
2544	КНС-528	8685	О.Е.Шигаева	Разработчик БД	1 200,00 ₽	44,4
2544	КНС-528	8547	М.Д.Чайка	Разработчик БД	1 200,00 ₽	32,0
2544	КНС-528	8475	Э.К.Чуприна	Системный аналитик	870,00 ₽	50,0
2544	КНС-528	8012	М.Г.Иванов	Системный администратор	450,00 ₽	42,1

Рис. 16. Таблица в первой нормальной форме (1НФ)

Шаг 2: Определение первичного ключа. Если посмотреть на рисунок 16, заметно, что ПРО_КОД не является адекватным первичным ключом, потому что номер проекта не уникально идентифицирует каждую строку. Например, значение 2543 ПРО_КОД может идентифицировать любую из пяти строк, содержащих сотрудников, работающих над проектом ДНС-21. Чтобы поддерживать правильный первичный ключ, который будет однозначно идентифицировать любое значение атрибута, новый ключ должен состоять из комбинации ПРО_КОД и СОТР_КОД. Например, используя данные, показанные на рисунке 16, если известно, что ПРО_КОД = 2543 и СОТР_КОД = 8547, значения для атрибутов ПРО_НАЗВ, СОТР_ФИО, ДЛЖ_НАЗВ, РАБ_СТМ и РАБ_КОЛ должны быть «ДНС-21», «М.Д.Чайка», «Разработчик БД», 1200 р. и 13.7 соответственно.

Шаг 3. Определение всех зависимостей. Идентификация ПК на шаге 2 означает, что уже определена следующая зависимость:

ПРО_КОД, СОТР_КОД → ПРО_НАЗВ, СОТР_ФИО, ДЛЖ_НАЗВ, РАБ_СТМ, РАБ_КОЛ.

То есть все значения ПРО_НАЗВ, СОТР_ФИО, ДЛЖ_НАЗВ, РАБ_СТМ и РАБ_КОЛ зависят от сочетания ПРО_КОД и СОТР_КОД и определяются им.

Достижение 1НФ недостаточно для устранения всех аномалий, которые существовали в исходной структуре. 1НФ работает с повторяющимися группами и гарантирует, что таблица

соответствует требованиям для реляционной таблицы. Однако аномалии сохраняются. Например, каждый раз, когда другой сотрудник назначается проекту, некоторые записи данных (такие как ПРО_НАЗВ, СОТР_ФИО, и РАБ_СТМ) повторяются без необходимости. В идеале, запись номера сотрудника должна быть достаточной для идентификации, например: М.Д.Чайка, описание ее работы и ее почасовой оплаты. Поскольку код 8547 идентифицирует только одного человека, его атрибуты (имя, должность и т.д.) не нужно вводить каждый раз при добавлении или обновлении задания.

Аномалии, которые остаются, существуют, потому что есть дополнительные зависимости. Например, код проекта определяет название проекта. Другими словами, имя проекта зависит от кода проекта. Можно записать эту зависимость как:

ПРО_КОД → ПРО_НАЗВ.

Кроме того, если известен код сотрудника, можно определить имя его, должность и плату за час. Следовательно, можно определить зависимость, показанную ниже:

СОТР_КОД → СОТР_ФИО, ДЛЖ_НАЗВ, РАБ_СТМ.

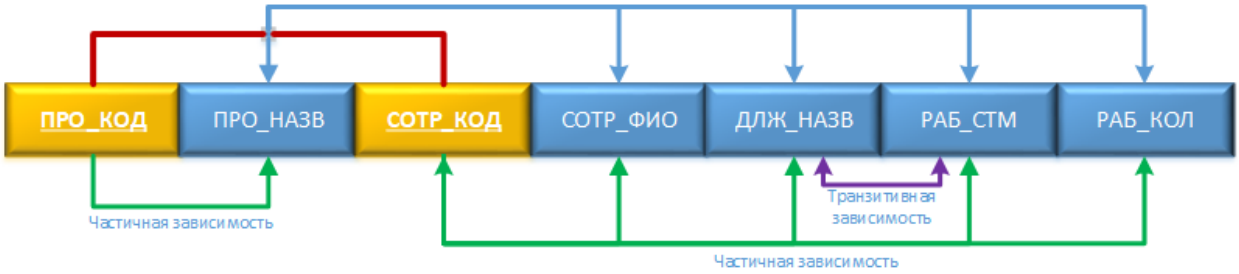
Проще говоря, у сотрудника есть следующие атрибуты: номер, имя, должность и плата за час.

Некоторые зависимости более очевидны, чем другие. Например, бизнес-правило «Каждая должность имеет определенную плату в час» подразумевает, что плата за час зависит от должности. Однако, обсуждения основаны на точке зрения процесса нормализации и служат для демонстрации того, как нормализация может также помочь в проверке бизнес-правил.

Изучив данные на рисунке 16, видно, что знание должностей означает знание платы за час для них. (Например, все системные аналитики или программисты имеют одинаковую плату за час независимо от проекта или сотрудника). Другими словами, плата за час зависит от классификации должностей, а не от сотрудника. Следовательно, можно определить последнюю зависимость:

ДЛЖ_НАЗВ → РАБ_СТМ.

Однако эта зависимость существует между двумя неключевыми атрибутами; следовательно, это говорит о том, что существует транзитивная зависимость. Зависимости показаны на рисунке 17. Поскольку такая диаграмма отображает все зависимости, найденные в данной структуре таблицы, она называется *диаграммой зависимостей*. Диаграммы зависимостей очень полезны для получения всех связей между атрибутами таблицы, и их использование снижает вероятность пропуска важной зависимости.



1НФ (ПРО_КОД, ПРО_НАЗВ, СОТР_КОД, СОТР_ФИО, ДЛЖ_НАЗВ, РАБ_СТМ, РАБ_КОЛ)

Частичные зависимости:
(ПРО_КОД → ПРО_НАЗВ)
(СОТР_КОД → СОТР_ФИО, ДЛЖ_НАЗВ, РАБ_СТМ)

Транзитивные зависимости:
(ДЛЖ_НАЗВ → РАБ_СТМ)

Рис. 17. 1НФ Диаграмма зависимости

Изучая рисунок 17, нужно обратить внимание на следующие особенности диаграммы:

1. Атрибуты первичного ключа выделены жирным шрифтом, подчеркнуты и имеют другой цвет.
2. Стрелки над атрибутами указывают на все желаемые зависимости, то есть зависимости, основанные на первичном ключе. В этом случае надо обратить внимание на то, что атрибуты сущности зависят от комбинации ПРО_КОД и СОТР_КОД.
3. Стрелки под диаграммой указывают на менее желательные зависимости. Существует два типа таких зависимостей:
   a) *Частичные зависимости*. Нужно знать только ПРО_КОД, чтобы определить ПРО_НАЗВ; то есть ПРО_НАЗВ зависит только от части первичного ключа. Кроме того, нужно знать только СОТР_КОД, чтобы найти СОТР_ФИО, ДЛЖ_НАЗВ и РАБ_СТМ. Зависимость, основанная только на части составного первичного ключа, является частичной зависимостью.
   b) *Транзитивные зависимости*. РАБ_СТМ зависит от ДЛЖ_НАЗВ. Поскольку ни РАБ_СТМ, ни ДЛЖ_НАЗВ не являются ключевым атрибутом, условие указывает на транзитивную зависимость. Другими словами, транзитивная зависимость существует, когда функциональная зависимость существует только среди неключевых атрибутов. Из-за транзитивных зависимостей получаются аномалии данных.

Термин *первая нормальная форма (1НФ)* описывает табличный формат, который соответствует определению реляционной таблицы, в которой:

- Все ключевые атрибуты определены.
- В таблице нет повторяющихся групп. Другими словами, каждое пересечение строки/столбца содержит одно и только одно значение.
- Все атрибуты зависят от первичного ключа.

Рисунок 17 включает в себя реляционную схему для таблицы в 1НФ и текстовую запись для каждой идентифицированной зависимости.

Все реляционные таблицы удовлетворяют требованиям 1НФ. Хотя данные 1НФ на рисунке 16 являются улучшением по сравнению с ненормализованными данными на рисунке 15, они по-прежнему имеют нежелательные проблемы, которые вызывают аномалии данных.

Хотя частичные зависимости иногда используются по соображениям производительности, их следует использовать с осторожностью, поскольку таблица, содержащая частичные зависимости, все еще подвержена избыточности данных и, следовательно, разным аномалиям. В примере есть следующие аномалии:

a) *Аномалии обновления*. Модификация ДЛЖ_НАЗВ для сотрудника М.Д.Чайки требует обновления многих записей; в противном случае это приведет к несоответствиям данных.
b) *Аномалии изменения*. Добавление нового сотрудника требует, чтобы сотрудник был назначен для проекта и, следовательно, вводил дубликаты информации о проекте. Если сотрудник еще не назначен проекту, для завершения ввода данных сотрудника необходимо создать несуществующий проект.
c) *Аномалии удаления*. Предположим, что только один сотрудник связан с данным проектом. Если этот сотрудник удален, информация о проекте также будет удалена.

Избыточность данных возникает потому, что каждая запись строки требует дублирования данных. В результате могут появиться разные версии имени сотрудника, должности или почасовой оплаты. Например, имя сотрудника для СОТР_КОД = 8547 может быть введено как «М.Д.Чайка», «Чайка М.Д.» или «Мария Дмитриевна Чайка». Название проекта также может быть введено правильно как «ДНС-21» или с ошибкой как «ДНС21». Такие аномалии данных нарушают правила целостности и согласованности реляционной базы данных.

Преобразование в 2НФ происходит только тогда, когда 1НФ имеет составной первичный ключ. Если 1НФ имеет первичный ключ с одним атрибутом, то таблица автоматически становится 2НФ. Преобразование 1НФ в 2НФ довольно простое. Начиная с формата 1НФ, показанного на рисунке 17, нужно делать следующие шаги:

Шаг 1. Создание новых таблиц для устранения частичных зависимостей. Для каждого компонента первичного ключа, который действует как определитель частичной зависимости, нужно создать новую таблицу с копией этого компонента в качестве первичного ключа. Хотя эти компоненты размещены в новых таблицах, важно, чтобы они также оставались в исходной таблице. Определители должны оставаться в исходной таблице, поскольку они будут внешними ключами для отношений, необходимых для связи этих новых таблиц с исходной таблицей. Чтобы построить пересмотренную диаграмму зависимостей, нужно записать каждый ключевой компонент в отдельной строке, а затем исходный (составной) ключ в последней строке. Например:

ПРО_КОД

СОТР_КОД

ПРО_КОД СОТР_КОД.

Каждый компонент станет ключом в новой таблице. Другими словами, исходная таблица теперь разделена на три таблицы (ПРОЕКТ, СОТРУДНИК и НАЗНАЧЕНИЕ).

Шаг 2. Переназначение соответствующих зависимых атрибутов. Можно использовать рисунок 17 для определения атрибутов, которые входят в частичные зависимости. Зависимости для исходных ключевых компонентов находятся путем изучения стрелок под диаграммой зависимостей, показанной на рисунке 17. Атрибуты, которые входят в частичные зависимости, удаляются из исходной таблицы и помещаются в новую таблицу с определителем зависимости. Любые атрибуты, которые не входят в частичные зависимости, останутся в исходной таблице. Другими словами, три таблицы, полученные в результате преобразования в 2НФ, имеют соответствующие имена (ПРОЕКТ, СОТРУДНИК и НАЗНАЧЕНИЕ) и описываются следующими реляционными схемами:

ПРОЕКТ (ПРО_КОД, ПРО_НАЗВ)

СОТРУДНИК (**СОТР_КОД**, СОТР_ФИО, ДЛЖ_НАЗВ, РАБ_СТМ)

НАЗНАЧЕНИЕ (**ПРО_КОД**, **СОТР_КОД**, НАЗ_ЧАСЫ).

Поскольку количество часов, потраченных на каждый проект каждым сотрудником, зависит как от ПРО_КОД, так и от СОТР_КОД, нужно оставить этот атрибут в таблице НАЗНАЧЕНИЕ как НАЗ_ЧАСЫ. Таблица НАЗНАЧЕНИЕ содержит составной первичный ключ, состоящий из атрибутов ПРО_КОД, СОТР_КОД. Оставив детерминанты в исходной таблице, а также сделав их первичными ключами новых таблиц, была создана связь первичный ключ/внешний ключ. Например, в таблице СОТРУДНИК СОТР_КОД является первичным ключом. В таблице НАЗНАЧЕНИЕ СОТР_КОД является частью составного первичного ключа (ПРО_КОД, СОТР_КОД) и является внешним ключом, связывающим таблицу СОТРУДНИК с таблицей НАЗНАЧЕНИЕ.

Результаты шагов 1 и 2 показаны на рисунке 18. На этом этапе большинство аномалий, обсужденных ранее, были устранены. Например, если необходимо добавить, изменить или удалить проект, нужно перейти только к таблице ПРОЕКТ и внести изменения только в одну строку.

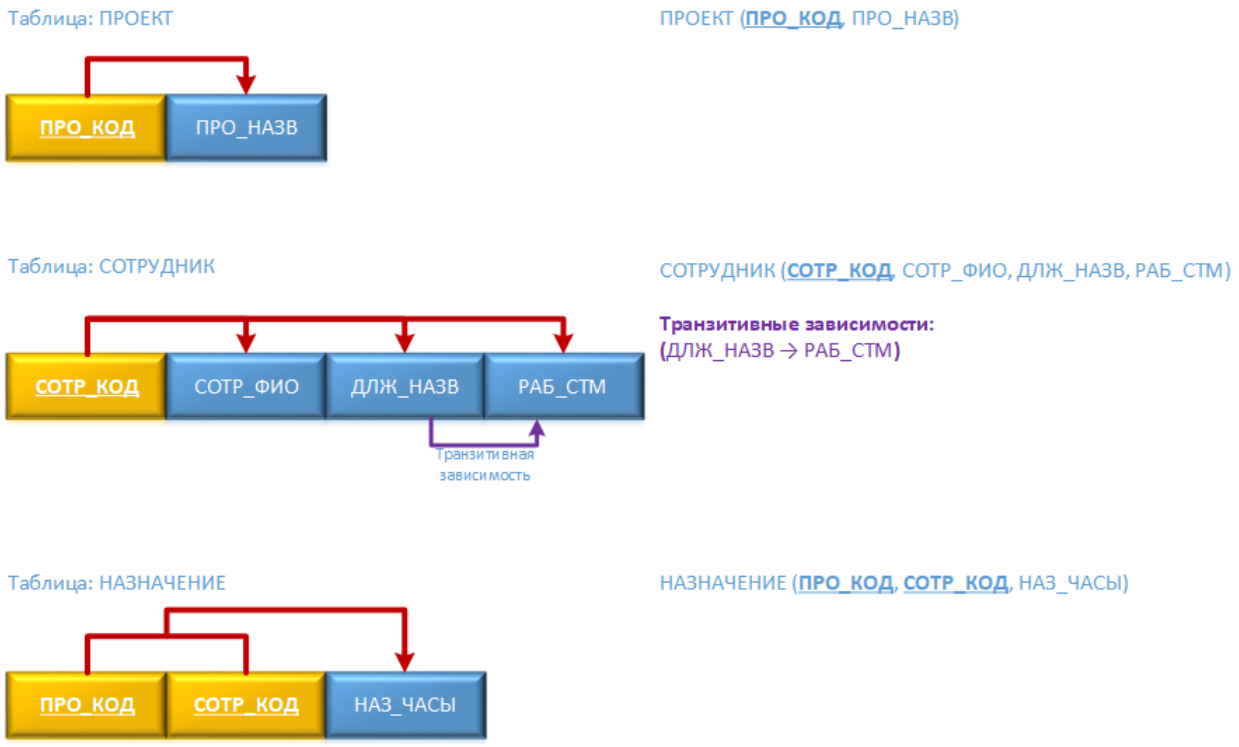


Рис. 18. Вторая нормальная форма (2НФ)

Поскольку частичная зависимость может существовать только в том случае, если первичный ключ таблицы состоит из нескольких атрибутов, таблица, которая находится в 1НФ и первичный ключ состоит только из одного атрибута, автоматически попадает в 2НФ.

Таблица находится во второй нормальной форме (2НФ), когда:

- Она находится в 1НФ;
- Не содержит частичных зависимостей; то есть ни один атрибут не зависит только от части первичного ключа.

Рисунок 18 по-прежнему показывает транзитивную зависимость, которая может генерировать аномалии. Например, если оплата за час изменяется для классификации должностей, проводимой многими сотрудниками, это изменение должно быть сделано для каждого из этих сотрудников. Если пропустить обновления некоторых записей о сотрудниках, на которые влияет изменение оплаты за час, то, для разных сотрудников с одинаковой должностью будут разные почасовые платежи.

Аномалии данных, созданные организацией базы данных, показанной на рисунке 18, легко устранить, выполнив следующие два шага:

Шаг 1. Создание новых таблиц для устранения транзитивных зависимостей. Для каждой транзитивной зависимости нужно записать копию ее детерминанта в качестве первичного ключа для новой таблицы. *Детерминант* – это любой атрибут, значение которого определяет другие значения в строке. Если существуют три разные транзитивные зависимости, то будут три разных детерминанта. Как и при преобразовании в 2НФ, важно, чтобы детерминант оставался в исходной таблице, чтобы служить внешним ключом. На

рисунке 18 показана только одна таблица, которая содержит транзитивную зависимость. Нужно записать детерминант для этой транзитивной зависимости как:

ДЛЖ_НАЗВ.

Шаг 2. Переназначение соответствующих зависимых атрибутов. Используя рисунок 18, нужно определить атрибуты, которые зависят от каждого детерминанта, определенного на шаге 1. Поместить зависимые атрибуты в новые таблицы с их детерминантами, и удалить их из исходных таблиц. В этом примере нужно исключить РАБ_СТМ из таблицы СОТРУДНИК, показанной на рисунке 18, и оставить определение следующим образом:

СОТР_КОД → СОТР_ФИО, ДЛЖ_НАЗВ.

На рисунке 19 показана диаграмма зависимостей, содержащая все таблицы, которые определены в шагах 1 и 2. Для новой таблицы подходящее имя: ДОЛЖНОСТЬ. Каждая таблица имеет детерминант, и ни одна таблица не содержит недопустимых зависимостей.

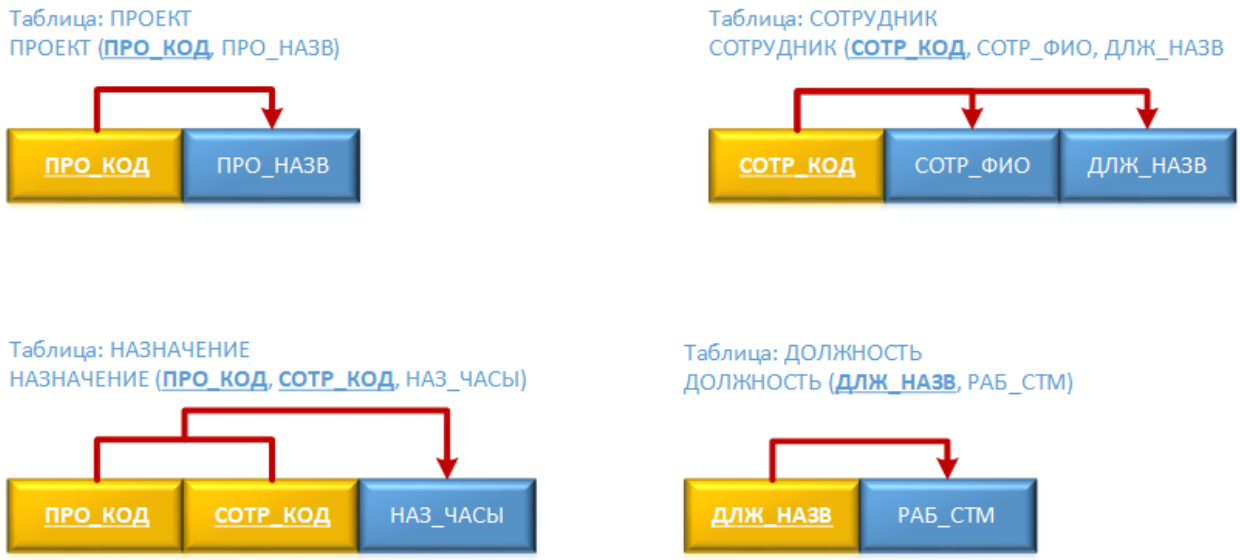


Рис. 19. Третья нормальная форма

Другими словами, после завершения преобразования 3НФ, база данных будет содержать четыре таблицы:

ПРОЕКТ (**ПРО_КОД**, ПРО_НАЗВ)

СОТРУДНИК (**СОТР_КОД**, СОТР_ФИО, ДЛЖ_НАЗВ)

ДОЛЖНОСТЬ (**ДЛЖ_НАЗВ**, РАБ_СТМ)

НАЗНАЧЕНИЕ (**ПРО_КОД**, **СОТР_КОД**, НАЗ_ЧАСЫ).

Данное преобразование устранило транзитивную зависимость исходной таблицы СОТРУДНИК. Таким образом, таблицы находятся в третьей нормальной форме (3НФ).

Таблица находится в третьей нормальной форме (3НФ), когда:

- Она находится в 2НФ;
- Не содержит транзитивных зависимостей.

Есть сходство между решением проблем 2НФ и 3НФ. Чтобы преобразовать таблицу из 1НФ в 2НФ, необходимо удалить частичные зависимости. Чтобы преобразовать таблицу из 2НФ в 3НФ, необходимо удалить транзитивные зависимости. Независимо от того, является ли «проблемная» зависимость частичной или транзитивной, решение остается одним и тем же: создать новую таблицу для каждой проблемной зависимости. Детерминант зависимости проблемы остается в исходной таблице и помещается в качестве первичного ключа новой таблицы. Зависимые элементы удаляются из исходной таблицы и помещаются в новую таблицу как неключевые атрибуты.

Прежде чем перейти к 3НФ, обязательно, чтобы 2НФ была достигнута. Необходимо разрешить частичные зависимости перед разрешением транзитивных зависимостей. Каждая таблица должна иметь только один ключ-кандидат, который является первичным ключом.

5.4. Улучшение проекта

Когда структуры таблиц были очищены для устранения проблемных частичных и транзитивных зависимостей, можно сосредоточиться на улучшении проекта БД.

Проверка первичных ключей. Каждый раз, когда в таблицу СОТРУДНИК вводится новый сотрудник, необходимо ввести значение ДЛЖ_НАЗВ. К сожалению, слишком легко совершать ошибки ввода данных, которые приводят к нарушениям ссылочной целостности. Например, ввод «Сисадмин» вместо «Системный администратор» для атрибута ДЛЖ_НАЗВ в таблице СОТРУДНИК вызовет такое нарушение. Поэтому, было бы лучше добавить атрибут ДЛЖ_КОД для создания уникального идентификатора. Добавление атрибута ДЛЖ_КОД создает следующую зависимость:

ДЛЖ_КОД → ДЛЖ_НАЗВ, РАБ_СТМ.

Наличие ДЛЖ_КОД значительно снижает вероятность нарушений ссылочной целостности. Новая таблица ДОЛЖНОСТЬ теперь имеет два возможных ключа – ДЛЖ_КОД и ДЛЖ_НАЗВ. В этом случае ДЛЖ_КОД является выбранным первичным ключом, а также суррогатным ключом. Суррогатный ключ – искусственный ПК, представленный разработчиком с целью упрощения. Суррогатные ключи обычно являются числовыми, они часто генерируются СУБД автоматически, они свободны от семантического содержимого (они не имеют особого значения) и обычно скрыты от конечных пользователей.

Проверка соглашений об именах. Нужно придерживаться соглашений об именах. Поэтому РАБ_СТМ будет изменен на ДЛЖ_СТМ, чтобы указать его связь с таблицей ДОЛЖНОСТЬ. Кроме того, что РАБ_КОЛ был изменен на НАЗ_ЧАСЫ при преобразовании из 1НФ в 2НФ. Это изменение позволяет связать отработанные часы с таблицей НАЗНАЧЕНИЕ.

Проверка атомарности атрибута. Как правило, рекомендуется обращать внимание на требование атомарности. *Атомарный атрибут* – атрибут, который не может быть далее подразделен. Говорят, что такой атрибут имеет *атомарность*. Ясно, что использование СОТР_ФИО в таблице СОТРУДНИК не является атомарным, поскольку СОТР_ФИО можно разложить на фамилию, имя и отчество. Высшая степень атомарности позволяет выполнять

гибкость запросов. Например, если СОТР_ФИО разделить на СОТР_ФАМ, СОТР_ИМЯ и СОТР_ОТЧ, можно легко создавать списки телефонов, сортируя фамилии, имена и отчества. Такая задача была бы очень сложной, если бы компоненты имени были внутри одного атрибута. Обычно разработчики предпочитают использовать простые однозначные атрибуты, как указано в бизнес-правилах и требованиях к обработке.

Определение новых атрибутов. Если бы таблица СОТРУДНИК использовалась в реальной среде, необходимо было бы добавить несколько других атрибутов. Например, желательны выплаты зарплаты с начала года, выплаты по пенсионному фонду и выплаты по профсоюзу. Атрибут даты найма сотрудника (СОТР_НДТ) можно использовать для отслеживания продолжительности работы сотрудника, и он может служить основой для присуждения премий работникам, работающим на постоянной основе, и для других мер. Тот же принцип должен применяться ко всем другим таблицам в проекте.

Определение новых связей. Согласно исходному отчету, пользователи должны отслеживать, какой сотрудник выступает в роли руководителя каждого проекта. Это может быть реализовано как связь между таблицами СОТРУДНИК и ПРОЕКТ. Из исходного отчета видно, что у каждого проекта есть только один менеджер. Таким образом, способность системы предоставлять подробную информацию о каждом менеджере проекта обеспечивается использованием СОТР_КОД в качестве внешнего ключа в ПРОЕКТ. Это действие гарантирует, что можно получить доступ к деталям данных руководителя каждого ПРОЕКТА без ненужного и нежелательного дублирования данных. Разработчик должен позаботиться о том, чтобы поместить правильные атрибуты в правильные таблицы, используя принципы нормализации.

Уточнение первичных ключей для гранулярности. Гранулярность – это уровень детализации, представленный значениями, хранящимися в строке таблицы. Данные, хранящиеся на самом низком уровне детализации, называются атомарными данными, как объяснялось ранее. На рисунке 19 таблица НАЗНАЧЕНИЕ в 3НФ использует атрибут НАЗ_ЧАСЫ для представления часов, отработанных данным сотрудником в данном проекте. Однако записаны ли эти значения на самом низком уровне детализации? Другими словами, представляет ли НАЗ_ЧАСЫ почасовой итог, дневной итог, недельный итог, месячный итог или годовой итог? Очевидно, что НАЗ_ЧАСЫ требует более тщательного определения. В этом случае соответствующий вопрос будет следующим: для какого периода времени – час, день, неделя, месяц и т.д. – нужно записать данные НАЗ_ЧАСЫ?

Например, комбинация СОТР_КОД и ПРО_КОД является приемлемым (составным) первичным ключом в таблице НАЗНАЧЕНИЕ. Этот первичный ключ полезен для представления только общего количества часов, которые сотрудник работал над проектом с момента его запуска. Использование суррогатного первичного ключа НАЗ_КОД, обеспечивает меньшую степень детализации и большую гибкость. Например, комбинация СОТР_КОД и ПРО_КОД используется в качестве первичного ключа, а затем сотрудник делает две записи «отработанные часы» в таблице НАЗНАЧЕНИЕ. Это действие нарушает требование целостности сущности. Даже если добавить НАЗ_ДТ как часть составного ПК,

нарушение целостности сущности все равно будет сгенерировано, если какой-либо сотрудник сделает две или более записи для одного и того же проекта в один и тот же день. (Возможно, сотрудник работал над проектом несколько часов утром, а затем снова работал над ним позже в тот же день). При таком же вводе данных не возникнет проблем, если НАЗ_КОД используется в качестве первичного ключа.

В идеальном проекте БД уровень желаемой степени детализации будет определяться во время концептуального проектирования или во время сбора требований. Однако, многие проекты БД включают в себя уточнение существующих требований к данным, что вызывает изменения в проекте. В реальной среде изменяющиеся требования к гранулярности могут диктовать изменения в выборе первичного ключа, и эти изменения в конечном итоге могут потребовать использования суррогатных ключей.

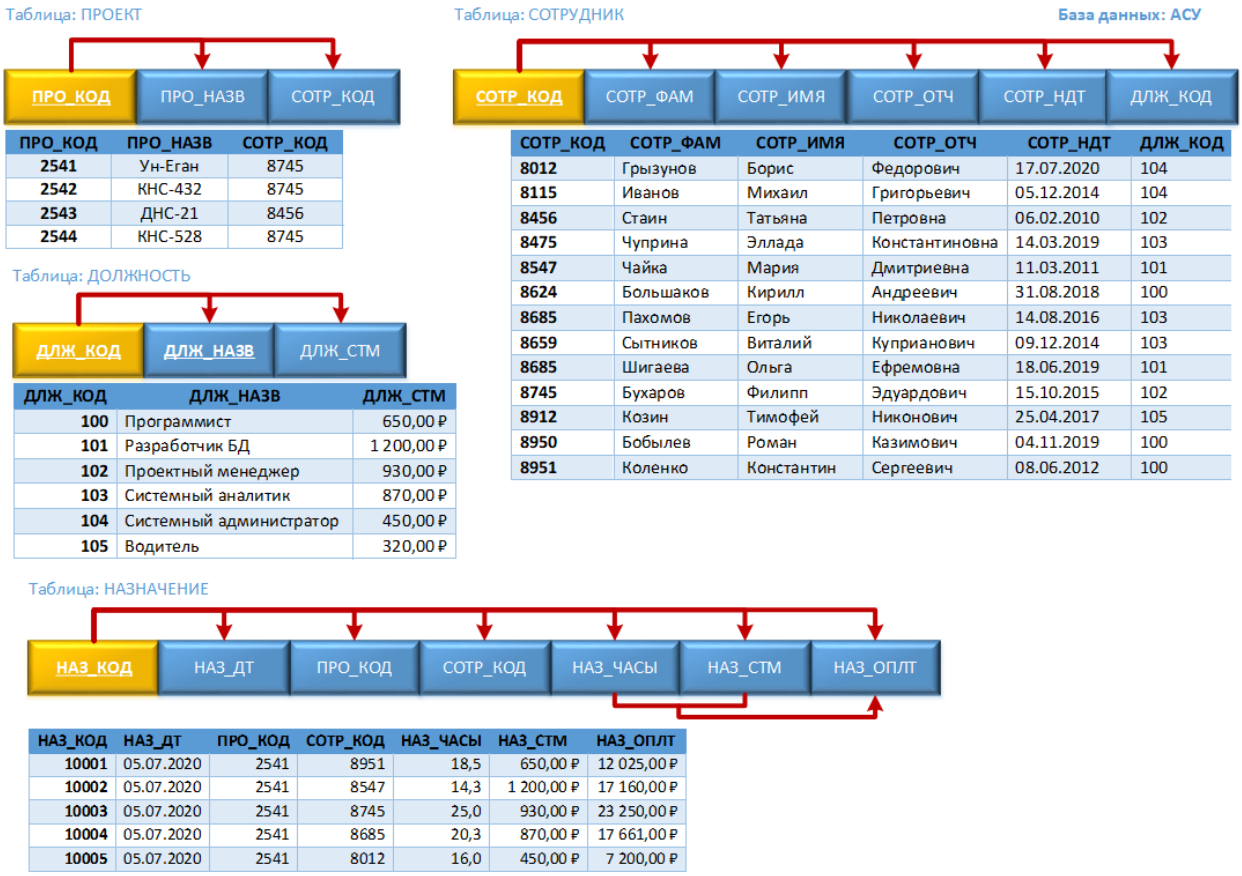
Сохранение исторической точности. Внесение оплаты за работу в час в таблицу НАЗНАЧЕНИЕ имеет решающее значение для поддержания исторической точности данных таблицы. Было бы целесообразно назвать этот атрибут НАЗ_СТМ. Хотя этот атрибут может иметь то же значение, что и ДЛЖ_СТМ, это верно только в том случае, если значение ДЛЖ_СТМ остается неизменным всегда. Разумно предположить, что стоимость работы за час будет меняться со временем. Если расходы по каждому проекту будут рассчитаны путем умножения отработанных часов из таблицы НАЗНАЧЕНИЕ на стоимость в час из таблицы ДОЛЖНОСТЬ, тогда они всегда будут отображать текущую стоимость работы за час, сохраненную в таблице ДОЛЖНОСТЬ, а не стоимость работы за час, который действовал на момент назначения.

Добавление производных атрибутов. Можно использовать производный атрибут в таблице НАЗНАЧЕНИЕ для хранения фактического начисления за проект. Этот производный атрибут с именем НАЗ_ОПЛТ является результатом умножения НАЗ_ЧАСЫ на НАЗ_СТМ. Это создает транзитивную зависимость:

(НАЗ_ОПЛТ + НАЗ_ЧАСЫ) → НАЗ_СТМ.

С точки зрения функциональности системы, такие производные значения атрибутов могут быть рассчитаны, когда они необходимы для написания отчетов. Однако сохранение производного атрибута в таблице облегчает написание прикладного программного обеспечения для получения желаемых результатов. Кроме того, если многие операции агрегировать, производный атрибут сэкономит время отчетности. Если вычисление выполняется во время ввода данных, оно будет завершено, когда пользователь нажмет клавишу Enter, что ускорит процесс.

Улучшения, описанные в предыдущих разделах, показаны в таблицах и диаграммах зависимостей, показанных на рисунке 20.



ПРО_КОД	ПРО_НАЗВ	СОТР_КОД
2541	Ун-Еган	8745
2542	КНС-432	8745
2543	ДНС-21	8456
2544	КНС-528	8745

СОТР_КОД	СОТР_ФАМ	СОТР_ИМЯ	СОТР_ОТЧ	СОТР_НДТ	ДЛЖ_КОД
8012	Грызунов	Борис	Федорович	17.07.2020	104
8115	Иванов	Михаил	Григорьевич	05.12.2014	104
8456	Стаин	Татьяна	Петровна	06.02.2010	102
8475	Чуприна	Эллада	Константиновна	14.03.2019	103
8547	Чайка	Мария	Дмитриевна	11.03.2011	101
8624	Большаков	Кирилл	Андреевич	31.08.2018	100
8685	Пахомов	Егорь	Николаевич	14.08.2016	103
8659	Сытников	Виталий	Куприанович	09.12.2014	103
8685	Шигаева	Ольга	Ефремовна	18.06.2019	101
8745	Бухаров	Филипп	Эдуардович	15.10.2015	102
8912	Козин	Тимофей	Никонович	25.04.2017	105
8950	Бобылев	Роман	Казимович	04.11.2019	100
8951	Коленко	Константин	Сергеевич	08.06.2012	100

ДЛЖ_КОД	ДЛЖ_НАЗВ	ДЛЖ_СТМ
100	Программист	650,00 ₽
101	Разработчик БД	1 200,00 ₽
102	Проектный менеджер	930,00 ₽
103	Системный аналитик	870,00 ₽
104	Системный администратор	450,00 ₽
105	Водитель	320,00 ₽

НАЗ_КОД	НАЗ_ДТ	ПРО_КОД	СОТР_КОД	НАЗ_ЧАСЫ	НАЗ_СТМ	НАЗ_ОПЛТ
10001	05.07.2020	2541	8951	18,5	650,00 ₽	12 025,00 ₽
10002	05.07.2020	2541	8547	14,3	1 200,00 ₽	17 160,00 ₽
10003	05.07.2020	2541	8745	25,0	930,00 ₽	23 250,00 ₽
10004	05.07.2020	2541	8685	20,3	870,00 ₽	17 661,00 ₽
10005	05.07.2020	2541	8012	16,0	450,00 ₽	7 200,00 ₽

Рис. 20. Проект базы данных

Рисунок 20 представляет собой значительное улучшение по сравнению с первоначальным проектом БД. Если прикладное программное обеспечение разработано правильно, для самой активной таблицы (НАЗНАЧЕНИЕ) требуется ввод только значений ПРО_КОД, СОТР_КОД и НАЗ_ЧАСЫ. Значения для атрибутов НАЗ_КОД и НАЗ_ДТ могут быть сгенерированы приложением. Например, НАЗ_КОД может быть создан с использованием счетчика, а НАЗ_ДТ может быть системной датой, считываемой приложением и автоматически вводимой в таблицу НАЗНАЧЕНИЕ. Кроме того, прикладное программное обеспечение может автоматически вставить правильное значение НАЗ_СТМ, записав значение ДЛЖ_СТМ из таблицы ДОЛЖНОСТЬ в таблицу НАЗНАЧЕНИЕ. Таблицы ДОЛЖНОСТЬ и НАЗНАЧЕНИЕ связаны через атрибут ДЛЖ_КОД. Если значение ДЛЖ_СТМ таблицы ДОЛЖНОСТЬ изменится, следующая вставка этого значения в таблицу НАЗНАЧЕНИЕ отразит это изменение автоматически. Таким образом, структура таблицы сводит к минимуму необходимость вмешательства человека.

Таблицы в 3НФ будут подходящим образом работать в БД бизнес-операций. Тем не менее, более высокие нормальные формы иногда полезны.

Таблица находится в *нормальной форме Бойса-Кодда (НФБК)*, когда каждый детерминант в таблице является ключом-кандидатом. Ключ-кандидат имеет те же характеристики, что и первичный ключ, но по определенной причине он не был выбран в качестве первичного ключа. Когда таблица содержит только один ключ-кандидат, 3НФ и

НФБК эквивалентны. Другими словами, НФБК может быть нарушена только тогда, когда таблица содержит более одного ключа-кандидата.

Таблица находится в четвертой нормальной форме (4НФ), когда она находится в 3НФ и не имеет многозначных зависимостей.

Таблица находится в пятой нормальной форме (5НФ), если она находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами.

5.5. Денормализация

Оптимальная реализация реляционной БД требует, чтобы все таблицы были как минимум в 3НФ. Современные СУРБД отлично справляются с управлением нормализованными таблицами. Создание нормализованных таблиц является важной целью проектирования БД, но это только одна из многих таких целей. Проект БД должен учитывать требования к информационности, скорость обработки. Проблема с нормализацией заключается в том, что при разложении таблиц в соответствии с требованиями нормализации, число таблиц увеличивается. Следовательно, для генерации информации необходимо собрать данные из разных таблиц. Объединение большого количества таблиц требует дополнительных операций ввода/вывода и логики обработки, тем самым снижая скорость работы системы. Большинство систем реляционных БД способно обрабатывать объединения очень эффективно. Однако некоторые обстоятельства могут требовать определенную степень денормализации.

Преимущество более высокой скорости обработки должно быть тщательно взвешено с недостатком аномалий данных. С другой стороны, некоторые аномалии представляют только теоретический интерес. Например, должны ли разработчики БД беспокоиться о том, что индекс определяет город в таблице КЛИЕНТ, первичным ключом которой является код клиента? Действительно ли практично производить отдельную таблицу для

ИНДЕКС (ИНД_КОД, ГОРОД).

Проблема с ненормализованными отношениями и избыточными данными заключается в том, что целостность данных может быть нарушена из-за возможности вставки, обновления и удаления аномалии.

Кроме того, процесс проектирования базы данных может, в некоторых случаях, вводить некоторую небольшую степень избыточных данных в модель. Это, по сути, создает «денормализованные» отношения.

Конфликты между эффективностью проектирования, требованиями к информации и производительностью часто разрешаются с помощью компромиссов, которые могут включать денормализацию. В этом случае, и при условии, что места для хранения достаточно, выбор разработчика может быть сужен до вариантов:

1. Хранить данные в постоянной денормализованной таблице. Это не рекомендуемое решение, потому что денормализованная таблица подвержена аномалиям данных (вставка, обновление и удаление). Это решение является жизнеспособным, только если производительность является проблемой.

2. Создать временную денормализованную таблицу из постоянных нормализованных таблиц. Денормализованная таблица существует только столько времени, сколько требуется для создания отчета; она исчезает после составления отчета. Следовательно, нет проблем с аномалией данных. Это решение практично только в том случае, если производительность не является проблемой и нет других жизнеспособных вариантов обработки.

Чистоту нормализации часто трудно поддерживать в современной среде БД. В специализированных БД, называемых хранилищами данных, встречаются (и даже требуются) более низкие формы нормализации. Такие специализированные БД отражают постоянно растущий спрос на больший объем и глубину данных, на которые все чаще опираются системы поддержки принятия решений. Хранилище данных регулярно использует структуры 2НФ в своей сложной многоуровневой среде с несколькими источниками данных.

Помимо возможности возникновения проблемных аномалий данных, ненормализованные таблицы страдают от следующих дефектов:

- Обновления данных менее эффективны, потому что программы, которые читают и обновляют таблицы, должны работать с большими таблицами.
- Индексирование является более громоздким. Просто нецелесообразно создавать все индексы, необходимые для множества атрибутов, которые могут находиться в одной ненормализованной таблице.
- Ненормализованные таблицы не дают простых стратегий для создания виртуальных таблиц, известных как представления.

Ненормализованные таблицы часто приводят к разным бедствиям избыточности данных в производственных БД. Нужно использовать денормализацию осторожно и убедиться, что есть разумное объяснение, почему ненормализованные таблицы являются лучшим выбором в определенных ситуациях, чем их нормализованные аналоги.

5.6. Контрольный список моделирования данных

Моделирование данных преобразует конкретную среду реального мира в модель данных, которая представляет реальные данные, пользователей, процессы и взаимодействия. Методы моделирования предлагают инструменты, необходимые для создания успешных проектов БД. Контрольный список моделирования данных поможет успешно выполнять задачи моделирования данных на основе имеющихся концепций и инструментов.

Бизнес-правила:

- Правильно документировать и проверять все бизнес-правила с пользователями.
- Убедиться, что все бизнес-правила написаны точно, четко и просто. Бизнес-правила должны помогать идентифицировать сущности, атрибуты, связи и ограничения.
- Определить источник всех бизнес-правил и убедиться, что каждое бизнес-правило обосновано, датировано и подписано утверждающим органом.

Моделирование данных:

1. Соглашения об именах: Все имена должны быть ограничены по длине (размер зависит от базы данных).
    - Имена сущностей:
        - Должны быть существительными, понятными, краткими и значимыми.
        - Должны документироваться сокращения, синонимы и псевдонимы для каждого объекта.
        - Должны быть уникальными в модели.
        - Для составных объектов могут включать в себя комбинацию сокращенных имен объектов, связанных через составной объект.
    - Имена атрибутов:
        - Должны быть уникальными в рамках организации.
        - Следует использовать аббревиатуру объекта в качестве префикса.
        - Должны описывать характеристики.
        - Следует использовать суффиксы, такие как _КОД, _НОМЕР или _ID для атрибута ПК.
        - Не должны быть зарезервированным словом.
        - Не должны содержать пробелов или специальных символов, таких как @,! или &.
    - Имена связей:
        - Должны быть активными или пассивными глаголами, четко указывающими на характер отношений.
2. Сущности:
    - Каждая сущность должна представлять один тип объекта.
    - Каждая сущность должна представлять множество отдельных экземпляров объекта.
    - Все сущности должны быть в 3НФ или выше. Любые сущности ниже 3НФ должны быть обоснованы.
    - Гранулярность должна быть четко определена.
    - ПК должен быть четко определен и поддерживать выбранную детализацию данных.
3. Атрибуты:
    - Должны быть простыми и однозначными (атомарность).
    - Должны документироваться значения по умолчанию, ограничения, синонимы и псевдонимы.
    - Производные атрибуты должны быть четко определены и включать источник.
    - Не должны быть избыточными, если это не требуется для точности транзакций, производительности или ведения истории.
    - Неключевые атрибуты должны полностью зависеть от атрибута ПК.

4. Связи:
    - Должны четко идентифицировать стороны связи.
    - Должны четко определять участие, возможности подключения и количество документов.
5. ER модель:
    - Должна быть проверена на соответствие ожидаемым процессам, таким как вставки, обновления и удаления.
    - Следует оценить, где, когда и как вести историю.
    - Не должна содержать избыточных отношений, за исключением случаев, когда это требуется.
    - Следует свести к минимуму избыточность данных для обеспечения единого обновления.
    - Должна соответствовать правилу минимальных данных: все, что нужно, есть, и все, что есть, нужно.

5.7. Итоги

 Нормализация – это метод, используемый для разработки таблиц, в которых избыточность данных сведена к минимуму. Первые три нормальные формы (1НФ, 2НФ и 3НФ) являются наиболее практичными. Со структурной точки зрения более высокие нормальные формы лучше, чем более низкие нормальные формы, потому что более высокие нормальные формы дают относительно меньшее количество избыточных данных в базе данных. Почти все бизнес-проекты используют 3НФ как идеальную нормальную форму. Также используется специальная, более ограниченная 3НФ, известная как нормальная форма Бойса-Кодда или НФБК.

 Таблица находится в 1НФ когда определены все ключевые атрибуты, а все остальные атрибуты зависят от первичного ключа. Однако таблица в 1НФ может содержать как частичные, так и транзитивные зависимости. Частичная зависимость – это зависимость, в которой атрибут функционально зависит только от части первичного ключа с несколькими атрибутами. Транзитивная зависимость – это зависимость, в которой атрибут функционально зависит от другого неключевого атрибута. Таблица с первичным ключом с одним атрибутом не может иметь частичных зависимостей.

 Таблица находится в 2НФ, когда она в 1НФ и не содержит частичных зависимостей. Следовательно, таблица 1НФ автоматически переходит в 2НФ, если ее первичный ключ основан только на одном атрибуте. Таблица в 2НФ все еще может содержать транзитивные зависимости.

 Таблица находится в 3НФ, когда она в 2НФ и не содержит транзитивных зависимостей. Учитывая это определение, нормальная форма Бойса-Кодда (НФБК) – особый случай 3НФ, в которой все определяющие ключи являются ключами-кандидатами. Когда таблица имеет только один ключ-кандидат, таблица 3НФ автоматически переходит в НФБК.

 Таблица, которая не входит в 3НФ, может быть разбита на новые таблицы, пока все таблицы не будут соответствовать требованиям 3НФ.

 Нормализация является важной частью процесса проектирования. Поскольку сущности и атрибуты определяются в процессе моделирования ER, каждая сущность подвергается проверкам нормализации и формирует новые сущности по мере необходимости.

 Таблица в 3НФ может содержать многозначные зависимости, которые создают либо многочисленные нулевые значения, либо избыточные данные. Поэтому можно преобразовать таблицу 3НФ в четвертую нормальную форму (4НФ), разделив таблицу, чтобы удалить многозначные зависимости. Таким образом, таблица находится в 4НФ, когда она находится в 3НФ и не содержит многозначных зависимостей.

 Чем больше таблиц, тем больше дополнительных операций ввода-вывода и логики обработки необходимо для их соединения. Поэтому таблицы иногда денормализуются, чтобы уменьшить ввод-вывод для увеличения скорости обработки. Денормализация больших таблиц увеличивает скорость обработки, но делает обновления данных менее эффективными, индексирование более громоздким и приводит к избыточности данных, которая может привести к аномалиям. При проектировании производственных баз данных нужно использовать денормализацию с осторожностью.

 Контрольный список для моделирования данных предоставляет разработчику возможность проверить, соответствует ли ERD набор минимальных требований.

Лекция 6. Проектирование баз данных

6.1. Введение

Базы данных являются частью общей картины, называемой информационной системой. Проекты баз данных, которые не распознают этот факт, вряд ли будут успешными. Разработчики базы данных должны признать, что база данных является критическим средством для достижения цели, а не самой целью.

Информационные системы являются продуктом тщательно организованного процесса разработки. Системный анализ используется для определения потребности в информационной системе и установления ее ограничений. В рамках системного анализа фактическая информационная система создается с помощью процесса, известного как разработка систем.

Создание и развитие информационных систем происходит по итерационной схеме, называемой жизненным циклом информационных систем (ЖЦ ИС), которая представляет собой непрерывный процесс создания, обслуживания, улучшения и замены информационной системы. Аналогичный цикл применяется к БД: база данных создается, поддерживается, совершенствуется и в конечном итоге заменяется. Жизненный цикл базы данных (ЖЦ БД) нужно изучать в контексте более широкого жизненного цикла разработки систем.

База данных – это тщательно спроектированное и сконструированное хранилище фактов. База данных является частью большего целого, известного как *информационная система (ИС)*, которая обеспечивает сбор, хранение, преобразование и поиск данных. Информационная система также помогает преобразовывать данные в информацию и позволяет управлять как данными, так и информацией. Таким образом, полная информационная система состоит из людей, оборудования, программного обеспечения, базы данных, прикладных программ и процедур. *Системный анализ* – процесс, который устанавливает потребность в информационной системе и ее объем. Процесс создания информационной системы известен как *разработка систем*.

Одной из ключевых характеристик современных информационных систем является стратегическая ценность информации в эпоху глобализации. Поэтому информационные системы должны соответствовать стратегической бизнес-миссии и целям. Изолированные и независимые информационные системы больше не действительны. Современные информационные системы всегда должны быть интегрированы с корпоративной архитектурой информационных систем компании.

В рамках разработки систем, приложения преобразуют данные в информацию, которая служит основой для принятия решений. Приложения обычно генерируют формальные отчеты, таблицы и графические формы, предназначенные для получения информации. На рисунке 21 показано, что каждое приложение состоит из двух частей: данных и кода (программных инструкций), с помощью которых данные преобразуются в информацию. Данные и код работают вместе, чтобы представить реальные бизнес-функции и действия. В любой момент

времени физически хранимые данные представляют собой снимок бизнеса, но картина не является полной без понимания бизнес-операций, представленных кодом.

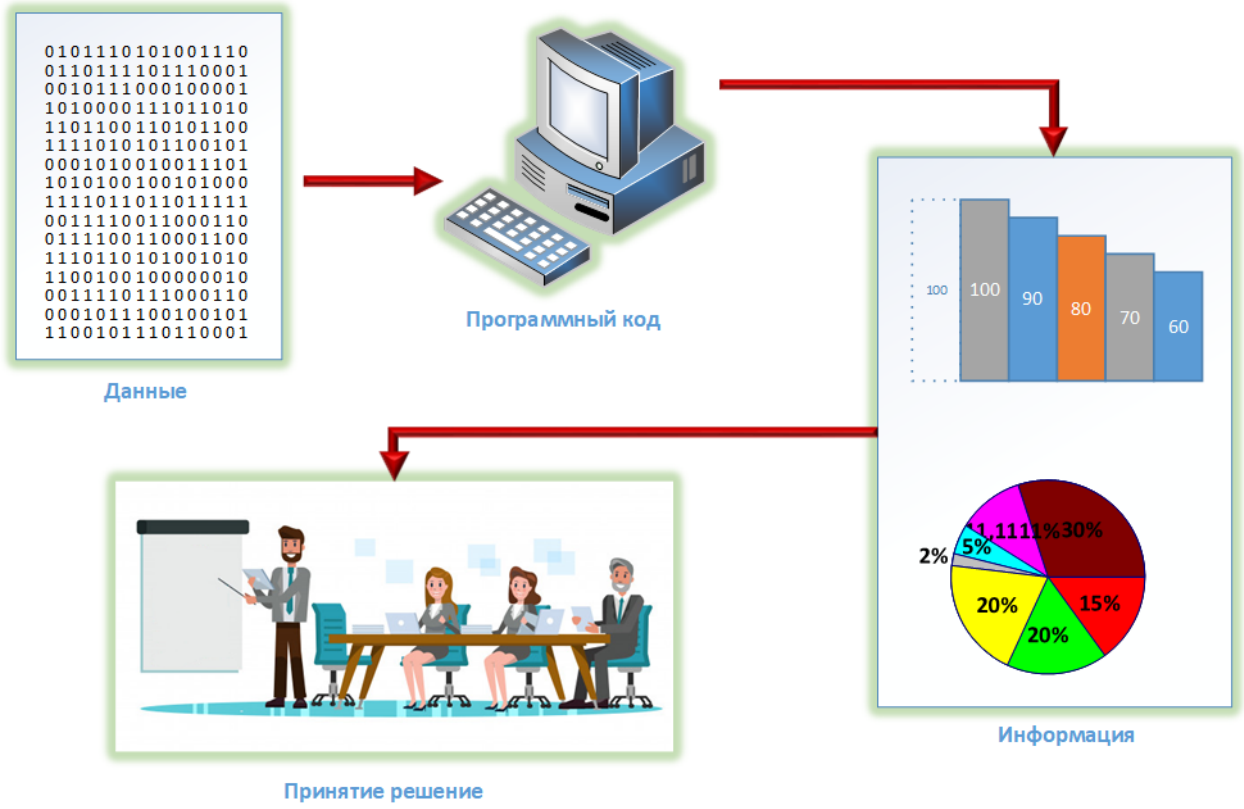


Рис. 21. Принятие решение на основе данных

Производительность информационной системы зависит от трех факторов:

1. Проектирование и реализация базы данных.
2. Разработка и внедрение приложений.
3. Административные процедуры.

Возможно, самый важный из трех факторов – проектирование и реализация базы данных. Создание надежной информационной системы – тяжелая работа: системный анализ и разработка требуют обширного планирования, чтобы гарантировать, что все действия будут взаимодействовать друг с другом, дополнять друг друга и выполняться вовремя.

В широком смысле термин *разработка базы данных* описывает процесс проектирования и реализации БД. Основной задачей при проектировании БД является создание законченных, нормализованных, интегрированных концептуальных, логических и физических моделей баз данных. Этап реализации включает создание структуры хранения БД, загрузку данных и обеспечение управления данными. Большинство проектов обычно ориентированы на решение текущих проблем, но важно создать проект, который будет достаточно гибким для адаптации к будущим изменениям.

6.2. Жизненный цикл информационных систем

Жизненный цикл информационных систем (ЖЦ ИС) отслеживает историю информационной системы. ЖЦ ИС – это общая структура, с помощью которой можно отслеживать и понимать действия, необходимые для разработки и обслуживания ИС. В этих рамках существует несколько способов выполнения различных заданий в ЖЦ ИС. Кроме моделирования ER, есть альтернативные методологии:

- Unified Modeling Language (UML) предоставляет объектно-ориентированные инструменты для поддержки задач, связанных с разработкой информационных систем.
- Быстрая разработка приложений (RAD) – это методология итеративной разработки программного обеспечения, в которой для разработки систем приложений используются прототипы, инструменты CASE и гибкое управление. RAD – альтернатива традиционной структурированной разработки, которая страдала от длительных сроков поставки и невыполненных требований.
- Agile Software Development – гибкая методология для разработки программных приложений, которая делит работу на более мелкие подпроекты, чтобы получить результаты за более короткое время и с лучшей согласованностью. Этот метод подчеркивает тесное общение между всеми пользователями и постоянную оценку с целью повышения удовлетворенности клиентов.

Хотя методологии разработки могут изменяться, базовая структура, в которой они используются, не меняется.

Как показано на рисунке 22, традиционный ЖЦ ИС разделен на пять этапов: планирование, анализ, детальное проектирование систем, разработка, внедрение и обслуживание. ЖЦ ИС – это итеративный процесс, а не последовательный процесс. Например, подробности технико-экономического обоснования могут помочь уточнить начальную оценку, а детали, обнаруженные во время части требований пользователя ЖЦ ИС, могут помочь уточнить технико-экономическое обоснование.

Поскольку жизненный цикл базы данных подходит и напоминает ЖЦ ИС, приведено краткое описание ЖЦ ИС.

Постановка задачи. Первая фаза ЖЦ ИС дает общий обзор предприятий и ее целей. Первоначальная оценка требований к потоку и объему информации должна быть сделана во время данного этапа ЖЦ ИС. Такая оценка должна ответить на несколько важных вопросов:

- *Следует ли продолжать существующую систему?* Если генератор информации делает свою работу хорошо, нет смысла изменять или заменять его.
- *Следует ли изменить существующую систему?* Если первоначальная оценка указывает на недостатки в объеме и потоке информации, могут потребоваться незначительные (или даже значительные) изменения. При рассмотрении изменений участники первоначальной оценки должны помнить о разнице между желаниями и потребностями.
- *Следует ли заменить существующую систему?* Первоначальная оценка может указывать на то, что недостатки нынешней системы не подлежат исправлению.

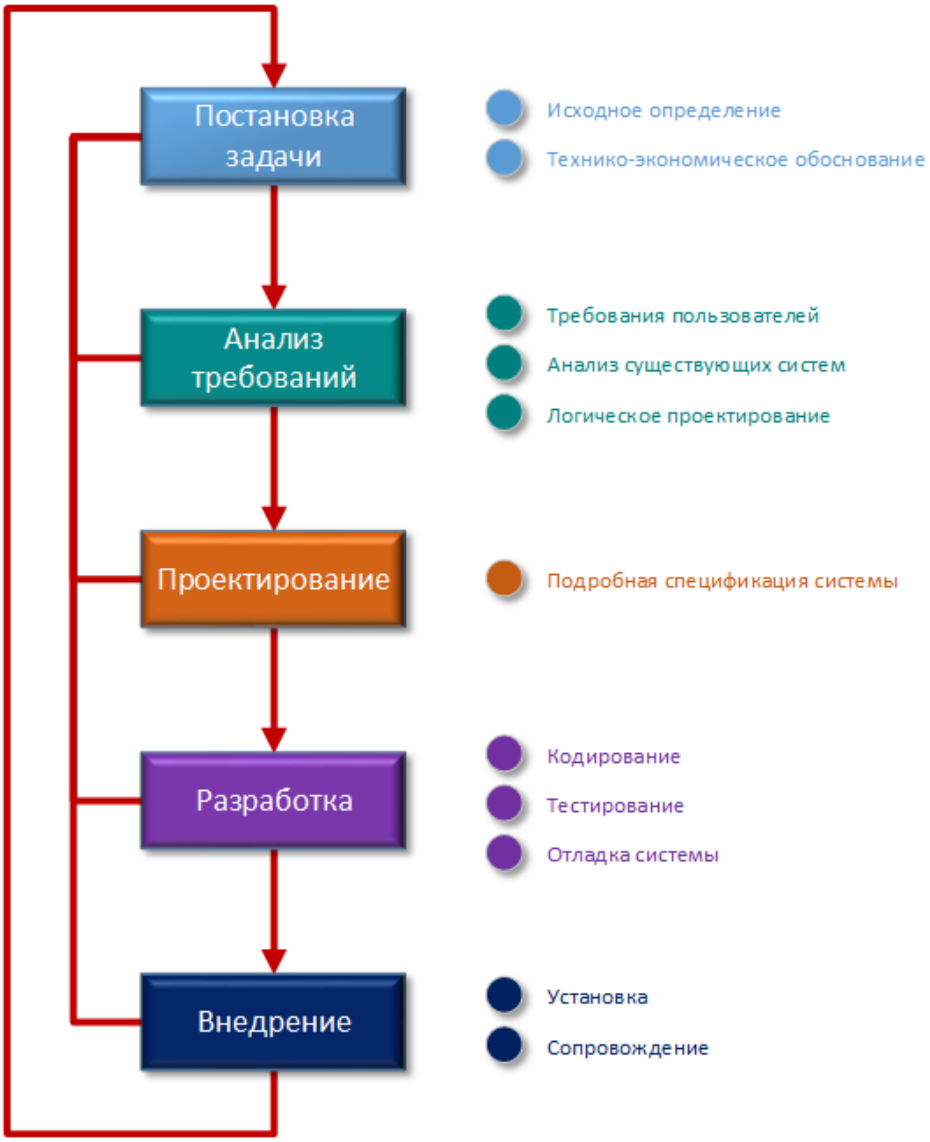


Рис. 22. Жизненный цикл разработки систем

Учитывая усилия, необходимые для создания новой системы, тщательное разграничение между пожеланиями и потребностями, возможно, даже более важно в этом случае, чем при модификации системы.

Участники первоначальной оценки ЖЦ ИС должны начать изучать и оценивать альтернативные решения. Если новая система необходима, то следующий вопрос – возможно ли это? Технико-экономическое обоснование должно касаться следующего:

- *Технические аспекты требований к аппаратному и программному обеспечению.* Решения могут еще не зависеть от поставщика, но они должны учитывать природу требований к оборудованию (настольный компьютер, мэйнфрейм, суперкомпьютер или мобильное устройство) и требования к программному обеспечению (однопользовательские или многопользовательские операционные системы, тип базы данных и программное обеспечение, языки программирования, используемые приложениями и т.д.).

- *Стоимость системы.* Общепризнанный приземленный вопрос «Можем ли мы себе это позволить?» является решающим. Ответ может вызвать тщательный анализ первоначальной оценки. В какой-то момент решение может быть принято между созданием собственной системы или покупкой (и настройкой) системы сторонних поставщиков. В долгосрочной перспективе необходимо найти экономически эффективное решение, которое наилучшим образом отвечает потребностям (настоящим и будущим) организации.
- *Операционные расходы.* Обладает ли предприятие человеческими, техническими и финансовыми ресурсами для поддержания работоспособности системы? Должно ли технико-экономическое обоснование включать затраты на управление и поддержку конечного пользователя, необходимые для внедрения операционных процедур для обеспечения успеха этой системы? Народное сопротивление переменам никогда не следует недооценивать.

Даже если решено «покупать», а не «строить», внедрение системы должно быть тщательно спланировано, чтобы она была успешной. Каким бы ни был выбранный вариант (построить или купить), необходимо провести анализ, чтобы развернуть решение в организации таким образом, чтобы уменьшить затраты и изменения в культуре, одновременно увеличивая ценность. ЖЦ ИС обеспечивает основу для тщательного планирования и реализации.

Анализ требований. Проблемы, определенные на первом этапе, рассматриваются более подробно на этапе анализа. Макроанализ должен проводиться как по индивидуальным потребностям, так и по организационным потребностям, затрагивая такие вопросы, как

- Каковы требования пользователей текущей системы?
- Вписываются ли эти требования в общие требования к информации?

Фаза анализа ЖЦ ИС – тщательный аудит требований пользователей.

Существующие аппаратные и программные системы также изучаются на этапе анализа. Результатом анализа должно стать лучшее понимание функциональных областей системы, реальных и потенциальных проблем и возможностей.

Конечные пользователи и разработчики систем должны работать вместе, чтобы идентифицировать процессы и раскрыть потенциальные проблемные области. Такое сотрудничество имеет жизненно важное значение для определения соответствующих целевых показателей эффективности, по которым можно судить о новой системе.

Наряду с изучением требований пользователей и существующих систем, фаза анализа также включает создание логического проектирования систем. Логический дизайн должен указывать соответствующую концептуальную модель данных, входные данные, процессы и ожидаемые выходные требования.

При создании логического проекта разработчик может использовать такие инструменты, как диаграммы потоков данных (DFD), диаграммы иерархического процесса ввода-вывода (HIPO), диаграммы отношений сущностей (ER) и даже некоторые прототипы приложений. На этом этапе выполняются действия по моделированию проекта базы данных,

чтобы обнаружить и описать все сущности и их атрибуты, а также связи между сущностями в базе данных.

Определение логической системы также дает функциональные описания компонентов системы (модулей) для каждого процесса в среде БД. Все преобразования (процессы) данных описываются и документируются с использованием инструментов системного анализа, таких как DFD. Концептуальная модель данных проверяется на соответствие этим процессам.

Детальное проектирование систем. На этапе детального проектирования систем проектировщик завершает проектирование процессов системы. Конструкция включает в себя все необходимые технические спецификации для экранов, меню, отчетов и других устройств, которые могут помочь сделать систему более эффективным генератором информации. Этапы изложены для преобразования из старой системы в новую систему. Принципы и методики обучения также планируются и представляются на утверждение руководству.

При разработке решений разработчик базы данных должен искать источник проблем. Многие системы баз данных не смогли удовлетворить пользователей, потому что они были разработаны для лечения симптомов проблем, а не их источника.

Разработка. На этапе разработки устанавливаются аппаратное обеспечение, программное обеспечение СУБД и прикладные программы, а также реализуется проект базы данных. На начальных этапах система входит в цикл кодирования, тестирования и отладки, пока не будет готова к доставке. Фактическая база данных создается, а система настраивается путем создания таблиц и представлений, авторизации пользователей и так далее.

Содержимое базы данных может быть загружено в интерактивном или пакетном режиме с использованием разных методов и устройств:

- Индивидуальные пользовательские программы.
- Программы интерфейса базы данных.
- Программы преобразования, которые импортируют данные из другой файловой структуры, используя пакетные программы, утилиту базы данных или и то, и другое.

Система подвергается тщательному тестированию до тех пор, пока не будет готова к использованию. Традиционно отладка и тестирование новой системы занимают от 50 до 60% общего времени разработки. Однако появление сложных генераторов приложений и инструментов отладки существенно сократило время кодирования и тестирования. После завершения тестирования окончательная документация проверяется и распечатывается, а пользователи проходят обучение. В конце этого этапа система полностью работает, но она будет постоянно оцениваться и настраиваться.

Внедрение. Как только система заработает, пользователи начнут запрашивать изменения в ней. Эти изменения порождают действия по обслуживанию системы, которые можно сгруппировать в три типа:

- *Корректирующее обслуживание* в ответ на системные ошибки.
- *Адаптивное обслуживание* в связи с изменениями в бизнес-среде.
- *Перфективное обслуживание* для улучшения системы.

Поскольку каждый запрос на структурные изменения требует пересмотра этапов ЖЦ ИС, в некотором смысле система всегда находится на некоторой стадии ЖЦ ИС.

6.3. Жизненный цикл базы данных

В более крупной информационной системе база данных также подвержена жизненному циклу. *Жизненный цикл базы данных (ЖЦ БД)* состоит из шести фаз, как показано на рисунке 23: изучение среды базы данных, проектирование базы данных, ее реализация и загрузка, тестирование и оценка, эксплуатация, а также обслуживание и развитие.

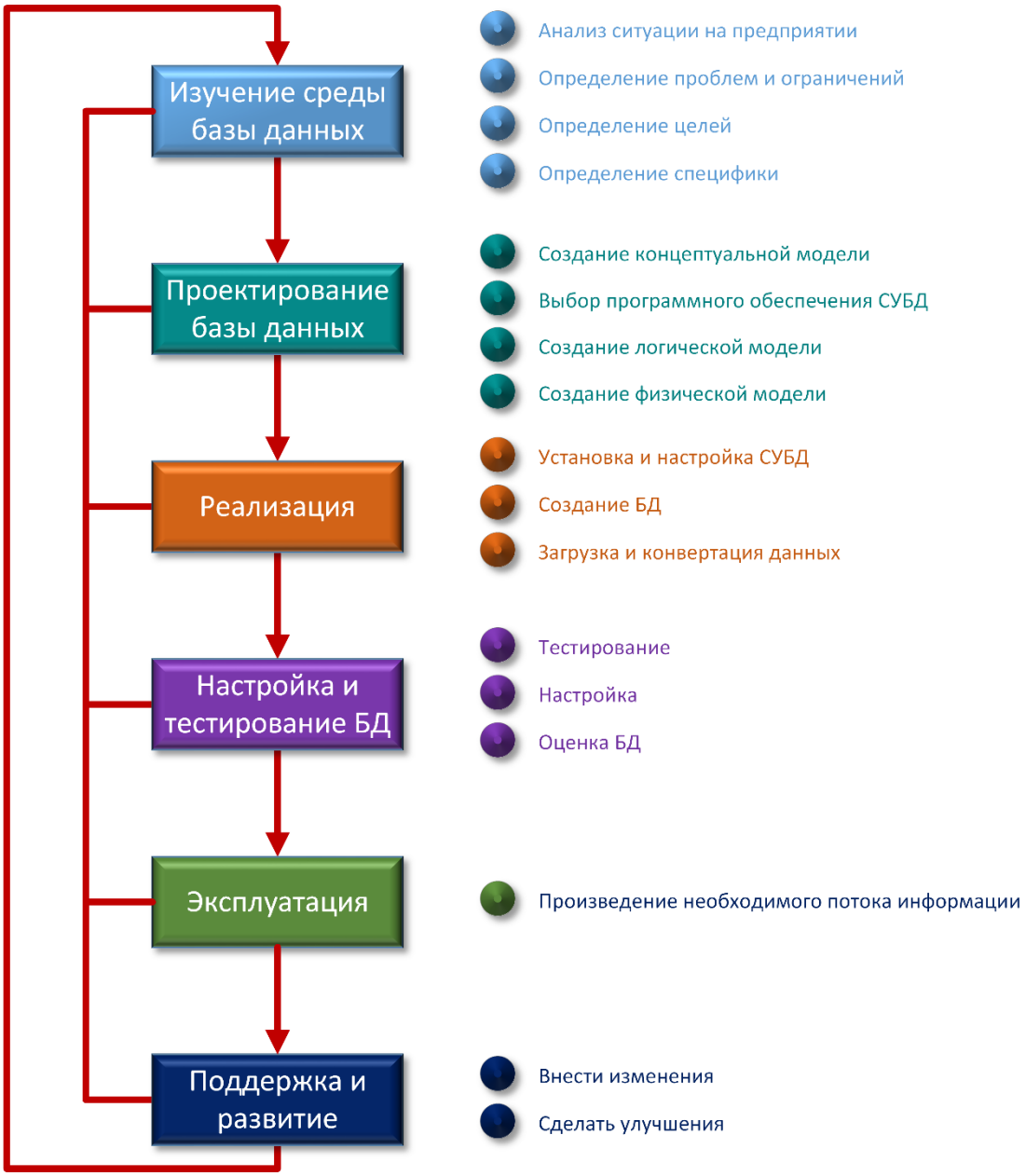


Рис. 23. Жизненный цикл базы данных

Изучение среды базы данных. Если вызван разработчик БД, есть вероятность, что текущая система не смогла выполнить функции, которые компания считает жизненно важными. Поэтому, помимо изучения текущей работы системы в компании, разработчик

должен определить, как и почему текущая система выходит из строя. Это означает, что нужно тратить много времени на общение с пользователями. Несмотря на то, что проектирование БД – техническая деятельность, оно также ориентировано на людей. Разработчики БД должны иметь навыки общения с клиентами.

В зависимости от сложности и объема работы, разработчик может действовать один или как часть группы, состоящей из руководителя проекта, одного или нескольких системных аналитиков, программистов.

Общая цель изучение среды базы данных состоит в том, чтобы:

- Анализировать ситуацию на предприятии.
- Определить проблемы и ограничения.
- Поставить цели.
- Определить специфику.

Анализ ситуации на предприятии. Ситуация на предприятии описывает общие условия, в которых оно работает, его организационную структуру и миссию. Чтобы проанализировать ситуацию, разработчик БД должен изучить операционные компоненты компании, как они функционируют и взаимодействуют.

Следующие проблемы должны быть решены:

- *Какова общая операционная среда организации и какова ее миссия в этой среде?* Конструкция должна удовлетворять эксплуатационным требованиям, предъявляемым миссией организации. Например, бизнес по почтовым заказам, вероятно, имеет операционные требования к своей базе данных, которые сильно отличаются от требований производственного бизнеса.
- *Какова структура организации?* Знание того, кто контролирует, что и кто отчитывается перед кем, весьма полезно, когда нужно определить необходимые информационные потоки, конкретные форматы отчетов и запросов и т.д.

Определение проблем и ограничений. У разработчика есть как формальные, так и неформальные источники информации. Если предприятие существует какое-то время, у него уже есть система (ручная или компьютерная). Как функционирует существующая система? Какой ввод требует система? Какие документы генерирует система? Кем и как используется выход системы? Изучение бумажного следа может быть очень информативным. Помимо официальной версии работы системы, существует и более неформальная, возможно, более реальная версия; разработчик должен быть достаточно проницательным, чтобы понять, чем они отличаются.

Процесс определения проблем изначально может показаться неструктурированным. Пользователи предприятия часто не могут точно описать больший объем операций компании или определить реальные проблемы, возникающие в ходе работы компании. Часто управленческий взгляд на деятельность компании и ее проблемы отличается от взгляда пользователей, которые выполняют обычную рутинную работу.

Во время начального процесса определения проблемы разработчик может собирать очень широкие описания проблем. Процесс определения проблемы быстро приводит к множеству общих описаний проблем. После первоначальных деклараций разработчик базы данных должен продолжать тщательно исследовать, чтобы сгенерировать дополнительную информацию, которая поможет определить проблемы в более широких рамках деятельности компании.

Важно найти точные ответы, особенно в сфере операционных отношений между бизнес-единицами. Если предложенная система решит проблемы отдела маркетинга, но усугубит проблемы производственного отдела, не будет достигнуто значительного прогресса.

Даже самое полное и точное определение проблемы не всегда приводит к идеальному решению. Реальный мир обычно вмешивается, чтобы ограничить проект даже самой элегантной базы данных, накладывая ограничения, такие как время, бюджет и персонал. Разработчик должен научиться различать, что идеально и что возможно.

Определение целей. Предлагаемая система БД должна быть спроектирована так, чтобы помочь решить хотя бы основные проблемы, выявленные в процессе обнаружения проблем. Поскольку список проблем разворачивается, несколько общих источников, вероятно, будут обнаружены.

При разработке решений разработчик БД должен искать источник проблем. Многие системы БД не смогли удовлетворить пользователей, потому что они были разработаны для лечения симптомов проблем, а не их источника.

Начальная фаза исследования также дает предлагаемые решения проблем. Задача разработчика состоит в том, чтобы убедиться, что его цели системы БД соответствуют тем, которые предусмотрены пользователем. В любом случае, разработчик БД должен начать решать следующие вопросы:

- Какова первоначальная цель предлагаемой системы?
- Будет ли система взаимодействовать с другими существующими или будущими системами в компании?
- Будет ли система делиться данными с другими системами или пользователями?

Определение специфики. Разработчик должен распознавать два набора ограничений: область действия и границы. *Область применения* системы определяет степень проекта в соответствии с эксплуатационными требованиями. Будет ли проект базы данных охватывать всю организацию, один или несколько отделов внутри организации или функций одного отдела? Знание области помогает определить необходимые структуры данных, тип и количество сущностей, физический размер базы данных и т.д.

Предлагаемая система также подвержена ограничениям, известным как *границы*, которые являются внешними по отношению к системе. Кроме времени, бюджета и кадров, границы также накладываются существующим аппаратным и программным обеспечением. В идеале разработчик может выбрать аппаратное и программное обеспечение, которое будет наилучшим образом выполнять системные задачи. Фактически, выбор программного обеспечения является важным аспектом жизненного цикла разработки систем. К сожалению,

в реальном мире система часто должна быть разработана на основе существующего оборудования. Таким образом, область действия и границы становятся факторами, и задача разработчика состоит в том, чтобы спроектировать наилучшую возможную систему с учетом этих ограничений.

Проектирование базы данных. Второй этап ЖЦ БД фокусируется на разработке модели базы данных, которая будет поддерживать операции и цели предприятия. Это, пожалуй, самый важный этап ЖЦ БД: убедиться, что конечный продукт соответствует требованиям пользователя и системы. В процессе проектирования БД разработчик должен сосредоточиться на свойствах данных, необходимых для построения модели. На этом этапе в системе существует два представления данных: бизнес-представление данных как источника информации и представление разработчика о структуре данных, ее доступе и действиях, необходимых для преобразования данных в информацию. Определение данных является неотъемлемой частью второй фазы ЖЦ БД.

При изучении процедур, необходимых для завершения этапа проектирования в ЖЦ БД, помните следующие моменты:

- Процесс проектирования БД слабо связан с анализом и проектированием более крупной системы. Компонент данных – это только один элемент более крупной информационной системы.
- Системные аналитики или системные программисты отвечают за разработку других компонентов системы. Их деятельность создает процедуры, которые помогут преобразовать данные в БД в полезную информацию.
- Проект БД не является последовательным процессом. Скорее, это итеративный процесс, который обеспечивает непрерывную обратную связь, предназначенную для отслеживания предыдущих шагов.

Процесс проектирования базы данных изображен на рисунке 24. На рисунке показано, что существует три основных этапа: концептуальное, логическое и физическое проектирование, а также решение о выборе СУБД, которое имеет решающее значение для определения типа логических и физических проектов. Процесс проектирования начинается с концептуального проектирования, и переходит к этапам логического и физического проектирования. На каждом этапе определяется и документируется более подробная информация о проекте модели данных. Можно рассматривать концептуальный проект как общие данные, видимые пользователю; логический проект – как данные, видимые СУБД; и физический проект – как данные, видимые устройствами управления хранением операционной системы.

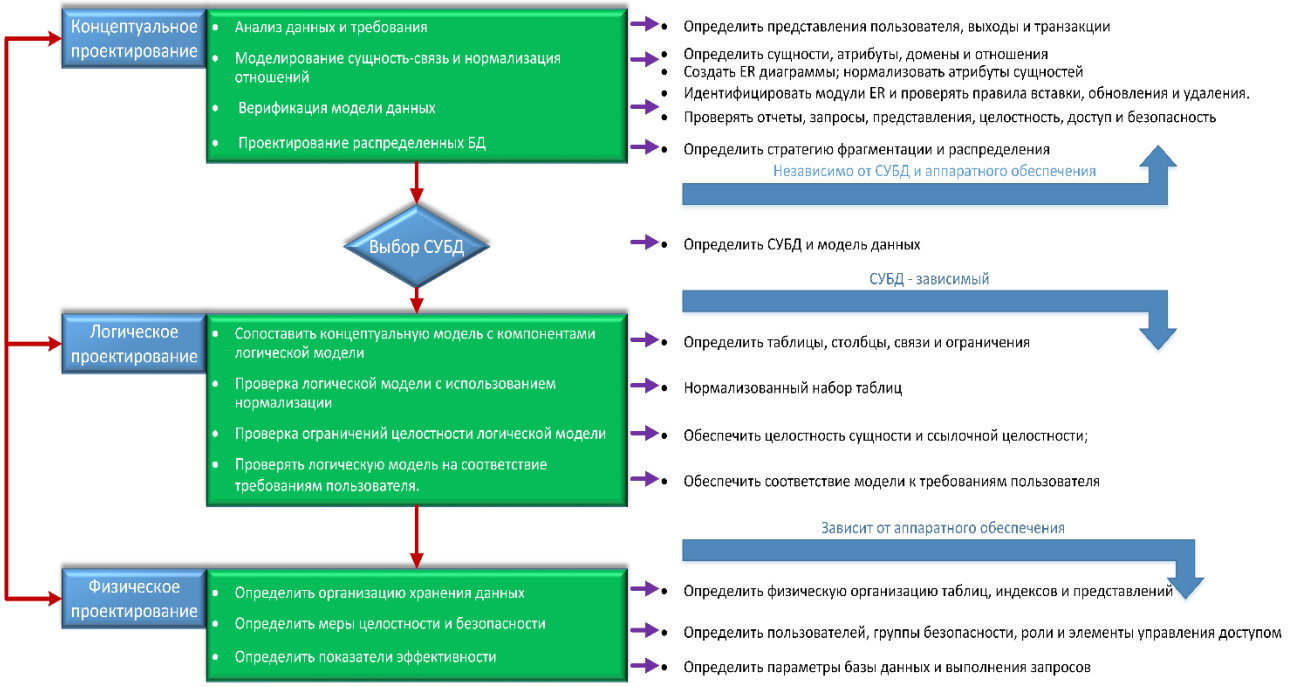


Рис. 24. Процесс проектирования БД

Важно отметить, что подавляющее большинство проектов и реализаций баз данных основаны на реляционной модели, и поэтому используют именно ее конструкции и методы.

Реализация. Результатом этапа проектирования БД является ряд инструкций, подробно описывающих создание таблиц, атрибутов, доменов, представлений, индексов, ограничений безопасности и рекомендаций по хранению и производительности. На этом этапе реализуются все спецификации проекта.

Установка СУБД. Этот шаг требуется только в том случае, если для системы необходим новый выделенный экземпляр СУБД. Во многих случаях организация делает конкретную СУБД стандартом, чтобы использовать инвестиции в технологии и навыки, которые сотрудники уже развили. СУБД может быть установлена на новом сервере или на существующих серверах. Одна из текущих тенденций называется виртуализацией. *Виртуализация* – это метод, который создает логические представления вычислительных ресурсов, не зависящие от базовых физических вычислительных ресурсов. Этот метод используется во многих областях вычислений, таких как создание виртуальных серверов, виртуальных хранилищ и виртуальных частных сетей. В среде БД под виртуализацией базы данных понимается установка нового экземпляра СУБД на виртуальном сервере, работающем на общем оборудовании. Обычно это задача, которая требует от системных и сетевых администраторов создания соответствующих групп пользователей и служб в конфигурации сервера и сетевой маршрутизации. Другой распространенной тенденцией является использование облачных служб баз данных, таких как служба баз данных Microsoft Azure или Amazon Relational Database Services (RDS). Это новое поколение сервисов позволяет

пользователю создавать базы данных, которыми можно легко управлять, тестировать и масштабировать по мере необходимости.

Создание базы данных. В большинстве современных реляционных СУБД реализация новой БД требует создания специальных конструкций, связанных с хранением, для размещения таблиц пользователя. Конструкции обычно включают группу хранения (или группы файлов), табличные пространства и таблицы.

Загрузка и конвертация данных. После создания БД данные должны быть загружены в таблицы. Как правило, данные должны быть перенесены из предыдущей версии системы. Часто данные, которые должны быть включены в систему, объединяются из нескольких источников. В лучшем случае все данные будут находиться в реляционной БД, чтобы их можно было легко перенести в новую БД. Однако в некоторых случаях данные, возможно, придется импортировать из других реляционных, нереляционных БД, текстовых файлов, устаревших систем или даже ручных систем «бумага и карандаш». Если формат данных не поддерживает прямой импорт в новую БД, может потребоваться создание программ преобразования для форматирования данных для импорта. В худшем случае большая часть данных может быть вручную введена в БД. После загрузки данных администратор БД работает с разработчиками приложений для проверки и оценки их.

Загрузка существующих данных в облачную базу данных иногда может быть дорогой. Причина этого заключается в том, что стоимость большинства облачных услуг определяется не только объемом хранимых данных, но и объемом данных, передаваемых по сети. В таких случаях загрузка базы данных объемом 1 ТБ может оказаться очень дорогим предложением. Поэтому системные администраторы должны быть очень внимательными при чтении и согласовании условий контрактов на облачные сервисы, чтобы не было «скрытых» затрат.

Тестирование и оценка. На этапе проектирования были приняты решения для обеспечения целостности, безопасности, производительности и возможности восстановления БД. Во время реализации и загрузки эти планы были реализованы. При тестировании и оценке администратор БД тестирует и настраивает БД, чтобы убедиться, что она работает должным образом. Этот этап происходит в сочетании с прикладным программированием. Программисты используют инструменты БД для создания прототипов приложений во время кодирования программ. Такие инструменты, как генераторы отчетов, экранные формы и генераторы меню особенно полезны для программистов приложений.

Тестирование базы данных. На этом этапе администратор БД тестирует базу данных, чтобы убедиться, что она поддерживает целостность и безопасность данных. Целостность данных обеспечивается СУБД путем правильного использования первичного и внешнего ключей. Многие СУБД также поддерживают создание ограничений домена и триггеров БД. Тестирование будет гарантировать, что эти ограничения были правильно спроектированы и реализованы. Целостность данных также является результатом правильно реализованной политикой управления данными, которые являются частью комплексной структуры управления данными.

Ранее пользователи и роли создавались для предоставления доступа к данным. На этом этапе должны быть проверены не только эти привилегии, но и более широкий взгляд на конфиденциальность и безопасность данных. Данные, хранящиеся в БД предприятия, должны быть защищены от доступа посторонних пользователей. Следовательно, нужно проверить по крайней мере следующее:

- *Физическая безопасность* разрешает только авторизованному персоналу физический доступ в определенные зоны. Однако, в зависимости от типа реализации базы данных, физическая защита не всегда может быть практичной.
- *Защита паролем* позволяет назначать права доступа определенным авторизованным пользователям. Безопасность пароля обычно обеспечивается во время входа в систему на уровне операционной системы.
- *Права доступа* могут быть установлены с помощью программного обеспечения базы данных. Назначение прав доступа может ограничивать операции (CREATE, UPDATE, DELETE и т.д.) над заранее определенными объектами, такими как базы данных, таблицы, представления, запросы и отчеты.
- *Контрольные журналы* обычно предоставляются СУБД для проверки нарушений доступа. Несмотря на то, что контрольный журнал является постфактумным устройством, его простое существование может препятствовать несанкционированному использованию.
- *Шифрование данных* может сделать данные бесполезными для неавторизованных пользователей, которые могли нарушить некоторые уровни безопасности базы данных.
- *Бездисковые терминалы* позволяют конечным пользователям получать доступ к БД, не имея возможности загружать информацию со своих рабочих станций.

Настройка базы данных. Производительность БД может быть трудно оценить, потому что нет стандартов для ее измерения, но, как правило, это один из наиболее важных факторов в реализации. Разные системы будут размещать собственные требования к производительности. Системы, поддерживающие быстрые транзакции, потребуют внедрения БД, чтобы обеспечить приемлемую производительность при больших объемах вставок, обновлений и удалений. Другие системы, такие как системы поддержки принятия решений, могут требовать приемлемую производительности для сложных задач поиска данных. Многие факторы могут влиять на производительность базы данных при выполнении разных задач, включая аппаратную и программную среду, в которой БД существует. Естественно, характеристики и объем данных также влияют на производительность базы данных: поиск по 10 кортежам быстрее, чем поиск по 100 000 кортежей. Другие важные факторы производительности БД включают параметры конфигурации системы и БД, такие как размещение данных, определение пути доступа, использование индексов и размер буфера.

Оценка базы данных и ее прикладных программ. По мере того, как БД и прикладные программы создаются и тестируются, система также должна оцениваться с использованием

более целостного подхода. Тестирование и оценка отдельных компонентов должны завершаться множеством более широких системных тестов, чтобы гарантировать, что все компоненты взаимодействуют должным образом для удовлетворения потребностей пользователей. На этом этапе уточняются вопросы интеграции и планы развертывания, проводится обучение пользователей и дорабатывается системная документация. Как только система получает окончательное одобрение, она должна стать устойчивым ресурсом для организации. Чтобы убедиться, что данные, содержащиеся в БД, защищены от потери, проверяются планы резервного копирования и восстановления.

Своевременная доступность данных имеет решающее значение практически для каждой БД. К сожалению, БД может потерять данные в результате непреднамеренного удаления, отключения питания и других причин. Процедуры резервного копирования и восстановления данных создают предохранительный клапан, обеспечивающий доступность согласованных данных. Как правило, поставщики БД поощряют использование отказоустойчивых компонентов, таких как источники бесперебойного питания (ИБП), устройства хранения RAID, кластерные серверы и технологии репликации данных, чтобы обеспечить непрерывную работу БД в случае сбоя оборудования. Даже с этими компонентами функции резервного копирования и восстановления составляют очень важную часть ежедневных операций с БД. Некоторые СУБД предоставляют функции, которые позволяют администратору БД планировать автоматическое резервное копирование на внешние устройства хранения, такие как диски, CD/DVD, магнитные ленты и онлайн-хранилище. Резервное копирование БД может выполняться на разных уровнях:

- *Полная резервная копия* или дамп всей базы данных. В этом случае все объекты БД полностью сохраняются.
- *Дифференцированное резервное копирование* БД, в котором копируются только объекты, которые были обновлены или изменены с момента последнего полного резервного копирования.
- *Резервная копия журнала транзакций*, которая создает резервные копии только тех операций журнала, которые не отражены в предыдущей резервной копии БД. В этом случае никакие другие объекты БД не резервируются.

Резервная копия БД хранится в безопасном месте, обычно в другом здании, чем сама БД, и защищена от таких опасностей, как пожар, кража, наводнение и других потенциальных бедствий. Основная цель резервного копирования – гарантировать восстановление БД после аппаратного или программного сбоя.

Сбои, которые мешают работе БД и систем, обычно вызваны программным обеспечением, оборудованием, программными исключениями, транзакциями или внешними факторами. В таблице 11 приведены наиболее распространенные источники сбоя базы данных.

Таблица 11

Распространенные источники сбоя базы данных

Источник	Описание	Пример
Программное обеспечение	Сбои, вызванные программным обеспечением, могут быть связаны с операционной системой, программным обеспечением СУБД, прикладными программами или вирусами и другими вредоносными программами.	В апреле 2017 года в Oracle E-Business Suite была обнаружена новая уязвимость, позволяющая злоумышленнику, не прошедшему проверку подлинности, создавать, изменять или удалять критические данные.
Аппаратные средства	Аппаратные сбои могут включать ошибки микросхемы памяти, сбои диска, поврежденные сектора диска и ошибки переполнения диска.	Неисправный модуль памяти или множественный сбой жесткого диска в системе базы данных может привести к его внезапной остановке.
Программные исключения	Прикладные программы или конечные пользователи могут откатывать транзакции при определенных условиях. Программные исключения также могут быть вызваны злонамеренным или неправильно протестированным кодом, который может быть использован хакерами.	В феврале 2016 года группа неопознанных хакеров обманным путем поручила Федеральному резервному банку Нью-Йорка перевести 81 миллион долларов из центрального банка Бангладеш на счета на Филиппинах. Хакеры использовали мошеннические сообщения, внедренные вредоносными программами, замаскированными для чтения PDF.
Транзакции	Система обнаруживает тупики и прерывает одну из транзакций.	Тупик возникает при выполнении нескольких одновременных транзакций.
Внешние факторы	Резервные копии особенно важны, когда система подвергается полному разрушению в результате пожара, землетрясения, наводнения или другого стихийного бедствия.	В августе 2015 года молния ударила по сетке местного поставщика коммунальных услуг рядом с дата-центрами Google в Бельгии. Несмотря на то, что резервное копирование включалось автоматически, прерывание было достаточно длительным, чтобы вызвать постоянную потерю данных в уязвимых системах.

В зависимости от типа и степени сбоя процесс восстановления может варьироваться от незначительного кратковременного неудобства до серьезного долгосрочного восстановления. Независимо от степени необходимого процесса восстановления, восстановление невозможно без полезной резервной копии.

Восстановление БД обычно происходит по предсказуемому сценарию. Сначала определяются тип и степень необходимого восстановления. Если всю БД необходимо

восстановить до согласованного состояния, для восстановления используется самая последняя резервная копия БД в известном согласованном состоянии. Затем резервная копия переносится для восстановления всех последующих транзакций с использованием информации журнала транзакций. Если требуется восстановить БД, но подтвержденная часть БД по-прежнему пригодна для использования, процесс восстановления использует журнал транзакций для «отмены» всех транзакций, которые не были зафиксированы.

В конце этого этапа база данных завершает итеративный процесс тестирования, оценки и модификации, который продолжается до тех пор, пока система не будет сертифицирована как готовая к вводу в эксплуатацию.

Эксплуатация. Как только БД прошла этап оценки, она считается действующей. На этом этапе БД, ее управление, пользователи и прикладные программы составляют целостную информационную систему.

Начало этапа эксплуатации неизменно запускает процесс эволюции системы. Как только все целевые пользователи вошли в рабочую фазу, проблемы, которые нельзя было предвидеть во время фазы тестирования, начинают появляться. Некоторые из проблем достаточно серьезны, в то время как другие – просто незначительное раздражение. Например, если проект БД реализован для взаимодействия с сетью, сам объем транзакций может привести к зависанию даже хорошо спроектированной системы. В этом случае проектировщики должны определить источник узкого места и выработать альтернативные решения. Эти решения могут включать использование программного обеспечения для балансировки нагрузки для распределения транзакций между несколькими компьютерами, увеличения доступного кэша для СУБД и так далее. Потребность в изменениях является постоянной заботой дизайнера, что приводит к фазе 6, техническому обслуживанию и развитию.

Поддержка и развитие. Администратор БД должен быть готов к выполнению плановых действий по обслуживанию в БД. Некоторые из необходимых периодических работ по техническому обслуживанию включают в себя:

- Профилактическое обслуживание (резервное копирование).
- Корректирующее обслуживание (восстановление).
- Адаптивное обслуживание (повышение производительности, добавление сущностей и атрибутов, и т.д.).
- Назначение прав доступа и их обслуживание для новых и старых пользователей.
- Создание статистики доступа к БД для повышения эффективности и полезности системных аудитов и мониторинга производительности системы.
- Периодические проверки безопасности на основе системной статистики.
- Ежемесячные, квартальные или годовые сводки по использованию системы для внутреннего выставления счетов или составления бюджета.

Вероятность появления новых информационных требований, а также потребность в дополнительных отчетах и новых форматах запросов, требуют изменений в приложении и возможных незначительных изменений в компонентах и содержимом базы данных. Эти

изменения могут быть легко реализованы, только если структура БД является гибкой, и когда вся документация обновляется и доступна. В конце концов, даже самая лучшая среда БД больше не сможет включать такие изменения, и тогда весь процесс ЖЦ БД начинается заново.

6.4. Концептуальное проектирование

Напомним, что второй этап ЖЦ БД – это проектирование базы данных, которая состоит из трех этапов: концептуальное проектирование, логическое проектирование и физическое проектирование, а также критическое решение о выборе СУБД.

Концептуальное проектирование является первым этапом процесса проектирования базы данных. Цель на этом этапе – создать БД, которая не зависит от программного обеспечения и физических данных. Результатом этого процесса является концептуальная модель данных, которая описывает основные сущности, атрибуты, отношения и ограничения данной проблемной области. Этот проект носит описательный и повествовательный характер. Другими словами, он обычно состоит из графического представления, а также текстовых описаний основных элементов данных, отношений и ограничений.

На этом этапе используется моделирование данных для создания абстрактной структуры БД, которая представляет объекты реального мира наиболее реалистичным способом. Концептуальная модель должна заключать в себе четкое понимание предметной области. На этом уровне абстракции тип используемого оборудования и модели БД, возможно, еще не определены. Поэтому проект должен быть независимым от программного и аппаратного обеспечения, чтобы система могла быть настроена на любой платформе, выбранной позже.

Правило минимальных данных: *Все, что нужно, там, и все, что там, нужно.*

Другими словами, нужно убедиться, что все необходимые данные находятся в модели и что все данные в модели необходимы. Все элементы данных, требуемые транзакциями БД, должны быть определены в модели, а все элементы данных, определенные в модели, должны использоваться хотя бы одной транзакцией БД.

Однако, применяя правило минимальных данных, нужно избегать чрезмерного краткосрочного смещения. Необходимо сосредоточиться не только на текущих, но и на будущих данных. Таким образом, проект базы данных должен оставить место для будущих модификаций и дополнений, гарантирующих, что информационные ресурсы будут увеличиваться.

Концептуальное проектирование состоит из четырех этапов, которые перечислены в таблице 12.

Таблица 12

Этапы концептуального проектирования

Этап	Действие
1	Анализ данных и требований.
2	Моделирование и нормализация связей между сущностями.
3	Верификация модели данных.
4	Проектирование распределенных базы данных.

Анализ данных и требований. Первым шагом в концептуальном дизайне является выявление свойств элементов данных. Эффективная БД – информационная фабрика, которая производит ключевые компоненты для успешного принятия решений. Подходящие свойства элементов данных могут быть преобразованы в соответствующую информацию. Поэтому усилия разработчика направлены на:

- *Информационные потребности.* Какая информация нужна? Какие выходные данные (отчеты и запросы) должны генерироваться системой, какую информацию генерирует текущая система, и в какой степени эта информация адекватна?
- *Пользователи информации.* Кто будет использовать информацию? Как информация будет использоваться? Каковы различные представления данных пользователя?
- *Источники информации.* Где найти информацию? Как получить информацию, когда она найдена?
- *Состав информации.* Какие элементы данных необходимы для получения информации? Каковы атрибуты данных? Какие связи существуют в данных? Какой объем данных? Как часто используются данные? Какие преобразования данных будут использоваться для генерации необходимой информации?

Разработчик получает ответы на эти вопросы из разных источников, чтобы собрать необходимую информацию:

- *Разработка и сбор представлений данных.* Разработчик БД и пользователь совместно разрабатывают точное описание представлений данных, которые, в свою очередь, используются для определения основных элементов БД.
- *Непосредственное наблюдение за текущей системой: существующий и желаемый результат.* Пользователь обычно имеет существующую систему, будь то ручная или компьютерная. Разработчик анализирует существующую систему, чтобы определить данные и ее свойства. Он проверяет входные формы и файлы (таблицы), чтобы определить тип и объем данных. Если пользователь уже имеет автоматизированную систему, разработчик тщательно изучает текущие и требуемые отчеты, чтобы описать данные, необходимые для поддержки отчетов.
- *Взаимодействие с группой разработчиков ПО.* Процесс проектирования БД является частью ЖЦ ИС. В некоторых случаях системный аналитик, отвечающий за разработку новой системы, также разрабатывает концептуальную модель БД. В других случаях проектирование БД считается частью работы администратора БД. Наличие АБД обычно подразумевает наличие формального отдела обработки данных. АБД проектирует базу данных в соответствии со спецификациями, созданными системным аналитиком.

Чтобы разработать точную модель данных, разработчик должен иметь полное представление о типах данных, их объеме и использовании. Но данные сами по себе не дают необходимого понимания всего процесса. С точки зрения БД, сбор данных имеет смысл только тогда, когда определены бизнес-правила. Бизнес-правило – это краткое и точное описание политики, процедуры или принципов в среде конкретной организации. Бизнес-правила,

основанные на подробном описании операций организации, помогают создавать и обеспечивать выполнение действий в среде этой организации. Когда бизнес-правила написаны правильно, они определяют сущности, атрибуты, отношения, связи, мощности и ограничения.

Чтобы быть эффективными, бизнес-правила должны быть простыми для понимания и широко распространяться, чтобы гарантировать, что каждый сотрудник организации разделяет общую интерпретацию правил. Используя простой язык, бизнес-правила описывают основные и отличительные характеристики данных с точки зрения компании. Примеры бизнес-правил:

• Клиент может сделать много платежей на счет.

• Каждый платеж на счету зачисляется только одному клиенту.

• Клиент может генерировать много счетов.

• Каждый счет генерируется только одним клиентом.

Учитывая их важную роль в разработке БД, бизнес-правила не должны устанавливаться случайно. Неправильно определенные или неточные бизнес-правила приводят к проектированию и реализации БД, которые не отвечают потребностям конечных пользователей организации.

В идеале бизнес-правила основаны на формальном описании операций, которое представляет собой документ, содержащий точное, обновленное и тщательно проанализированное описание действий, определяющих операционную среду организации. Естественно, операционная среда организации зависит от миссии организации. Например, операционная среда университета будет сильно отличаться от среды производителя стали, авиакомпании или дома престарелых. Тем не менее, какими бы разными ни были организации, компонент анализа данных и требований в структуре БД улучшается, когда среда данных и использование данных описываются точно в описании операций.

В бизнес-среде основными источниками информации для описания операций являются руководители компаний, политики, руководители отделов и письменная документация, такая как процедуры, стандарты и руководства по эксплуатации компании. Более быстрым и прямым источником бизнес-правил являются интервью с пользователями. К сожалению, из-за различий в восприятии пользователь может быть менее надежным источником при определении бизнес-правил. Например, механик отдела техобслуживания может полагать, что любой механик может инициировать процедуру техобслуживания, когда на самом деле такую задачу должны выполнять только механики с разрешением на проверку. Это различие может показаться тривиальным, но оно имеет серьезные правовые последствия. Хотя пользователи вносят решающий вклад в разработку бизнес-правил, стоит проверить их восприятие. Зачастую собеседования с несколькими людьми, выполняющими одну и ту же работу, дают совершенно разные представления об их рабочих компонентах. Хотя такое открытие может указывать на «проблемы управления», эта общая диагностика не помогает разработчику БД. Учитывая обнаружение таких проблем, задача разработчика БД состоит в том, чтобы согласовать различия и проверить результаты сверки, чтобы убедиться, что бизнес-правила являются правильными и точными.

Знание бизнес-правил позволяет разработчику полностью понять, как работает бизнес и какую роль играют данные в деятельности компании. Следовательно, разработчик должен определить бизнес-правила компании и проанализировать их влияние на характер, роль и объем данных.

Бизнес-правила дают несколько важных преимуществ при разработке новых систем:

- Они помогают стандартизировать представление компании о данных.
- Они представляют собой инструмент связи между пользователями и дизайнерами.
- Они позволяют разработчику понять природу, роль и объем данных.
- Они позволяют разработчику понимать бизнес-процессы.
- Они позволяют разработчику сформировать соответствующие правила участия в связях и ограничения внешнего ключа.

Последний пункт особенно примечателен: является ли данное отношение обязательным или необязательным, обычно является функцией применимого бизнес-правила.

Моделирование и нормализация связей между сущностями. Перед созданием модели ER разработчик должен изучать и внедрить соответствующие стандарты, которые будут использоваться в документации проекта. Стандарты включают использование диаграмм и символов, стиля написания документации, макета и любых других соглашений, которым необходимо следовать при документировании. Разработчики часто игнорируют это очень важное требование, особенно когда они работают в команде. Неспособность стандартизировать документацию часто означает неспособность общаться, а сбои связи часто приводят к плохой проектной работе. Напротив, четко определенные и соблюдаемые стандарты облегчают проектную работу и обещают (но не гарантируют) плавную интеграцию всех компонентов системы.

Поскольку бизнес-правила обычно определяют характер связей между сущностями, разработчик должен включить их в концептуальную модель. Процесс определения бизнес-правил и разработки концептуальной модели с использованием ER-диаграмм можно описать с помощью шагов, показанных в таблице 13.

Таблица 13

Разработка концептуальной модели с использованием ER диаграмм

Этап	Действие
1	Определить, проанализировать и уточнить бизнес-правила.
2	Определить основные сущности, используя результаты шага 1.
3	Определить связи между сущностями, используя результаты шагов 1 и 2.
4	Определить атрибуты, первичные и внешние ключи для каждой сущности.
5	Нормализовать сущности.
6	Заполнить исходную диаграмму ER.
7	Проверка модели ER на соответствие требованиям пользователей к информации и обработке.
8	Изменить модель ER, используя результаты шага 7.

Некоторые из этапов, перечисленных в таблице 13, выполняются одновременно, а некоторые, такие как процесс нормализации, могут генерировать потребность в дополнительных сущностях или атрибутах, в результате чего разработчик пересматривает модель ER.

Все объекты (сущности, атрибуты, отношения, представления и т.д.) определены в словаре данных, который используется в тандеме с процессом нормализации, чтобы помочь устранить аномалии данных и проблемы избыточности. Во время этого процесса моделирования ER разработчик должен:

- Определить сущности, атрибуты, первичные и внешние ключи.
- Принимать решения о добавлении новых атрибутов первичного ключа для удовлетворения требований пользователя и обработки.
- Принимать решения об обработке составных и многозначных атрибутов.
- Принимать решения о добавлении производных атрибутов для удовлетворения требований обработки.
- Принимать решения о размещении внешних ключей в связях 1:1.
- Избегать ненужных тернарных связей.
- Готовить соответствующую диаграмму ER.
- Нормализовать сущности.
- Включить все определения элементов данных в словарь данных.
- Принимать решения о стандартных соглашениях об именах.

Важным является требование к соглашениям об именах, однако оно часто игнорируется на риск разработчика. Реальное проектирование БД обычно выполняется командами. Поэтому важно убедиться, что члены команды работают в среде, в которой стандарты именования определены и применяются. Надлежащая документация имеет решающее значение для успешного завершения проектирования, и соблюдение соглашений об именах хорошо помогает разработчикам БД.

Верификация модели данных. Верификация модели данных является одним из последних этапов на этапе концептуального проектирования и является одним из наиболее важных. На этом этапе модель ER должна быть проверена на соответствие предлагаемым системным процессам, чтобы подтвердить, что они могут поддерживаться моделью БД. Проверка требует, чтобы модель прошла серию тестов на:

- Представления данных пользователя.
- Все необходимые транзакции: операции SELECT, INSERT, UPDATE и DELETE.
- Права доступа и безопасность.
- Деловые требования к данным и ограничения.

Поскольку реальное проектирование БД обычно выполняется командами, проектирование БД, вероятно, разделено на основные компоненты, известные как модули. *Модуль* – компонент информационной системы, который обрабатывает определенные

функции, такие как инвентаризация, заказы или расчет заработной платы. В этих условиях каждый модуль поддерживается сегментом ER, который является подмножеством или фрагментом корпоративной модели ER. Работа с модулями выполняет несколько важных задач:

- Модули могут быть делегированы группам разработчиков внутри команд, что значительно ускоряет разработку.
- Модули упрощают проектные работы. Каждый модуль содержит более управляемое количество объектов.
- Модули могут быть быстро созданы. Проблемные области реализации и прикладного программирования могут быть определены более легко.
- Даже если всю систему невозможно быстро реализовать, внедрение одного или нескольких модулей покажет, что достигнут определенный прогресс и что, по крайней мере, часть системы готова начать обслуживание пользователей.

Как бы ни были полезны модули, они представляют собой свободную коллекцию фрагментов модели ER, которые могут привести к хаосу в базе данных, если их не остановить. Например, фрагменты модели ER:

- Может представлять перекрывающиеся, дублированные или противоречивые представления одних и тех же данных.
- Может не поддерживать все процессы в модулях системы.

Чтобы избежать этих проблем, лучше объединить фрагменты ER модулей в единую корпоративную модель ER. Этот процесс начинается с выбора центрального сегмента и итеративного добавления фрагментов. На каждом этапе для каждой новой сущности, добавляемой в модель, необходимо проверять, чтобы новая сущность не перекрывалась и не конфликтовала с ранее идентифицированной сущностью в корпоративной модели ER.

Объединение сегментов модели ER в корпоративную модель ER вызывает тщательную переоценку сущностей с последующим подробным изучением атрибутов, которые описывают эти сущности. Этот процесс служит нескольким важным целям:

- Появление деталей атрибута может привести к пересмотру самих сущностей. Возможно, некоторые из компонентов, которые первоначально считались сущностями, вместо этого окажутся атрибутами внутри других сущностей. Или компонент, который первоначально считался атрибутом, может содержать достаточное количество подкомпонентов, чтобы гарантировать введение одной или нескольких новых сущностей.
- Сосредоточение на деталях атрибута может дать подсказки о природе связей, поскольку они определяются первичным и внешним ключами. Неправильно определенные связи вначале приводят к проблемам с реализацией, а затем к проблемам разработки приложений.
- Чтобы удовлетворить требования обработки и пользователя, может быть полезно создать новый первичный ключ для замены существующего первичного ключа.

- Если детали сущности (атрибуты и их характеристики) не определены точно, трудно оценить степень нормализации проекта. Знание уровней нормализации помогает избежать нежелательных уменьшений.
- Тщательный анализ примерного проекта БД может привести к пересмотру. Эти изменения помогут обеспечить соответствие проекта требованиям пользователя.

После завершения процесса слияния полученная в результате корпоративная модель ER сверяется с каждым из процессов модуля. Процесс проверки модели ER подробно описан в таблице 14:

Таблица 14

Процесс верификации ER модели

Этап	Действие
1	Определить центральную сущность модели ER.
2	Определить каждый модуль и его компоненты.
3	Определить требования к операциям каждого модуля: Внутренний: обновления / добавления / удаления / запросы / отчеты Внешний: интерфейс модулей
4	Проверить все процессы на соответствие требованиям модуля к обработке и отчетности.
5	Внести все необходимые изменения, предложенные в шаге 4.
6	Повторить шаги 2–5 для всех модулей.

Процесс требует постоянной проверки бизнес-транзакций, а также системных и пользовательских требований. Последовательность проверки должна быть повторена для каждого из модулей системы.

Процесс проверки начинается с выбора центральной (наиболее важной) сущности, на которой сосредоточено большинство операций системы.

Чтобы идентифицировать центральную сущность, разработчик выбирает сущность, участвующую в наибольшем числе связей. На диаграмме ER это сущность, к которой подключено больше линий, чем к любой другой.

Следующим шагом является определение модуля или подсистемы, к которой принадлежит центральная сущность, и определение границ и области действия этого модуля. Сущность принадлежит модулю, который использует ее чаще всего. Как только каждый модуль идентифицирован, центральная сущность помещается в структуру модуля, чтобы разработчик мог сосредоточиться на деталях модуля.

В рамках центральной сущности/модуля разработчик должен:

- *Обеспечить когезивность модуля.* Термин *когезивность* описывает силу связи, обнаруживаемой между сущностями модуля. Модуль должен отображать высокую когезивность, то есть сущности должны быть тесно связаны, а модуль должен быть полным и самодостаточным.
- *Анализировать связи каждого модуля с другими модулями, чтобы решить проблему связывания модулей. Связывание модулей* описывает степень, в которой

модули независимы друг от друга. Модули должны отображать низкую связь, указывая, что они не зависят от других модулей. Низкая связь уменьшает ненужные межмодульные зависимости, что позволяет создать действительно модульную систему и устранить ненужные связи между сущностями.

Процессы могут быть классифицированы в соответствии с их:

- частотой (ежедневно, еженедельно, ежемесячно, ежегодно или исключения);
- типом операции (INSERT или ADD, UPDATE или CHANGE, DELETE, запросы и отчеты, пакеты, обслуживание и резервные копии).

Все идентифицированные процессы должны быть проверены по модели ER. При необходимости, соответствующие изменения осуществляются. Проверка процесса повторяется для всех модулей модели. Можно ожидать, что дополнительные сущности и атрибуты будут включены в концептуальную модель во время ее проверки.

На этом этапе концептуальная модель была определена как аппаратно-программная независимая. Такая независимость обеспечивает переносимость системы между платформами. Переносимость может продлить срок службы БД, сделав возможным переход на другую СУБД и аппаратную платформу.

Проектирование распределенных баз данных. Хотя это и не является обязательным требованием для большинства БД, некоторые из них могут быть распределены по географически разрозненным местам. Процессы, которые обращаются к БД, также могут варьироваться от одного местоположения к другому. Например, процессы розничной торговли и процессы складского хранения, вероятно, находятся в разных физических местах. Если данные и процессы БД будут распределены по всей системе, фрагменты БД, могут находиться в нескольких физических местах. *Фрагмент базы данных* – подмножество БД, которое хранится в данном месте. Фрагмент БД может быть подмножеством строк или столбцов из одной или нескольких таблиц.

Структура распределенной БД определяет оптимальную стратегию размещения фрагментов БД для обеспечения целостности, безопасности и производительности. Стратегия распределения определяет, как разделить БД и где хранить каждый фрагмент.

6.5. Выбор программного обеспечения СУБД

Выбор программного обеспечения СУБД имеет решающее значение для бесперебойной работы информационной системы. Следовательно, достоинства и недостатки предлагаемого программного обеспечения СУБД должны быть тщательно изучены. Чтобы избежать ложных ожиданий, пользователь должен быть осведомлен об ограничениях как СУБД, так и БД.

Хотя факторы, влияющие на решение о покупке, варьируются от компании к компании, некоторые из наиболее распространенных:

- *Стоимость.* Включает в себя первоначальную стоимость покупки, а также расходы на техническое обслуживание, эксплуатацию, лицензию, установку, обучение и конверсию.

- *Особенности и инструменты СУБД.* Некоторое программное обеспечение базы данных включает в себя множество инструментов, которые облегчают разработку приложений. Средства администрирования БД, средства запросов, простота использования, производительность, безопасность, контроль параллелизма, обработка транзакций и поддержка сторонних производителей влияют на выбор программного обеспечения СУБД.
- *Базовая модель.* Это может быть иерархическая, сетевая, реляционная, объектно-реляционная или объектно-ориентированная.
- *Портативность.* СУБД может быть переносимой на разные платформы, системы и языки.
- *Требования к оборудованию СУБД.* Элементы, которые следует учитывать, включают процессор(ы), оперативную память, дисковое пространство и т.д.

6.6. Логическое проектирование

Логическое проектирование является вторым этапом в процессе проектирования БД. Целью логического проектирования является создание БД в масштабах предприятия, основанной на конкретной модели данных, но независящей от деталей физического уровня. Логическое проектирование требует, чтобы все сущности в концептуальной модели были сопоставлены с конкретными конструкциями, используемыми выбранной моделью БД. Например, логическое проектирование для реляционной СУБД включает в себя спецификации для таблиц, связей и ограничений.

Логическое проектирование обычно выполняется в четыре этапа, которые перечислены в таблице 15.

Таблица 15

Этапы логического проектирования

Этап	Действие
1	Сопоставить концептуальную модель с компонентами логической модели.
2	Подтвердить логическую модель с помощью нормализации.
3	Подтвердить ограничения целостности логической модели.
4	Проверить логическую модель на соответствие требованиям пользователя.

Такие этапы, как и большинство процессов моделирования данных, не обязательно выполняются последовательно, но итеративно.

Сопоставить концептуальную модель с логической моделью. Первым шагом в создании логического проекта является сопоставление концептуальной модели с выбранными конструкциями БД. В реальном мире логическое проектирование обычно включает перевод модели ER в набор таблиц, столбцов и ограничений.

Первым шагом на этапе логического проектирования является сопоставление сильных сущностей с таблицами. Сильная сущность – это сущность, которая находится на стороне «1»

всех его связей, то есть сущность, у которой нет обязательного атрибута, являющегося внешним ключом для другой таблицы.

Как только все сильные сущности сопоставлены, можно отобразить любые сущности, вовлеченные в связях супертип/подтип, или любые слабые сущности.

Затем отображаются все бинарные связи. Процесс продолжается со связями между тремя или более сущностями, пока все связи в модели не будут четко определены. Конечным результатом этого процесса является список таблиц, атрибутов и связей, которые станут основой для следующего шага.

Проверка логической модели с помощью нормализации. Логический проект должен содержать только правильно нормализованные таблицы. Процесс сопоставления концептуальной модели с логической моделью может раскрыть некоторые новые атрибуты или открытие новых многозначных или составных атрибутов. Следовательно, очень вероятно, что новые атрибуты могут быть добавлены в таблицы или что целые новые таблицы могут быть добавлены в логическую модель. Для каждой идентифицированной таблицы (старой и новой) разработчик должен убедиться, что все атрибуты полностью зависят от первичного ключа, и таблицы имеют как минимум третью нормальную форму (3НФ).

Проектирование базы данных является итеративным процессом. Такие действия, как нормализация происходят на разных этапах процесса проектирования. Каждый раз, когда повторяется шаг, модель дополнительно уточняется и лучше документируется. Новые атрибуты могут быть созданы и назначены соответствующим объектам. Функциональные зависимости между детерминантами и зависимыми атрибутами оцениваются, и аномалии данных предотвращаются посредством нормализации.

Проверка ограничений целостности логической модели. Перевод концептуальной модели в логическую модель также требует определения областей атрибутов и соответствующих ограничений. На этом этапе определяется, какие атрибуты являются обязательными, а какие – необязательными, и гарантируется, что все сущности поддерживают целостность сущностей и ссылок.

Право на использование базы данных также указывается на этапе логического проектирования. Кому будет разрешено использовать таблицы, и какие части таблиц будут доступны для каких пользователей? В реляционных рамках ответы на эти вопросы требуют определения соответствующих взглядов.

На этом этапе требуется особое внимание, чтобы обеспечить возможность разрешения всех представлений и обеспечения безопасности для обеспечения конфиденциальности данных. Кроме того, в проекте распределенной базы данных, данные могут храниться в нескольких местах, и в каждом месте могут быть разные ограничения безопасности.

Проверка логической модели на соответствие требованиям пользователя. Логическое проектирование переводит программно-независимую концептуальную модель в программно-зависимую модель. Последний шаг в процессе логического проектирования заключается в проверке всех определений логической модели по всем требованиям к данным, транзакциям и безопасности пользователя. Процесс, аналогичный показанному в таблице 14,

повторяется для обеспечения правильности логической модели. Теперь этап настроен на определение физических требований, которые позволяют системе функционировать в выбранной среде СУБД / аппаратное обеспечение.

6.7. Физическое проектирование

Физическое проектирование – это процесс определения организации хранения данных и характеристик доступа к базе данных для обеспечения ее целостности, безопасности и производительности. Это последний этап в процессе проектирования базы данных. Характеристики хранения являются функцией типов устройств, поддерживаемых аппаратным обеспечениям, типов методов доступа к данным, поддерживаемых системой и СУБД. Физическое проектирование может стать очень технической задачей, которая влияет не только на доступность данных в запоминающем устройстве(ах), но и на производительность системы.

Этап физического проектирования состоит из шагов в таблице 16.

Таблица 16

Этапы физического проектирования

Этап	Действие
1	Определить организацию хранения данных.
2	Определить меры целостности и безопасности.
3	Определить измерения производительности.

Определение организации хранения данных. Прежде чем определить организацию хранения данных, необходимо определить объем данных для управления и шаблоны использования данных.

- Знание объема данных поможет определить, сколько места для хранения нужно зарезервировать для БД. Для этого разработчик следует процессу, аналогичному тому, который использовался при проверке модели ER. Для каждой таблицы нужно определить все возможные транзакции, их частоту и объем. Для каждой транзакции определяется объем данных, которые будут добавлены или удалены из базы данных. Эта информация поможет определить объем данных, которые будут сохранены в соответствующей таблице.
- И наоборот, знание того, как часто новые данные вставляются, обновляются и извлекаются, поможет разработчику определить шаблоны использования данных. Шаблоны использования имеют решающее значение, особенно в проектировании распределенных БД. Например, нужно ли генерировать какие-либо еженедельные пакетные загрузки или ежемесячные отчеты об агрегации? Как часто новые данные добавляются в систему?

Оборудованный двумя предыдущими порциями информации, разработчик должен:

- *Определить местоположение и физическую организацию хранения для каждой таблицы.* Таблицы хранятся в табличных пространствах, и табличное пространство может содержать данные из нескольких таблиц. На этом этапе разработчик

назначает, какие таблицы будут использовать какие табличные пространства, и назначает расположение этих табличных пространств. Например, полезным методом, доступным в большинстве реляционных баз данных, является использование кластерных таблиц. *Кластерные таблицы* хранят связанные строки из двух связанных таблиц в смежных блоках данных на диске. Это гарантирует, что данные хранятся в последовательно смежных местах, что сокращает время доступа к данным и повышает производительность системы.

- *Определить индексы и тип индексов, которые будут использоваться для каждой таблицы.* Индексы полезны для обеспечения уникальности значений данных в столбце и для облегчения поиска данных. СУБД автоматически создает уникальный индекс для первичного ключа каждой таблицы. На этом этапе разработчик определяет все необходимые индексы и лучший тип организации для использования данных и требований к производительности.
- *Определить тип представлений, которые будут использоваться в каждой таблице.* Представление полезно для ограничения доступа к данным на основе потребностей пользователей или транзакций. Представления также могут использоваться для упрощения обработки и доступа к данным пользователя. На этом этапе разработчик должен убедиться, что все представления могут быть реализованы, и что они предоставляют только необходимые данные. Разработчик также должен ознакомиться с типами представлений, поддерживаемыми СУБД, и тем, как они могут помочь в достижении системных целей.

Определение меры целостности и безопасности. Как только физическая организация таблиц, индексов и представлений определена, база данных готова для пользователей. Прежде чем пользователи смогут получить доступ к данным в БД, они должны быть надлежащим образом аутентифицированы. На этом этапе физического проектирования необходимо решить две задачи:

- *Определить группы пользователей, безопасность и роли.* Управление пользователями – скорее функция администрирования БД, чем проектирования. Однако, разработчик должен знать разные типы и группы пользователей, чтобы должным образом обеспечить безопасность БД. Большинство реализаций СУБД поддерживает использование ролей. *Роль базы данных* – набор привилегий, которые можно назначить пользователю или группе как единому целому.
- *Назначить меры безопасности.* СУБД также позволяет администраторам назначать определенные права доступа для объектов базы данных пользователю или группе пользователей. Право доступа также может быть отозвано у определенного пользователя или группы пользователей. Эта функция может пригодиться во время резервного копирования БД, запланированных мероприятий по обслуживанию или даже во время инцидентов с утечкой данных.

Определение показателей эффективности. Физическое проектирование становится более сложным, когда данные распространяются в разных местах, поскольку

производительность зависит от пропускной способности коммуникационных сред. Учитывая такие сложности, неудивительно, что разработчики предпочитают программное обеспечение для БД, которое скрывает как можно больше действий на физическом уровне. Несмотря на то, что реляционные модели имеют тенденцию скрывать сложности физических характеристик компьютера, на производительность реляционных БД влияют свойства физического хранилища. Например, на производительность могут влиять характеристики носителя, такие как время поиска, размер сектора и блока (страницы), размер пула буферов, а также количество дисковых пластин и головок чтения/записи. Кроме того, такие факторы, как создание индекса могут оказать значительное влияние на производительность реляционной БД, то есть скорость и эффективность доступа к данным.

Таким образом, измерение производительности физического проектирования касается тонкой настройки СУБД и запросов, чтобы гарантировать, что они будут соответствовать требованиям производительности пользователя.

6.8. Стратегии проектирования баз данных

Существует два классических подхода к проектированию баз данных:

- *Нисходящее проектирование* начинается с идентификации наборов данных, а затем определяет элементы данных для каждого из этих наборов. Этот процесс включает в себя идентификацию разных типов сущностей и определение атрибутов каждой сущности.
- *Восходящее проектирование* сначала идентифицирует элементы данных, а затем группирует их в наборы данных. Другими словами, он сначала определяет атрибуты, а затем группирует их для формирования сущностей.

Эти две методологии являются взаимодополняющими, а не взаимоисключающими, основной акцент на восходящем подходе может быть более продуктивным для небольших БД с небольшим количеством сущностей, атрибутов, связей и транзакций. Для ситуаций, в которых количество, разнообразие и сложность сущностей, отношений и транзакций являются подавляющими, в первую очередь может быть проще подход сверху вниз. У большинства компаний уже есть стандарты для разработки систем и проектирования БД.

Даже при выборе в основном нисходящего подхода процесс нормализации, который пересматривает существующие структуры таблиц, неизбежно является восходящим методом. Модели ER составляют нисходящий процесс, даже когда выбор атрибутов и сущностей может быть описан как восходящий. Поскольку как модель ER, так и методы нормализации составляют основу для большинства проектов, дебаты «сверху-вниз» и «снизу-вверх» могут основываться на теоретических, а не на фактических различиях.

Два общих подхода к проектированию базы данных (снизу-вверх и сверху-вниз) могут зависеть от таких факторов, как объем и размер системы, стиль управления и структура компании (централизованная или децентрализованная). В зависимости от этих факторов проект БД может основываться на двух очень разных принципах проектирования: централизованном и децентрализованном.

Централизованное проектирование является продуктивным, когда компонент данных имеет относительно небольшое количество сущностей и процедур. Проект может быть выполнен и представлен в довольно простой базе данных. Централизованное проектирование типично для относительно простых, небольших БД, и может быть успешно выполнено одним администратором БД, или небольшой, неформальной командой разработчиков. Деятельность компании и масштабы проблемы достаточно ограничены, чтобы позволить даже одному разработчику определить проблему, создать концептуальную модель, проверить ее с помощью пользовательских представлений, определить системные процессы и ограничения данных для обеспечения эффективности дизайна, и убедиться, что проект будет соответствовать всем требованиям.

Децентрализованное проектирование может использоваться, когда компонент данных системы имеет значительное количество сущностей и сложные связи, над которыми выполняются очень сложные операции. Децентрализованный дизайн также часто используется, когда сама проблема распространяется на несколько рабочих мест, и каждый элемент является подмножеством всего набора данных.

В больших и сложных проектах база данных обычно не может быть разработана только одним человеком. Вместо этого тщательно отобранная команда разработчиков БД занимается сложным проектом базы данных. В рамках децентрализованной структуры проектирования задача разделена на несколько модулей. После того, как критерии проектирования были установлены, ведущий разработчик назначает модули для разработки внутри группы.

Поскольку каждая группа разработки фокусируется на моделировании подмножества системы, определение границ и взаимосвязи между подмножествами данных должно быть очень точным. Каждая группа разработки создает концептуальную модель данных, соответствующую моделируемому подмножеству. Затем каждая концептуальная модель проверяется индивидуально с учетом пользовательских представлений, процессов и ограничений для каждого из модулей. После завершения процесса проверки все модули объединяются в одну концептуальную модель. Поскольку словарь данных описывает характеристики всех сущностей в концептуальной модели данных, он играет жизненно важную роль в процессе интеграции. После объединения подмножеств в более крупную концептуальную модель ведущий разработчик должен убедиться, что он по-прежнему может поддерживать все необходимые транзакции.

Процесс агрегирования требует, чтобы разработчик создал единую модель, в которой должны быть решены различные проблемы агрегирования.

- *Синонимы и омонимы.* Разные отделы могут знать одну и ту же сущность под разными именами (синонимами) или могут использовать одно и то же имя для адресации разных сущностей (омонимов).
- *Сущность и подтипы сущности.* Подтип сущности может рассматриваться как отдельная сущность одним или несколькими отделами. Разработчик должен интегрировать такие подтипы в сущность более высокого уровня.

- *Конфликтующие определения объектов.* Атрибуты могут быть записаны как разные типы (символьные, числовые), разные домены могут быть определены для одного и того же атрибута. Определения ограничений также могут различаться. Разработчик должен удалить такие конфликты из модели.

6.9. Итоги

- Информационная система призвана помочь преобразовать данные в информацию и управлять как данными, так и информацией. Таким образом, база данных является очень важной частью информационной системы. Системный анализ устанавливает потребность в информационной системе и ее объем. Разработка систем – процесс создания информационной системы.
- Жизненный цикл информационной системы (ЖЦ ИС) отслеживает историю приложения в информационной системе. ЖЦ ИС можно разделить на пять этапов: постановка задачи, анализ требований, проектирование, разработка и внедрение. ЖЦ ИС – это итеративный, а не последовательный процесс.
- Жизненный цикл базы данных (ЖЦ БД) описывает историю базы данных в информационной системе. ЖЦ БД состоит из шести этапов: изучение среды базы данных, проектирование, реализация, тестирование и настройка, эксплуатация, обслуживание и развитие. Как и ЖЦ ИС, ЖЦ БД является итеративным, а не последовательным.
- Концептуальная часть проекта может подвергаться нескольким вариациям, основанным на двух основных принципах проектирования: восходящий против нисходящего, и централизованный против децентрализованного.

Лекция 7. Управление транзакциями

7.1. Введение

Анализируя реляционную диаграмму на рисунке 25, необходимо обратить внимание на следующие особенности:

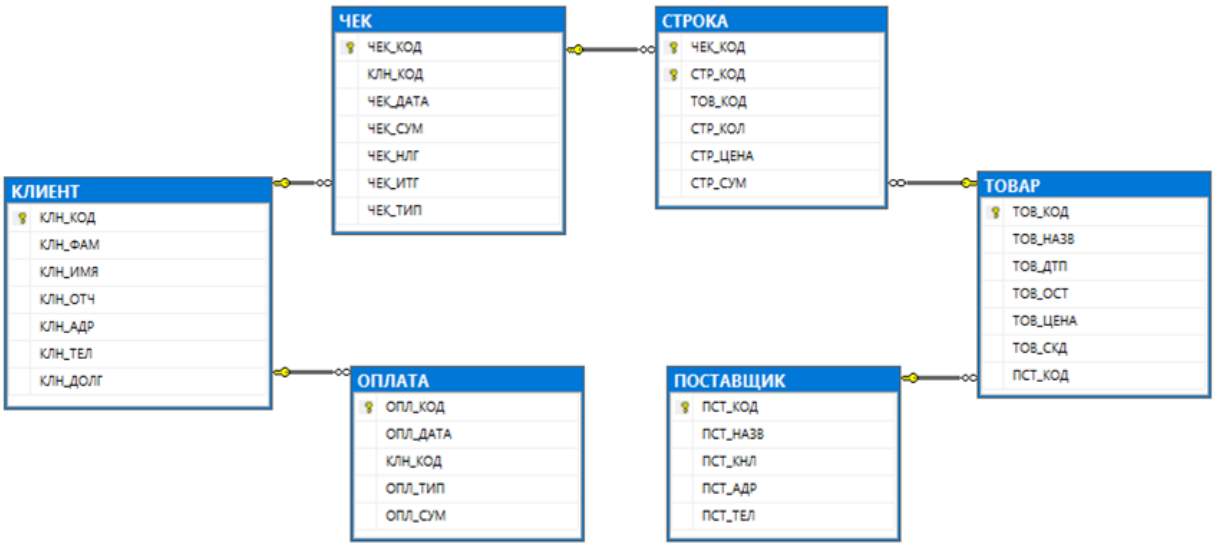


Рис. 25. Диаграмма базы данных ТОРГОВЛЯ

- Проект сохраняет баланс клиента (КЛН_ДОЛГ) в таблице КЛИЕНТ, чтобы указать общий долг клиента. Атрибут КЛН_ДОЛГ увеличивается, когда клиент совершает покупку в кредит, и уменьшается, когда клиент совершает платеж. Включение текущего долга клиента в таблицу КЛИЕНТ позволяет написать запрос для определения долга любого клиента и создания важных сводок, таких как итоговые, средние, минимальные и максимальные остатки.
- В таблице ОПЛАТА записываются все платежи клиентов, чтобы отслеживать детали операций по учетным записям клиентов.

Продажа товара клиенту по кредиту состоит из следующих шагов:

- Создать новый чек с типом "кредит".
- Уменьшить количество товара.
- Увеличить долг клиента

Операция продажи должна быть отражена в базе данных. С точки зрения базы данных, транзакция – это любое действие, которое читает или записывает в базу данных. Транзакция может состоять из следующих частей:

- Оператор SELECT для генерации списка содержимого таблицы.
- Серия связанных операторов UPDATE для изменения значений атрибутов в разных таблицах.
- Серия операторов INSERT для добавления строк в одну или несколько таблиц.

- Комбинация операторов SELECT, UPDATE и INSERT.

Пример транзакции продажи включает комбинацию операторов INSERT и UPDATE.

Учитывая предыдущее обсуждение, можно дополнить определение транзакции. *Транзакция* – это логическая единица работы, которая должна быть полностью завершена или полностью прервана; промежуточные состояния не принимаются. Другими словами, многокомпонентная транзакция, такая как ранее упомянутая продажа, не должна быть частично завершена. Обновление только товаров или только задолженности недопустимо. Все операторы SQL в транзакции должны быть успешно выполнены. Если какой-либо из операторов SQL не выполняется, вся транзакция возвращается к исходному состоянию базы данных, которое существовало до ее начала. Успешная транзакция переводит базу данных из одного согласованного состояния в другое. *Согласованное состояние базы данных* – это состояние, в котором все ограничения целостности данных выполняются.

Чтобы обеспечить согласованность базы данных, каждая транзакция должна начинаться с базы данных в известном согласованном состоянии. Если база данных не находится в согласованном состоянии, транзакция приведет к несовместимости базы данных, которая нарушает ее целостность. По этой причине, с учетом ограничений, которые будут обсуждаться позже, все транзакции контролируются и выполняются СУБД, чтобы гарантировать целостность базы данных.

Большинство реальных транзакций базы данных формируются двумя или более запросами базы данных. Запрос к базе данных является эквивалентом одного оператора SQL в прикладной программе или транзакции. Например, если транзакция состоит из двух операторов UPDATE и одного оператора INSERT, транзакция использует три запроса к базе данных. В свою очередь, каждый запрос к базе данных генерирует несколько операций ввода/вывода (I/O), которые читают или записывают на физический носитель.

Оценка результатов транзакции. Не все транзакции обновляют базу данных. Например, изучение таблицы КЛИЕНТ, чтобы определить текущий долг для клиента с кодом 4591 может быть с помощью следующего кода SQL:

```
SELECT КЛН_КОД, КЛН_ДОЛГ
FROM КЛИЕНТ
WHERE КЛН_КОД = 4591;
```

Хотя запрос не вносит никаких изменений в таблицу КЛИЕНТ, код SQL представляет транзакцию, поскольку он обращается к базе данных. Если база данных находилась в согласованном состоянии до доступа, база данных остается в согласованном состоянии после доступа, поскольку транзакция не изменила базу данных.

Таблица: ЧЕК

ЧЕК_КОД	КЛН_КОД	ЧЕК_ДАТА	ЧЕК_СУМ	ЧЕК_СКД	ЧЕК_ИТГ	ЧЕК_ТИП
35647	4587	2020-07-20	80000	9020	70980	нал.
35648	4588	2020-07-25	72000	7500	64500	безнал.
35649	4589	2020-08-02	102500	10000	92500	кредит
35650	4595	2020-08-03	725200	72500	652700	кредит
35651	4591	2020-08-03	18000	0	18000	кредит

Таблица: СТРОКА

ЧЕК_КОД	СТР_КОД	ТОВ_КОД	СТР_КОЛ	СТР_ЦЕНА	СТР_СУМ
35647	1	65343	1	80000	80000
35648	1	65335	1	50000	50000
35648	2	65336	1	22000	22000
35649	1	65330	1	52500	52500
35649	2	65335	1	50000	50000
35650	1	65343	2	80000	160000
35650	2	65332	5	101600	508000
35650	3	65335	1	50000	50000
35650	4	65339	1	7200	7200
35651	1	65337	1	18000	18000

Таблица: ТОВАР

ТОВ_КОД	ТОВ_НАЗВ	ТОВ_ДТП	ТОВ_ОСТ	ТОВ_ЦЕНА	ТОВ_СКД	ПСТ_КОД
65324	Ноутбук HP 17-ca1030...	04.07.2020	5	48700	10	6534
65325	Ноутбук Lenovo IdeaP...	15.07.2020	6	49900	10	6534
65326	Ноутбук G717-3875 DE...	30.06.2020	2	206999	10	6534
65327	Ноутбук Lenovo IdeaP...	05.06.2020	4	98799	10	6534
65328	Ноутбук HP 17-by3022...	06.06.2020	8	48199	10	6534
65329	Ноутбук ASUS M509DJ ...	10.05.2020	5	44400	10	6535
65330	Ноутбук HP 255 G7 < ...	04.04.2020	2	52500	10	6535
65331	Ноутбук Lenovo IdeaP...	09.06.2020	6	43600	5	6535
65332	Ноутбук HP EliteBook...	03.06.2020	9	101600	10	6534
65333	Ноутбук Acer Aspire ...	25.06.2020	7	46999	10	6534
65334	Планшет Huawei MateP...	08.06.2020	5	42500	10	6536
65335	Планшет Samsung Gala...	11.07.2020	2	50000	5	6536
65336	Планшет Huawei MateP...	28.07.2020	4	22000	0	6536
65337	Планшет Huawei Media...	09.07.2020	6	18000	0	6534
65338	Планшет Samsung Gala...	11.06.2020	5	11700	0	6534
65339	Планшет Lenovo TAB 7...	23.06.2020	8	7200	5	6535
65340	Планшет Prestigio Wi...	08.06.2020	2	4500	5	6536
65341	Планшет Digma Plane ...	25.07.2020	4	5100	3	6535
65342	Планшет IRBIS < TZ89...	19.05.2020	6	6400	3	6534
65343	Планшет 1LV15EA#ACB ...	12.07.2020	4	80000	10	6534

Таблица: КЛИЕНТ

КЛН_КОД	КЛН_ФАМ	КЛН_ИМЯ	КЛН_ОТЧ	КЛН_АДР	КЛН_ТЕЛ	КЛН_ДОЛГ
4587	Стародубов	Иван	Пахомович	Нижневартовск. ул Мира, дом 456, кв. 45	3466-079672	0
4588	Безукладников	Леонид	Адрианович	Нижневартовск. ул Ленина, дом 386, кв. 14	3466-790681	0
4589	Осенних	Николай	Серафимович	Нижневартовск. ул Интернациональная, дом 332, кв. 25	3466-127339	92500
4590	Генкин	Гавриил	Матвеевич	Нижневартовск. ул Северная, дом 326, кв. 12	3466-729121	0
4591	Серых	Лукьян	Кондратович	Нижневартовск. ул Южная, дом 5, кв. 69	3466-163133	18000
4592	Братцев	Ефрем	Владимирович	Нижневартовск. ул Лесная, дом 6, кв. 5	3466-143952	0
4593	Чижиков	Всеволод	Федотович	Нижневартовск. ул Весенная, дом 12, кв. 63	3466-351466	242000
4594	Попков	Давид	Семенович	Нижневартовск. ул Мира, дом 636, кв. 50	3466-279008	158500
4595	Кондратов	Ипполит	Казимирович	Нижневартовск. ул Западная, дом 2, кв. 25	3466-205378	652700
4596	Левицкий	Ерофей	Савельевич	Нижневартовск. ул Индустриальная, дом 45, кв. 652	3466-200245	0

Рис. 26. Транзакция в базе данных ТОРГОВЛЯ

Транзакция может состоять из одного оператора SQL или набора связанных операторов SQL. В предыдущем примере продаж, иллюстрирующем сложную транзакцию в базе данных ТОРГОВЛЯ, 3 августа 2020 года зарегистрирована продажа в кредит одной единицы продукта 65537 покупателю 4591 за 18000 рублей. Требуемая транзакция влияет на таблицы ЧЕК, СТРОКА, ТОВАР и КЛИЕНТ. Операторы SQL, которые представляют эту транзакцию, следующие:

INSERT INTO ЧЕК VALUES (35651, 4591, ‘2020-08-03’, 18000, 0, 18000, ‘кредит’);

INSERT INTO СТРОКА VALUES (35651, 1, 65337, 1, 18000, 18000);

UPDATE ТОВАР SET ТОВ_ОСТ = ТОВ_ОСТ – 1 WHERE ТОВ_КОД = 65337;

UPDATE КЛИЕНТ SET КЛН_ДОЛГ = КЛН_ДОЛГ + 18000 WHERE КЛН_КОД = 4591;

COMMIT.

Результаты успешно завершенной транзакции показаны на рисунке 26. Все записи, участвующие в транзакции, выделены красным.

Чтобы лучше понять результаты транзакции, необходимо обратить внимание на следующее:

- Новая строка 35651 была добавлена в таблицу ЧЕК.
- Была добавлена строка для чека 35651, чтобы отразить покупку одной единицы товара 65337 по цене 18000 рублей.
- Остаток товара 65337 в таблице ТОВАР был уменьшен на один, с 7 до 6.
- Долг клиента 4591 был обновлен путем добавления 18000 рублей к существующему долгу (первоначальное значение составляло 0,00 рублей).
- Оператор COMMIT использовался для завершения успешной транзакции.

Если после завершения первых трех операторов SQL, во время выполнения четвертого оператора компьютерная система теряет электроэнергию, транзакция не может быть завершена. Были добавлены строки ЧЕК и СТРОКА, таблица ТОВАР была обновлена, но у клиента 4591 не был обновлен долг. Соответственно, БД находится в несовместимом состоянии и не может использоваться для последующих транзакций. Предполагая, что СУБД поддерживает управление транзакциями, СУБД выполнит откат базы данных до предыдущего согласованного состояния.

Все известные СУБД, включая MS Access, Oracle, SQL Server и DB2, поддерживают управление транзакциями.

Хотя СУБД предназначена для восстановления базы данных до предыдущего согласованного состояния, когда прерывание предотвращает завершение транзакции, сама транзакция определяется пользователем или программистом и должна быть семантически правильной. СУБД не может гарантировать, что семантический смысл транзакции действительно представляет реальное событие. Например, предположим, что после продажи 10 единиц продукта 65337 команда UPDATE для обновления была написана следующим образом:

```
UPDATE ТОВАР
SET ТОВ_ОСТ = ТОВ_ОСТ + 10
WHERE ТОВ_КОД = 65337.
```

Продажа должна была уменьшить остаток для продукта 65337 на 10. Вместо этого UPDATE добавил 10 к значению ТОВ_ОСТ.

Хотя синтаксис команды UPDATE правильный, ее использование дает неверные результаты, то есть база данных не соответствует реальному событию. Тем не менее, СУБД выполнит транзакцию в любом случае. СУБД не может оценить, правильно ли транзакция представляет реальное событие, это ответственность пользователя. Пользователи и программисты способны вносить много ошибок таким способом.

Неправильные транзакции могут иметь разрушительные последствия для целостности базы данных. Некоторые СУБД, особенно реляционная разновидность, предоставляют средства, с помощью которых пользователь может определять принудительные ограничения

на основе бизнес-правил. Другие правила целостности, такие как правила, регулирующие ссылочную целостность и целостность объектов, автоматически применяются СУБД при правильном определении структур таблиц, что позволяет СУБД проверять некоторые транзакции. Например, если транзакция вставляет новый код клиента в таблицу клиентов, и этот номер уже существует, СУБД завершит транзакцию кодом ошибки, указывающим на нарушение правила целостности первичного ключа.

Свойства транзакции. Каждая отдельная транзакция должна отображать атомарность, согласованность, изоляцию и долговечность. Эти четыре свойства иногда называют тестом ACID. Давайте кратко рассмотрим каждое из свойств.

• *Атомарность* требует, чтобы все операции (запросы SQL) транзакции были завершены; если нет – транзакция отменяется. Если транзакция T_1 имеет четыре запроса SQL, все четыре запроса должны быть успешно завершены; в противном случае вся транзакция прерывается. Другими словами, транзакция рассматривается как единая неделимая логическая единица работы.

• *Согласованность* указывает на постоянство согласованного состояния базы данных. Транзакция переводит базу данных из одного согласованного состояния в другое. Когда транзакция завершена, база данных должна быть в согласованном состоянии. Если какая-либо из частей транзакции нарушает ограничение целостности, вся транзакция прерывается.

• *Изолированность* означает, что данные, используемые во время выполнения транзакции, не могут использоваться второй транзакцией, пока первая не будет завершена. Другими словами, если транзакция T_1 выполняется и использует элемент данных X, этот элемент данных не может быть доступен любой другой транзакции (T_2… T_n) до тех пор, пока не закончится T_1. Это свойство особенно полезно в многопользовательских средах баз данных, поскольку пользователи могут одновременно обращаться к базе данных и обновлять ее.

• *Долговечность* гарантирует, что после внесения и подтверждения изменений транзакции их нельзя отменить или потерять, даже в случае сбоя системы.

В дополнение к отдельным свойствам транзакции, указанным выше, есть еще одно важное свойство, которое применяется при одновременном выполнении нескольких транзакций. Например, предположим, что СУБД имеет три транзакции (T_1, T_2 и T_3), выполняющиеся одновременно. Для правильного выполнения транзакций СУБД должна планировать параллельное выполнение операций транзакции. В этом случае каждая отдельная транзакция должна соответствовать свойствам ACID, и в то же время график таких операций с несколькими транзакциями должен демонстрировать свойство сериализуемости. *Сериализуемость* гарантирует, что график одновременного выполнения транзакций даст согласованные результаты. Это свойство важно в многопользовательских и распределенных базах данных, в которых несколько транзакций могут выполняться одновременно. Естественно, если выполняется только одна транзакция, сериализуемость не является проблемой.

Система однопользовательской базы данных автоматически обеспечивает сериализуемость и изоляцию базы данных, поскольку одновременно выполняется только одна

транзакция. Атомарность, согласованность и долговечность транзакций должны гарантироваться однопользовательскими СУБД. (Даже однопользовательская СУБД должна управлять восстановлением после ошибок, вызванных прерываниями, сбоями операционной системы, перебоями в питании и ненормальными завершениями или сбоями приложений).

Многопользовательские базы данных обычно подвергаются нескольким параллельным транзакциям. Следовательно, многопользовательская СУБД должна реализовывать элементы управления для обеспечения сериализуемости и изоляции транзакций – в дополнение к атомарности и долговечности – для обеспечения своей целостности и целостности базы данных. Например, если несколько параллельных транзакций выполняются для одного и того же набора данных, а вторая транзакция обновляет базу данных до завершения первой транзакции, свойство изоляции нарушается, и база данных перестает быть согласованной. СУБД должна управлять транзакциями, используя методы управления параллелизмом, чтобы избежать нежелательных ситуаций.

Управление транзакциями с помощью SQL. ANSI определил стандарты, которые управляют транзакциями базы данных SQL. Поддержка транзакций обеспечивается двумя операторами SQL: COMMIT и ROLLBACK. Стандарты ANSI требуют, чтобы, когда последовательность транзакций инициировалась пользователем или прикладной программой, последовательность продолжалась через все последующие операторы SQL до тех пор, пока не произойдет одно из следующих четырех событий:

- Оператор COMMIT достигается, и в этом случае все изменения записываются в базу данных. Оператор COMMIT автоматически завершает транзакцию SQL.
- Оператор ROLLBACK достигнут, и в этом случае все изменения отменяются, и база данных возвращается к своему предыдущему согласованному состоянию.
- Конец программы успешно достигнут, и в этом случае все изменения записываются в базу данных. Это действие эквивалентно COMMIT.
- Программа аварийно завершена, и в этом случае изменения базы данных отменяются, и база данных возвращается к своему предыдущему согласованному состоянию. Это действие эквивалентно ROLLBACK.

Использование COMMIT иллюстрируется в следующем упрощенном примере продаж, который обновляет количество товара в наличии (ТОВ_ОСТ) и долг клиента, когда клиент покупает две единицы продукта 65335 по цене 50000 рублей за единицу (на общую сумму 100000 рублей):

```
UPDATE ТОВАР
SET ТОВ_ОСТ = ТОВ_ОСТ – 2
WHERE ТОВ_КОД = 65535;
UPDATE КЛИЕНТ
SET КЛН_ДОЛГ = КЛН_ДОЛГ + 100000
WHERE КЛН_КОД = 4589;
COMMIT.
```

Оператор COMMIT, использованный в предыдущем примере, не является обязательным, если оператор UPDATE является последним действием приложения, и приложение нормально завершается. Однако хорошая практика программирования требует, чтобы включали оператор COMMIT в конце объявления транзакции.

Транзакция начинается неявно, когда встречается первый оператор SQL. Не все реализации SQL соответствуют стандарту ANSI; некоторые (например, SQL Server) используют операторы управления транзакциями, такие, как следующие, чтобы указать начало новой транзакции:

BEGIN TRANSACTION.

Другие реализации SQL позволяют назначать характеристики для транзакций в качестве параметров оператора BEGIN. Например, СУБД Oracle использует инструкцию SET TRANSACTION для объявления начала новой транзакции и ее свойств.

Журнал транзакций. СУБД использует *журнал транзакций* для отслеживания всех транзакций, которые обновляют базу данных. СУБД использует информацию, хранящуюся в этом журнале, для требования восстановления, вызванного оператором ROLLBACK, ненормальным завершением программы или системным отказом, таким как ошибка сети или сбой диска. Некоторые СУРБД используют журнал транзакций для восстановления базы данных до текущего согласованного состояния. Например, после сбоя сервера Oracle автоматически откатывает незафиксированные транзакции и откатывает транзакции, которые были зафиксированы, но еще не записаны в физическую базу данных. Такое поведение требуется для корректности транзакции и типично для любой транзакционной СУБД.

Хотя СУБД выполняет транзакции, которые изменяют базу данных, она также автоматически обновляет журнал транзакций. В журнале транзакций хранится следующее:

- Запись о начале транзакции.
- Для каждого компонента транзакции (оператор SQL):
  - Тип выполняемой операции (INSERT, UPDATE, DELETE).
  - Имена объектов, затронутых транзакцией (название таблицы).
  - Значения «до» и «после» для обновляемых полей.
  - Указатели на предыдущую и следующую записи журнала транзакций для той же транзакции.
- Окончание (COMMIT) транзакции.

Хотя использование журнала транзакций увеличивает нагрузку на обработку СУБД, возможность восстановления поврежденной базы данных стоит свою цену.

Если происходит системный сбой, СУБД проверит журнал транзакций на предмет всех незавершенных или неподтвержденных транзакций и восстановит (ROLLBACK) базу данных до ее предыдущего состояния на основе этой информации. По завершении процесса восстановления СУБД запишет в журнал все подтвержденные транзакции, которые не были физически записаны в базу данных до возникновения сбоя.

Если ROLLBACK выдается до завершения транзакции, СУБД восстановит базу данных только для этой конкретной транзакции, а не для всех из них, чтобы сохранить долговечность предыдущих транзакций. Другими словами, совершенные транзакции не откатываются.

Журнал транзакций является важной частью базы данных, и он обычно реализуется как один или несколько файлов, которые управляются отдельно от фактических файлов базы данных. Журнал транзакций подвержен распространенным опасностям, таким как переполнение диска и сбой диска. Поскольку журнал транзакций содержит некоторые наиболее важные данные в СУБД, некоторые реализации поддерживают журналы на нескольких разных дисках, чтобы уменьшить последствия сбоя системы.

7.2. Параллельные транзакции

Координация одновременного выполнения транзакций в многопользовательской системе баз данных называется *контролем параллелизма*. Целью контроля параллелизма является обеспечение сериализуемости транзакций в многопользовательской среде баз данных. Для достижения этой цели большинство методов управления параллелизмом ориентированы на сохранение свойства изоляции при одновременном выполнении транзакций. Контроль параллелизма важен, потому что одновременное выполнение транзакций над общей базой данных может создать несколько проблем целостности данных и согласованности. Три основные проблемы – это потерянные обновления, незафиксированные данные и непоследовательный поиск.

Потерянные обновления. Проблема *потерянного обновления* возникает, когда две параллельные транзакции T_1 и T_2, обновляют один и тот же элемент данных, и одно из обновлений теряется (перезаписывается другой транзакцией). Чтобы увидеть иллюстрацию потерянных обновлений, можно посмотреть таблицу TOBAP. Одним из атрибутов таблицы является количество товара в наличии (TOB_OCT). Предположим, что есть товар с текущим значением TOB_OCT 25. Также предположим, что происходят две параллельные транзакции, T_1 и T_2, и обновляют значение TOB_OCT для некоторого элемента в таблице TOBAP. Транзакции показаны в таблице 17.

Таблица 17

Две параллельные транзакции над TOB_OCT

Транзакция	Вычисление
T_1: Покупка 50 единиц	TOB_OCT = TOB_OCT + 50
T_2: Продажа 20 единиц	TOB_OCT = TOB_OCT - 20

Таблица 18 показывает последовательное выполнение транзакций при нормальных обстоятельствах, выдает правильный ответ TOB_OCT = 55.

Таблица 18

Последовательное выполнение двух транзакций

Шаг	Транзакция	Команда	Результат
1	T_1	Чтение TOB_OCT	25
2	T_1	TOB_OCT=25+50	
3	T_1	Запись TOB_OCT	75
4	T_2	Чтение TOB_OCT	75
5	T_2	TOB_OCT=75-20	
6	T_2	Запись TOB_OCT	55

Предположим, что транзакция может прочитать значение TOB_OCT из таблицы до того, как была совершена предыдущая транзакция, используя тот же товар. Последовательность, записанная в таблице 19, показывает, как может возникнуть проблема с потерянным обновлением. Нужно обратить внимание, что первая транзакция (T_1) еще не была зафиксирована при выполнении второй транзакции (T_2). Следовательно, T_2 все еще работает со значением 25, и его вычитание дает 5 в памяти. Тем временем, T_1 записывает значение 75 на диск, который быстро перезаписывается T_2. А добавление 50 единиц «теряется» во время процесса.

Таблица 19

Потерянные обновления

Шаг	Транзакция	Команда	Результат
1	T_1	Чтение TOB_OCT	25
2	T_2	Чтение TOB_OCT	25
3	T_1	TOB_OCT=25+50	
4	T_2	TOB_OCT=25-20	
5	T_1	Запись TOB_OCT	75
6	T_2	Запись TOB_OCT	5

Незафиксированные данные. Явление незафиксированных данных происходит тогда, когда две транзакции T_1 и T_2 выполняются одновременно, и первая транзакция (T_1) откатывается после того, как вторая транзакция (T_2) уже получила доступ к незафиксированным данным, что нарушает свойство изоляции транзакций. Примером будут те же транзакции, которые описаны во время обсуждения потерянных обновлений. T_1 имеет две атомарные части, первая – обновление остатка товара; вторая – обновление суммы чека. T_1 вынужден откатиться из-за ошибки при обновлении суммы; он полностью откатывается, отменяя также обновление остатков товара. На этот раз транзакция T_1 откатывается, чтобы исключить добавление 50 единиц – таблица 20. Поскольку T_2 вычитает 20 из исходных 25 единиц, правильный ответ должен быть 5.

Таблица 20

Транзакции, создающие ситуацию "незафиксированные данные"

Транзакция	Вычисление
T_1: Покупка 50 единиц	TOB_OCT = TOB_OCT + 50 (ОТМЕНЕН)
T_2: Продажа 20 единиц	TOB_OCT = TOB_OCT - 20

Таблица 21 показывает, как последовательное выполнение этих транзакций дает правильный ответ при нормальных обстоятельствах.

Таблица 21

Правильное выполнение двух транзакций

Шаг	Транзакция	Команда	Результат
1	T_1	Чтение TOB_OCT	25
2	T_1	TOB_OCT=25+50	
3	T_1	Запись TOB_OCT	75
4	T_1	ROLLBACK	25
5	T_2	Чтение TOB_OCT	25
6	T_2	TOB_OCT=25-20	
7	T_2	Запись TOB_OCT	5

В таблице 22 показано, как может возникнуть проблема с незафиксированными данными, когда ROLLBACK завершен после того, как T_2 начал свое выполнение.

Таблица 22

Ситуация "незафиксированные данные"

Шаг	Транзакция	Команда	Результат
1	T_1	Чтение TOB_OCT	25
2	T_1	TOB_OCT=25+50	
3	T_1	Запись TOB_OCT	75
4	T_2	Чтение TOB_OCT (Незафиксированные данные)	75
5	T_2	TOB_OCT=25-20	
6	T_1	ROLLBACK	25
7	T_2	Запись TOB_OCT	55

Несовместимые извлечения. *Несогласованные извлечения* происходят, когда транзакция обращается к данным до и после того, как одна или несколько других транзакций заканчивают работу с такими данными. Например, несогласованный поиск может произойти, если транзакция T_1 вычисляет некоторую агрегатную функцию, в то время как другая транзакция (T_2) обновляет те же данные. Проблема заключается в том, что транзакция может прочитать некоторые данные до того, как они будут изменены, и другие данные после того, как они будут изменены, что приведет к противоречивым результатам.

Следующие условия проиллюстрируют проблему:

1. T_1 вычисляет общее количество товаров, хранящихся в таблице TOBAP.
2. В то же время T_2 обновляет имеющееся количество (TOB_OCT) для двух товаров.

Две транзакции показаны в таблице 23.

Таблица 23

Извлечение во время обновления

Транзакция T_1	**Транзакция** T_2
SELECT SUM(TOB_OCT) FROM TOBAP	UPDATE TOBAP SET TOB_OCT = TOB_OCT + 5 WHERE TOB_КОД= 65330
	UPDATE TOBAP SET TOB_OCT = TOB_OCT - 5 WHERE TOB_КОД= 65338
	COMMIT

В то время как T_1 вычисляет суммарный остаток товаров (TOB_OCT) для всех позиций, T_2 представляет исправление ошибки печати: пользователь добавил 5 штук к товару 65338, но намеревался добавить 5 штук к товару 65330. Чтобы устранить проблему, пользователь добавляет 5 к товару 65330 и вычитает 5 из товара 65338. Начальные и конечные значения TOB_OCT отражены в таблице 24.

Таблица 24

Результат транзакции

	До	**После**
TOB_КОД	TOB_OCT	TOB_OCT
65330	2	2 + 5 → 7
65331	6	6
65332	9	9
65333	7	7
65334	5	5
65335	2	2
65336	4	4
65337	6	6
65338	5	5 – 5 → 0
ИТОГО	46	46

Хотя окончательные результаты, показанные в таблице 24, верны после корректировки, таблица 25 показывает, что во время выполнения транзакции возможны несогласованные извлечения, что делает результат выполнения T_1 некорректным. Суммирование «После», показанное в Таблице 25, отражает то, что значение 0 для товара 65338 было прочитано после выполнения оператора UPDATE. Таким образом, итоговое значение «После» равно 41 + 0 = 41. Итоговое значение «До» отражает то, что значение 2 для товара 65330 было прочитано до того, как была выполнена следующая команда UPDATE, чтобы исправить значение на 7. Следовательно, итого «До» составляет 0 + 2 = 2.

Таблица 25

Несовместимое извлечение

Шаг	Транзакция	Команда	Значение	Итого
1	T_1	Чтение TOB_OCT для TOB_КОД=65330	2	2 (До)
2	T_1	Чтение TOB_OCT для TOB_КОД=65331	6	8
3	T_1	Чтение TOB_OCT для TOB_КОД=65332	9	17
4	T_2	Чтение TOB_OCT для TOB_КОД=65330	2	
5	T_2	TOB_OCT = 2 + 5		
6	T_2	Запись TOB_OCT для TOB_КОД=65330	7	
7	T_1	Чтение TOB_OCT для TOB_КОД=65333	7	24
8	T_1	Чтение TOB_OCT для TOB_КОД=65334	5	29
9	T_1	Чтение TOB_OCT для TOB_КОД=65335	2	31
10	T_2	Чтение TOB_OCT для TOB_КОД=65338	5	
11	T_2	TOB_OCT = 5 - 5		
12	T_2	Запись TOB_OCT для TOB_КОД=65338	0	
13	T_2	COMMIT		
14	T_1	Чтение TOB_OCT для TOB_КОД=65336	4	35
15	T_1	Чтение TOB_OCT для TOB_КОД=65337	6	41
16	T_1	Чтение TOB_OCT для TOB_КОД=65338	0	41 (После)

Вычисленный ответ 41, очевидно, неверен, поскольку из таблицы 24 известно, что правильный ответ = 46. Если СУБД не осуществляет контроль параллелизма, среда многопользовательской базы данных может создать хаос в информационной системе.

Планировщик. Серьезные проблемы могут возникнуть при выполнении двух или более параллельных транзакций. Транзакция базы данных включает в себя ряд операций ввода-вывода, которые переводят базу данных из одного согласованного состояния в другое. Согласованность базы данных может быть обеспечена только до и после выполнения транзакций. База данных всегда проходит через неизбежное временное состояние несоответствия во время выполнения транзакции, если такая транзакция обновляет несколько таблиц и строк. Временное несоответствие существует, поскольку компьютер выполняет операции последовательно, один за другим. Во время этого последовательного процесса свойство изоляции транзакций не позволяет им получать доступ к данным, еще невыпущенным другими транзакциями. Это соображение еще более важно сегодня с использованием многоядерных процессоров, которые могут выполнять несколько инструкций одновременно. Что произойдет, если две транзакции выполняются одновременно, и они получают доступ к одним и тем же данным?

В предыдущих примерах операции внутри транзакции выполнялись в произвольном порядке. Пока две транзакции, T_1 и T_2, обращаются к несвязанным данным, конфликт отсутствует, и порядок выполнения не имеет отношения к конечному результату. Однако, если транзакции работают с соответствующими данными или одними и теми же данными, возможен конфликт между компонентами транзакции, и выбор одного порядка выполнения

над другим может иметь некоторые нежелательные последствия. Как определяется правильный порядок, и кто определяет этот порядок? К счастью, СУБД справляется с этим сложным назначением с помощью встроенного планировщика.

Планировщик – это особый процесс СУБД, который устанавливает порядок, в котором операции выполняются в параллельных транзакциях. Планировщик чередует выполнение операций базы данных, чтобы обеспечить сериализуемость и изоляцию транзакций. Чтобы определить соответствующий порядок, планировщик основывает свои действия на алгоритмах управления параллелизмом таких, как методы блокировки или отметки времени. Однако важно понимать, что не все транзакции являются сериализуемыми. СУБД определяет, какие транзакции являются такими, и по очереди выполняет их. Основная задача планировщика заключается в создании *сериализуемого расписания* операций транзакции, в котором чередованное выполнение транзакций (T_1, T_2, T_3 и т.д.) дает те же результаты, как если бы транзакции выполнялись в последовательном порядке.

Планировщик также обеспечивает эффективное использование центрального процессора компьютера и памяти. Если бы не было возможности запланировать выполнение транзакций, все они были бы выполнены в порядке поступления. Проблема этого подхода заключается в том, что время обработки теряется, когда процессор ожидает завершения операции READ или WRITE, тем самым теряя несколько циклов. Планирование «первым пришел, первым обслужен» в многопользовательской среде СУБД приводит к недопустимому времени отклика. Поэтому для повышения эффективности всей системы необходим какой-то другой метод планирования.

Кроме того, планировщик облегчает изоляцию данных, чтобы две транзакции не обновляли один и тот же элемент данных одновременно. Операции с базой данных могут потребовать действий READ и/или WRITE, которые вызывают конфликты. Например, в таблице 26 показаны возможные сценарии конфликта, когда две транзакции T_1 и T_2, выполняются одновременно для одних и тех же данных. Обратите внимание, что в таблице 26 две операции находятся в конфликте, когда они получают доступ к одним и тем же данным, и по крайней мере одна из них является операцией WRITE.

Таблица 26

Конфликт чтение/запись

	Транзакции		
	T_1	**T_2**	**Результат**
Операция	Чтение	Чтение	Без конфликта
	Чтение	Запись	Конфликт
	Запись	Чтение	Конфликт
	Запись	Запись	Конфликт

Было предложено несколько методов для планирования выполнения конфликтующих операций в параллельных транзакциях. Эти методы классифицируются как блокировка, отметка времени и оптимистичные. Методы блокировки, обсуждаемые далее, используются чаще всего.

7.3. Управление с методами блокировки

Методы блокировки являются одним из наиболее распространенных методов, используемых в управлении параллелизмом, поскольку они облегчают изоляцию элементов данных, используемых при одновременном выполнении транзакций. *Блокировка* гарантирует исключительное использование элемента данных для текущей транзакции. Другими словами, транзакция T_2 не имеет доступа к элементу данных, который в данный момент используется транзакцией T_1. Транзакция получает блокировку до доступа к данным; блокировка снимается, когда транзакция завершается, так что другая транзакция может заблокировать элемент данных для его исключительного использования. Эта серия блокирующих действий предполагает, что параллельные транзакции могут пытаться манипулировать одним и тем же элементом данных в одно и то же время. Использование блокировок, основанных на предположении о вероятности конфликта между транзакциями, обычно называют *пессимистической блокировкой*.

Согласованность данных не может быть гарантирована во время транзакции; база данных может находиться во временном несовместимом состоянии при выполнении нескольких обновлений. Следовательно, блокировки необходимы для предотвращения чтения несогласованными данными другой транзакции.

Современные СУБД автоматически инициируют и применяют процедуры блокировки. Вся информация о блокировке обрабатывается *менеджером блокировок*, который отвечает за назначение и контроль блокировок, используемых транзакциями.

Степень детализации блокировки. *Степень детализации блокировки* показывает уровень использования блокировки. Блокировка может осуществляться на следующих уровнях: база данных, таблица, страница, строка или даже поле (атрибут).

При *блокировке уровня базы данных* вся база данных блокируется, что предотвращает использование любых таблиц в базе данных транзакцией T_2 во время выполнения транзакции T_1. Этот уровень блокировки хорош для пакетных процессов, но не подходит для многопользовательских СУБД.

При *блокировке на уровне таблицы* вся таблица блокируется, предотвращая доступ к любой строке транзакции T_2, пока транзакция T_1 использует таблицу. Если транзакция требует доступа к нескольким таблицам, каждая таблица может быть заблокирована. Однако две транзакции могут обращаться к одной и той же базе данных, если они обращаются к разным таблицам. Блокировки на уровне таблицы, хотя и менее строгие, чем блокировки на уровне базы данных, вызывают пробки, когда многие транзакции ожидают доступа к одной и той же таблице. Такое условие особенно утомительно, если блокировка вызывает задержку, когда разные транзакции требуют доступа к разным частям одной и той же таблицы, то есть когда транзакции не будут мешать друг другу. Следовательно, блокировки на уровне таблицы не подходят для многопользовательских СУБД.

При *блокировке на уровне страницы* СУБД блокирует всю страницу диска. *Страница диска*, или *страница*, является эквивалентом дискового блока, который может быть описан как непосредственно адресуемый раздел диска. Страница имеет фиксированный размер,

например, 4КБ, 8КБ или 16КБ. Например, если необходимо записать только 73 байта на страницу 4КБ, всю страницу необходимо прочитать с диска, обновить в памяти и записать обратно на диск. Таблица может занимать несколько страниц, а страница может содержать несколько строк одной или нескольких таблиц. В настоящее время блокировки на уровне страниц являются наиболее часто используемым методом блокировки для многопользовательских СУБД.

Блокировка на уровне строк гораздо менее ограничительная. СУБД позволяет одновременным транзакциям получать доступ к разным строкам одной и той же таблицы, даже если строки расположены на одной странице. Хотя подход блокировки на уровне строк улучшает доступность данных, его управление требует больших накладных расходов, поскольку существует блокировка для каждой строки в таблице базы данных, участвующей в конфликтующей транзакции. Современные СУБД автоматически повышают блокировку с уровня строки до уровня страницы, когда сеанс приложения запрашивает несколько блокировок на одной странице.

Блокировка на уровне поля позволяет одновременным транзакциям получать доступ к одной и той же строке, если они требуют использования разных полей (атрибутов) в этой строке. Хотя блокировка на уровне поля однозначно обеспечивает наиболее гибкий многопользовательский доступ к данным, она редко реализуется в СУБД, поскольку требует чрезвычайно высокого уровня издержек компьютера и потому что блокировка на уровне строк гораздо более полезна на практике.

Типы замков. Независимо от уровня блокировки, СУБД может использовать разные типы или режимы блокировки: двоичная или разделяемая/исключительная.

Двоичная блокировка имеет только два состояния: заблокировано (1) или разблокировано (0). Если объект, такой как база данных, таблица, страница или строка заблокирован транзакцией, никакая другая транзакция не может использовать этот объект. Если объект разблокирован, любая транзакция может заблокировать объект для его использования. Каждая операция с базой данных требует, чтобы уязвимый объект был заблокирован. Как правило, транзакция должна разблокировать объект после его завершения. Поэтому каждая транзакция требует операции блокировки и разблокировки для каждого элемента данных, к которому осуществляется доступ. Такие операции автоматически управляются и планируются СУБД; пользователь не блокирует и не разблокирует элементы данных.

Исключительная блокировка существует, когда доступ зарезервирован специально для транзакции, которая заблокировала объект. Исключительная блокировка должна использоваться, когда существует вероятность конфликта. *Общая блокировка* существует, когда параллельным транзакциям предоставляется доступ на чтение на основе общей блокировки. Общая блокировка не вызывает конфликта, если все параллельные транзакции доступны только для чтения.

Общая блокировка выдается, когда транзакция хочет прочитать данные из базы данных, и для этого элемента данных не установлена исключительная блокировка.

Исключительная блокировка выдается, когда транзакция хочет обновить (записать) элемент данных, и в настоящий момент никакие блокировки для этого элемента данных не удерживаются какой-либо другой транзакцией. Используя концепцию общей/исключительной блокировки, блокировка может иметь три состояния: разблокирована, общая (чтение) и исключительная (запись).

Две транзакции конфликтуют только тогда, когда хотя бы одна транзакция является записью. Поскольку две транзакции чтения могут быть безопасно выполнены одновременно, общие блокировки позволяют нескольким транзакциям чтения одновременно считывать один и тот же элемент данных. Например, если транзакция T_1 имеет общую блокировку элемента X данных, и транзакция T_2 хочет прочитать элемент X данных, T_2 также может получить общую блокировку элемента X данных.

Если транзакция T_2 обновляет элемент данных X, T_2 требует исключительную блокировку для элемента данных X. Исключительная блокировка предоставляется, если никакие другие блокировки не удерживаются для элемента данных (это условие известно, как *правило взаимоисключающего доступа*: только одна транзакция за раз может владеть исключительной блокировкой объекта). Следовательно, если совместно используемая (или исключительная) блокировка для элемента данных X уже удержана транзакцией T_1, исключительная блокировка не может быть предоставлена транзакции T_2, и T_2 должна ждать, пока T_1 не завершится. Другими словами, общая блокировка всегда будет запрещать исключительную запись (эксклюзивную блокировку); следовательно, происходит снижение параллелизма транзакций.

Хотя использование общих блокировок делает доступ к данным более эффективным, схема общей/эксклюзивной блокировки увеличивает накладные расходы менеджера блокировок по нескольким причинам:

- Тип блокировки должен быть известен до того, как она может быть предоставлена.
- Существуют три операции блокировки: READ_LOCK для проверки типа блокировки, WRITE_LOCK – для выдачи блокировки, и UNLOCK – для снятия блокировки.
- Схема была усовершенствована, чтобы разрешить обновление блокировки с общей на эксклюзивную, и понижение блокировки с эксклюзивной на общую.

Хотя блокировки предотвращают серьезные несоответствия данных, они могут привести к двум основным проблемам:

- Итоговое расписание транзакций может не быть сериализуемым.
- Расписание может создать тупики. *Взаимная блокировка* возникает, когда две транзакции бесконечно ждут друг друга, чтобы разблокировать данные. Блокировка базы данных, которая аналогична блокировке трафика в большом городе, возникает, когда две или более транзакций ждут друг друга, чтобы разблокировать данные.

К счастью, обеими проблемами можно управлять: сериализуемость достигается с помощью протокола блокировки, известного как двухфазная блокировка, а тупиковыми

ситуациями можно управлять с помощью методов обнаружения и предотвращения взаимных блокировок. Эти методы рассматриваются в следующих двух разделах.

Двухфазная блокировка. *Двухфазная блокировка (2ФБ)* определяет, как транзакции получают и снимают блокировки. Двухфазная блокировка гарантирует сериализуемость, но не предотвращает взаимоблокировки. Две фазы:

1. Растущая фаза, на которой транзакция получает все необходимые блокировки без разблокировки каких-либо данных. Как только все блокировки были получены, транзакция находится в своей заблокированной точке.
2. Фаза сжатия, при которой транзакция снимает все блокировки и не может получить новую блокировку.

Протокол двухфазной блокировки регулируется следующими правилами:

- Две транзакции не могут иметь конфликтующие блокировки.
- Операция блокировки не может предшествовать операции блокировки в той же транзакции.
- На данные не влияют, пока не будут получены все блокировки, то есть до тех пор, пока транзакция не окажется в своей заблокированной точке.

Тупики. Взаимная блокировка возникает, когда две транзакции бесконечно ждут друг друга, чтобы разблокировать данные. Например, тупик возникает, когда две транзакции T_1 и T_2 существуют в следующем режиме:

T_1 = доступ к элементам данных X и Y,

T_2 = доступ к элементам данных Y и X.

Если T_1 не имеет разблокированного элемента данных Y, T_2 не может начаться; если T_2 не разблокировала элемент данных X, T_1 не может продолжиться. Следовательно, и T_1, и T_2 ждут, пока одна из них разблокирует требуемый элемент данных. Такой тупик также известен как *смертельные объятия*.

В СУБД одновременно может быть выполнено гораздо больше транзакций, что увеличивает вероятность возникновения тупиковых ситуаций. Взаимоблокировки возможны только тогда, когда одна из транзакций хочет получить монопольную блокировку элемента данных; между общими блокировками не может быть тупиковых условий.

Три основных метода управления тупиками:

- *Предотвращение тупиковой ситуации.* Транзакция, запрашивающая новую блокировку, прерывается, когда существует вероятность возникновения тупика. Если транзакция отменяется, все изменения, сделанные этой транзакцией, откатываются и все блокировки, полученные транзакцией, снимаются. Затем транзакция переносится на исполнение. Предотвращение взаимоблокировки работает, потому что оно избегает условий, которые приводят к взаимоблокировке.
- *Обнаружение тупика.* СУБД периодически проверяет базу данных на наличие взаимоблокировок. Если обнаружена взаимоблокировка, транзакция «жертвы» прерывается (откатывается и перезапускается), а другая транзакция продолжается.

- *Избегание от тупиковой ситуации.* Транзакция должна получить все необходимые блокировки, прежде чем она может быть выполнена. Этот метод позволяет избежать отката конфликтующих транзакций, требуя, чтобы блокировки были получены последовательно. Однако назначение последовательной блокировки, необходимое для предотвращения тупика, увеличивает время отклика на действие.

Выбор используемого метода управления взаимоблокировкой зависит от среды базы данных. Если вероятность взаимоблокировки низкая, рекомендуется обнаружение взаимоблокировки. Если вероятность взаимоблокировки высока, рекомендуется предотвращение взаимоблокировки. Если время отклика не является высоким в списке приоритетов системы, может быть использовано предотвращение тупиковых ситуаций. Все существующие СУБД поддерживают обнаружение взаимоблокировок в транзакционных базах данных, в то время как некоторые СУБД используют сочетание методов предотвращения взаимоблокировки и предотвращения для других типов данных, таких как хранилища данных или данные XML.

7.4. Управление с метками времени и с оптимистичными методами

Подход *с метками времени* для планирования параллельных транзакций назначает глобальную уникальную метку времени каждой транзакции. Значение метки времени создает явный порядок, в котором транзакции передаются в СУБД. Метки времени должны иметь два свойства: уникальность и монотонность. *Уникальность* гарантирует, что равные значения меток времени не могут существовать, а *монотонность* гарантирует, что значения меток времени всегда увеличиваются.

Все операции с базой данных (чтение и запись) в рамках одной транзакции должны иметь одинаковую отметку времени. СУБД выполняет конфликтующие операции в порядке отметки времени, обеспечивая тем самым сериализуемость транзакций. Если две транзакции конфликтуют, одна останавливается, откатывается, перепланируется и назначается новое значение метки времени.

Недостаток подхода с отметками времени состоит в том, что для каждого значения, хранящегося в базе данных, требуются два дополнительных поля отметки времени: одно для последнего времени чтения поля и одно для последнего обновления. Таким образом, временные метки увеличивают потребность в памяти и накладные расходы на обработку базы данных. Временная метка требует много системных ресурсов, потому что многие транзакции могут быть остановлены, перенесены и перепланированы.

Оптимистический подход основан на предположении, что большинство операций базы данных не конфликтуют. Оптимистический подход не требует ни методов блокировки, ни временных меток. Вместо этого транзакция выполняется без ограничений, пока она не будет зафиксирована. Используя оптимистический подход, каждая транзакция проходит две или три фазы, называемые чтением, проверкой и записью.

- На этапе чтения транзакция считывает базу данных, выполняет необходимые вычисления и обновляет частную копию значений базы данных. Все операции

обновления транзакции записываются во временный файл обновления, к которому не обращаются остальные транзакции.

- На этапе проверки транзакция проверяется, чтобы убедиться, что внесенные изменения не повлияют на целостность и согласованность базы данных. Если проверочный тест положительный, транзакция переходит в фазу записи. Если проверочный тест отрицательный, транзакция перезапускается и изменения отменяются.
- На этапе записи изменения постоянно применяются к базе данных.

Оптимистический подход является приемлемым для большинства систем баз данных для чтения или запроса, которые требуют мало транзакций обновления. В интенсивно используемой среде СУБД управление взаимоблокировками – их предотвращение и обнаружение – составляет важную функцию СУБД.

7.5. Уровни изоляции транзакций

Стандарт ANSI SQL (1992) определяет управление транзакциями на основе уровней изоляции транзакций. Уровни изоляции транзакций относятся к степени, в которой данные транзакций «защищены или изолированы» от других одновременных транзакций. Уровни изоляции описаны на основе того, какие данные другие транзакции могут видеть (читать) во время выполнения. Точнее, уровни изоляции транзакции описываются типом «чтения», который транзакция позволяет или нет. Типы операций чтения:

- *Грязное чтение*: транзакция может читать данные, которые еще не зафиксированы.
- *Неповторяемое чтение*: транзакция читает данную строку в момент времени t_1, а затем читает одну и ту же строку в момент времени t_2, получая разные результаты. Исходная строка может быть обновлена или удалена.
- *Фантомное чтение*: транзакция выполняет запрос в момент времени t_1, а затем выполняет тот же запрос в момент времени t_2, получая дополнительные строки, которые удовлетворяют запросу.

Основываясь на вышеупомянутых операциях, ANSI определила четыре уровня изоляции транзакций: чтение незафиксировано, чтение зафиксировано, повторное чтение и сериализуемое чтение. Существует еще дополнительный уровень изоляции, обеспечиваемый базами данных Oracle и MS SQL Server.

Read Uncommitted будет считывать незафиксированные данные из других транзакций. На этом уровне изоляции база данных не блокирует данные, что повышает производительность транзакций, но за счет согласованности данных. *Read Committed* заставляет транзакции читать только зафиксированные данные. Это режим работы по умолчанию для большинства баз данных. На этом уровне база данных будет использовать эксклюзивные блокировки данных, заставляя другие транзакции ждать, пока исходная транзакция не будет зафиксирована. Уровень изоляции *Repeatable Read* гарантирует, что запросы возвращают согласованные результаты. Этот тип уровня изоляции использует общие блокировки, чтобы другие транзакции не обновляли строку после того, как исходный запрос

ее читает. Однако новые строки читаются (фантомное чтение), так как эти строки не существовали, когда выполнялся первый запрос. *Serializable* уровень изоляции является наиболее ограничительным уровнем, определенным стандартом ANSI SQL. Тем не менее, важно отметить, что даже при Serializable уровне изоляции всегда возможны взаимоблокировки. Большинство баз данных используют подход обнаружения взаимоблокировок к управлению транзакциями, и, следовательно, они будут обнаруживать «взаимоблокировки» на этапе проверки транзакции и перепланировать транзакцию.

Причина разных уровней изоляции заключается в повышении параллелизма транзакций. Уровни изоляции переходят от наименее ограничивающего (Read Uncommitted) к более ограничивающему (Serializable). Чем выше уровень изоляции, тем больше блокировок (общих и исключительных) требуется для улучшения согласованности данных за счет производительности параллелизма транзакций. Уровень изоляции транзакции определяется в операторе транзакции, например, с использованием общего синтаксиса ANSI SQL:

```
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED
… SQL КОМАНДЫ….
COMMIT TRANSACTION.
```

Oracle и MS SQL Server используют инструкцию SET TRANSACTION ISOLATION LEVEL для определения уровня изоляции. SQL Server поддерживает все четыре уровня изоляции ANSI. По умолчанию Oracle обеспечивает согласованное чтение на уровне инструкций для обеспечения транзакций Read Committed и Repeatable Read. MySQL использует START TRANSACTION WITH CONSISTENT SNAPSHOT для обеспечения транзакций с согласованным чтением; то есть транзакция может видеть зафиксированные данные только во время ее запуска.

7.6. Управление восстановлением базы данных

Восстановление базы данных из заданного состояния (обычно несовместимого) в ранее согласованное состояние. Методы восстановления основаны на атомарном свойстве транзакции: все части транзакции должны рассматриваться как единая логическая единица работы, в которой все операции применяются и выполняются для создания согласованной базы данных. Если операция по какой-либо причине не может быть завершена, транзакция должна быть прервана, а любые изменения в базе данных должны быть отменены. Короче говоря, восстановление транзакции отменяет все изменения, которые транзакция внесла в базу данных до того, как транзакция была прервана.

Методы восстановления также применяются к базе данных и системе после возникновения критических ошибок определенного типа. Критические события могут привести к прекращению работы базы данных и нарушению целостности данных. Примеры критических событий:

- *Аппаратные/программные сбои*. Отказ такого типа может быть отказом жесткого диска, неисправным конденсатором на материнской плате или неисправным банком памяти. Другие причины ошибок в этой категории включают ошибки прикладной

программы или операционной системы, которые приводят к перезаписи, удалению или потере данных. Некоторые администраторы баз данных утверждают, что это один из наиболее распространенных источников проблем с базами данных.

- *Техногенные инциденты*. Событие такого типа можно отнести к категории непреднамеренных или преднамеренных.
  - Непреднамеренный сбой вызван неосторожным конечным пользователем. К таким ошибкам относятся удаление неправильных строк из таблицы, нажатие неправильной клавиши на клавиатуре или аварийное завершение работы основного сервера базы данных.
  - Преднамеренные события имеют более серьезный характер и обычно указывают на то, что данные компании подвергаются серьезному риску. В эту категорию входят угрозы безопасности, вызванные хакерами, пытающимися получить несанкционированный доступ к ресурсам данных, и вирусные атаки, вызванные недовольными сотрудниками, пытающимися скомпрометировать работу базы данных и нанести ущерб компании.
- *Стихийные бедствия*. К этой категории относятся пожары, землетрясения, наводнения и перебои в подаче электроэнергии.

Восстановление транзакции. Восстановление транзакций базы данных использует данные в журнале транзакций для восстановления базы данных из несовместимого состояния в согласованное состояние.

Прежде чем продолжить, необходимо изучить четыре важные понятия, которые влияют на процесс восстановления:

- *Протокол лог с опережением записи* гарантирует, что журналы транзакций всегда записываются до того, как какие-либо данные базы данных будут фактически обновлены. Этот протокол гарантирует, что в случае сбоя база данных может быть впоследствии восстановлена до согласованного состояния с использованием данных в журнале транзакций.
- *Избыточные журналы транзакций* (несколько копий журнала транзакций) гарантируют, что сбой физического диска не повлияет на способность СУБД восстанавливать данные.
- *Буферы базы данных* – временные области в основной памяти, используемые для ускорения дисковых операций. Для сокращения времени обработки программное обеспечение СУБД считывает данные с физического диска и сохраняет их копию в «буфере» в памяти. Когда транзакция обновляет данные, она фактически обновляет копию данных в буфере, потому что этот процесс намного быстрее, чем каждый раз доступ к физическому диску. Буферы, которые содержат обновленные данные, записываются на физический диск во время одной операции, тем самым экономя значительное время обработки.

- *Контрольные точки базы данных* – это операции, в которых СУБД записывает все свои обновленные буферы в память (также известные как грязные буферы) на диск. Пока это происходит, СУБД не выполняет никаких других запросов. Операция контрольной точки также регистрируется в журнале транзакций. В результате этой операции физическая база данных и журнал транзакций будут синхронизированы. Эта синхронизация необходима, потому что операции обновления обновляют копию данных в буферах, а не в физической базе данных. Контрольные точки автоматически и периодически выполняются СУБД в соответствии с определенными операционными параметрами (такими как верхний водяной знак для размера журнала транзакций или объема незавершенных транзакций), но также могут выполняться явно (как часть оператора транзакции базы данных) или неявно (как часть операции резервного копирования базы данных). Конечно, слишком частые контрольные точки могут повлиять на производительность транзакций; слишком редкие контрольные точки могут повлиять на производительность восстановления базы данных. В любом случае, контрольно-пропускные пункты выполняют очень практическую функцию. Контрольные точки также играют важную роль в восстановлении транзакций.

Процесс восстановления базы данных включает в себя приведение базы данных в согласованное состояние после сбоя. Процедуры восстановления транзакций обычно используют методы отложенной записи и сквозной записи.

Когда процедура восстановления использует *метод отложенной записи* (также называемый *отложенным обновлением*), операции транзакции не сразу обновляют физическую базу данных. Вместо этого обновляется только журнал транзакций. База данных физически обновляется только данными из зафиксированных транзакций, используя информацию из журнала транзакций. Если транзакция прерывается до того, как достигнет точки фиксации, в базу данных не нужно вносить никаких изменений (без ROLLBACK), поскольку она никогда не обновлялась. Процесс восстановления для всех запущенных и подтвержденных транзакций (до сбоя) состоит из следующих шагов:

1. Определить последнюю контрольную точку в журнале транзакций. Это последний раз, когда данные транзакции были физически сохранены на диск.
2. Для транзакции, которая началась и была зафиксирована до последней контрольной точки, ничего делать не нужно, поскольку данные уже сохранены.
3. Для транзакции, которая выполнила операцию фиксации после последней контрольной точки, СУБД использует записи журнала транзакций, чтобы повторить транзакцию и обновить базу данных, используя значения «после» в журнале транзакций. Изменения вносятся в порядке возрастания, от самого старого до самого нового.
4. Для любой транзакции, в которой выполнялась операция ROLLBACK после последней контрольной точки или которая оставалась активной (без COMMIT или

ROLLBACK) до возникновения ошибки, ничего не нужно делать, поскольку база данных никогда не обновлялась.

Когда процедура восстановления использует *метод сквозной записи* (также называемый *немедленным обновлением*), база данных немедленно обновляется операциями транзакции во время выполнения транзакции, даже до того, как транзакция достигает своей точки фиксации. Если транзакция прерывается до того, как достигнет точки фиксации, необходимо выполнить операцию ROLLBACK, чтобы восстановить базу данных в согласованное состояние. В этом случае операция ROLLBACK будет использовать значения «до» журнала транзакций. Процесс восстановления состоит из следующих шагов:

1. Определение последней контрольной точки в журнале транзакций. Это последний раз, когда данные транзакции были физически сохранены на диск.
2. Для транзакции, которая началась и была зафиксирована до последней контрольной точки, ничего делать не нужно, поскольку данные уже сохранены.
3. Для транзакции, которая была зафиксирована после последней контрольной точки, СУБД повторно выполняет транзакцию, используя значения «после» журнала транзакций. Изменения применяются в порядке возрастания, от самого старого до самого нового.
4. Для любой транзакции, в которой выполнялась операция ROLLBACK после последней контрольной точки или которая оставалась активной (без COMMIT или ROLLBACK) до возникновения ошибки, СУБД использует записи журнала транзакций для ROLLBACK или отмены операций, используя значения «до» в журнале транзакций. Изменения применяются в обратном порядке, от самых новых до самых старых.

7.7. Итоги

- Транзакция – это последовательность операций базы данных, которые обращаются к базе данных. Транзакция – это логическая единица работы; то есть все части выполняются или отменяется. Транзакция переводит базу данных из одного согласованного состояния в другое. Согласованное состояние базы данных – это состояние, в котором все ограничения целостности данных выполняются.
- Транзакции имеют четыре основных свойства: атомарность, согласованность, изолированность и долговечность. Атомарность означает, что все части транзакции должны быть выполнены; в противном случае транзакция прерывается. Согласованность – когда согласованное состояние базы данных поддерживается. Изолированность означает, что данные, используемые одной транзакцией, не могут быть доступны другой транзакции, пока первая не будет завершена, а долговечность – что изменения, внесенные транзакцией, не могут быть отменены после фиксации транзакции. Кроме того, расписания транзакций обладают свойством сериализуемости – результат одновременного выполнения транзакций такой же, как и у транзакций, выполняемых в последовательном порядке.

- SQL обеспечивает поддержку транзакций с помощью двух операторов: COMMIT, который сохраняет изменения на диске, и ROLLBACK, который восстанавливает предыдущее состояние базы данных. Транзакции SQL формируются несколькими операторами SQL или запросами базы данных. Каждый запрос к базе данных инициирует несколько операций с базой данных ввода/вывода. Журнал транзакций отслеживает все транзакции, которые изменяют базу данных. Информация, хранящаяся в журнале транзакций, используется для целей восстановления (ROLLBACK).
- Контроль параллелизма координирует одновременное выполнение транзакций. Одновременное выполнение транзакций может привести к трем основным проблемам: потерянные обновления, незафиксированные данные и несогласованные поиски. Планировщик отвечает за установление порядка, в котором выполняются параллельные транзакции. Порядок выполнения транзакций имеет решающее значение и обеспечивает целостность базы данных в многопользовательских системах баз данных. Планировщик использует методы блокировки, отметки времени и оптимистичные методы для обеспечения сериализуемости транзакций.
- Блокировка гарантирует уникальный доступ к элементу данных посредством транзакции. Блокировка не позволяет одной транзакции использовать элемент данных, в то время как другая транзакция использует его. Существует несколько уровней блокировки: база данных, таблица, страница, строка и поле. В системах баз данных могут использоваться два типа блокировок: бинарные блокировки и общие/эксклюзивные блокировки. Двоичная блокировка может иметь только два состояния: заблокировано (1) или разблокировано (0). Общая блокировка используется, когда транзакция хочет прочитать данные из базы данных, и никакая другая транзакция не обновляет эти данные. Для определенного элемента могут существовать несколько общих или «читаемых» блокировок. Исключительная блокировка выдается, когда транзакция хочет обновить (записать) базу данных, и никакие другие блокировки (общие или исключительные) не сохраняются для данных.
- Сериализуемость графиков гарантируется благодаря использованию двухфазной блокировки. Схема двухфазной блокировки имеет растущую фазу, в которой транзакция получает все необходимые ей блокировки без разблокировки каких-либо данных, и фазу сжатия, в которой транзакция снимает все блокировки без получения новых блокировок. Когда две или более транзакции бесконечно ждут друг друга, чтобы снять блокировку, они оказываются в тупике, также называемом смертельным объятием. Существует три метода контроля тупиковых ситуаций: предотвращение, обнаружение и избегание.
- Управление параллелизмом с использованием меток времени назначает уникальную метку времени каждой транзакции и планирует выполнение

конфликтующих транзакций в порядке меток времени. Для определения того, какая транзакция откатывается, а какая продолжает выполняться, используются две схемы: схема ожидания/завершения и схема обмотки/ожидания.

- Контроль параллелизма с оптимистическими методами предполагает, что большинство транзакций базы данных не конфликтуют и что транзакции выполняются одновременно с использованием частных временных копий данных. Во время фиксации личные копии обновляются в базе данных. Стандарт ANSI определяет четыре уровня изоляции транзакции: чтение незафиксировано, чтение зафиксировано, повторное чтение и сериализуемое.
- Восстановление базы данных восстанавливает базу данных из заданного состояния в предыдущее согласованное состояние. Восстановление базы данных запускается при возникновении критического события, такого как аппаратная ошибка или ошибка приложения.

Лекция 8. Оптимизация БД

8.1. Введение

Настройка производительности базы данных является критически важной темой, однако она обычно получает минимальное освещение в учебной программе по базам данных. Большинство баз данных, используемых в университете, имеют только несколько записей на таблицу. В результате основное внимание часто уделяется тому, чтобы запросы SQL выполняли намеченную задачу без учета эффективности процесса запроса. Фактически, даже самая эффективная среда запросов не дает видимых улучшений производительности по сравнению с наименее эффективной средой запросов, когда запрашивается только 20 или 30 строк (записей) таблицы. Отсутствие внимания к эффективности запросов может привести к недопустимо медленным результатам в реальном мире, когда запросы выполняются для десятков миллионов записей.

Одной из основных функций системы баз данных является предоставление своевременных ответов пользователям. Пользователи взаимодействуют с СУБД с помощью запросов для генерации информации, используя следующую последовательность:

1. Клиентское приложение генерирует запрос.
2. Запрос отправляется в СУБД (серверная часть).
3. СУБД (серверная часть) выполняет запрос.
4. СУБД отправляет результирующий набор данных в приложение (на стороне клиента).

Пользователи ожидают, что их запросы будут возвращать результаты как можно быстрее. Хорошую производительность базы данных сложно оценить. Как узнать, достаточно ли времени ответа на запрос в 1,06 секунды? Легче определить плохую производительность базы данных, чем хорошую – достаточно лишь жалоб пользователя на медленные результаты запросов. К сожалению, один и тот же запрос может работать хорошо один день и не так хорошо два месяца спустя. Независимо от восприятия пользователя, целью производительности базы данных является выполнение запросов максимально быстро. Поэтому производительность базы данных должна тщательно контролироваться и регулярно настраиваться. *Настройка производительности базы данных* относится к набору действий и процедур, предназначенных для сокращения времени отклика системы базы данных, то есть для обеспечения того, чтобы запрос пользователя обрабатывался СУБД в минимальное время.

Время, необходимое запросу для возврата набора результатов, зависит от многих факторов, которые, как правило, имеют широкий диапазон и зависят от среды и поставщиков. В целом, производительность типичной СУБД ограничена тремя основными факторами: вычислительной мощностью процессора, доступной оперативной памятью и пропускной способностью ввода/вывода (жесткого диска и сети).

Естественно, система будет работать лучше, когда ее аппаратные и программные ресурсы будут оптимизированы. Однако эти ресурсы ограничены. Системные компоненты

должны быть оптимизированы для получения максимально возможной пропускной способности с существующими (и часто ограниченными) ресурсами, поэтому настройка производительности базы данных важна.

Точная настройка производительности системы требует целостного подхода. Все факторы должны быть проверены, чтобы гарантировать, что каждый из них работает на своем оптимальном уровне и имеет достаточные ресурсы, чтобы минимизировать возникновение узких мест. Архитектура базы данных является важным фактором, определяющим эффективность работы системы баз данных.

Хорошая производительность БД начинается с хорошего проекта. Никакие настройки не приведут к тому, что плохо спроектированная БД будет работать так же хорошо, как и хорошо спроектированная. Это особенно верно при перепроектировании существующих БД, когда пользователь ожидает нереального прироста производительности от старых.

Что представляет собой хороший, эффективный проект БД? С точки зрения настройки производительности, разработчик БД должен убедиться, что в проекте используются функции СУБД, которые гарантируют целостность и оптимальную производительность.

Настройка производительности: клиент и сервер. Действия по настройке производительности БД можно разделить на действия на стороне клиента и на стороне сервера.

- На стороне клиента цель состоит в том, чтобы сгенерировать запрос SQL, который возвращает правильный ответ за наименьшее время, используя минимальное количество ресурсов на стороне сервера. Действия, необходимые для достижения этой цели, обычно называют *настройкой производительности SQL*.
- На стороне сервера – СУБД должна быть правильно настроена для максимально быстрого ответа на запросы клиентов при оптимальном использовании существующих ресурсов. Действия, необходимые для достижения этой цели, обычно называют *настройкой производительности СУБД.*

Реализации СУБД, как правило, более сложны, чем просто двухуровневая конфигурация "клиент/сервер". Сетевой компонент играет важную роль в доставке сообщений между клиентами и серверами; это особенно важно в распределенных БД. В многоуровневых клиент-серверных средах, которые состоят из клиентской части, промежуточного программного обеспечения приложений и серверной части БД, действия по настройке производительности часто делятся на подзадачи, чтобы обеспечить максимально быстрое время отклика между любыми двумя компонентными точками. Администратор БД должен тесно сотрудничать с сетевой группой, чтобы обеспечить эффективную передачу трафика в сетевой инфраструктуре. Это еще более важно, если учесть, что большинство систем БД обслуживают географически разрозненных пользователей.

Архитектура СУБД. Архитектура СУБД представлена процессами и структурами (в оперативной и постоянной памяти), используемыми для управления базой данных. Такие процессы взаимодействуют друг с другом для выполнения определенных функций. На рисунке 27 показана архитектура СУБД.

Рис. 27. Архитектура СУБД

Необходимо обратить внимание на следующие компоненты и функции на рисунке 27:

- Все данные в базе данных хранятся в *файлах данных*. Типичная база данных предприятия обычно состоит из нескольких файлов данных. Файл данных может содержать строки из одной таблицы или строки из разных таблиц. Администратор базы данных (DBA) определяет начальный размер файлов данных, которые составляют базу данных; однако файлы данных могут автоматически расширяться по мере необходимости с заранее определенными приращениями, известными как *экстенты*. Например, если требуется больше места, администратор базы данных может определить, что каждый новый экстент будет с шагом 10 КБ или 10 МБ.
- Файлы данных обычно группируются в файловые группы или табличные пространства. *Табличное пространство* или *группа файлов* – логическая группа из нескольких файлов данных, в которых хранятся данные со сходными характеристиками. Например, в системном табличном пространстве хранятся метаданные, табличное пространство пользовательских данных для хранения созданных пользователем таблиц, табличное пространство индексов для хранения всех индексов и временное табличное пространство для временных сортировок, группировка и т.д. Каждый раз, когда создается новая БД, СУБД автоматически создает минимальный набор табличных пространств.
- *Кэш данных* или *буферный кэш*, является общей, зарезервированной областью памяти, в которой хранятся последние доступные блоки данных в ОЗУ. Данные, считанные из файлов данных, сохраняются в кэше данных после того, как данные были прочитаны или до того, как данные были записаны в файлы данных. Кэш данных также кэширует данные системного каталога и содержимое индексов.
- *Кэш SQL*, или *кэш процедур*, является общей зарезервированной областью памяти, в которой хранятся последние выполненные операторы SQL или процедуры, включая триггеры и функции. Кэш SQL не хранит код пользователя. Вместо этого в кэше хранится «обработанная» версия SQL, которая готова к выполнению СУБД.

- Для работы с данными СУБД должна извлечь данные из постоянного хранилища и поместить их в оперативную память. Другими словами, данные извлекаются из файлов данных и помещаются в кэш данных.
- Для перемещения данных из постоянного хранилища (файлы данных) в ОЗУ (кэш данных) СУБД выдает запросы ввода-вывода и ожидает ответы. *Запрос ввода/вывода (I/O)* – это операция доступа к данным низкого уровня, которая считывает или записывает данные на физические носители, такие как память, жесткие диски, видеоустройства и принтеры. Операция чтения с диска ввода-вывода извлекает весь блок физического диска, обычно содержащий несколько строк, из постоянного хранилища в кэш данных, даже если нужен только один атрибут только из одной строки. Размер блока физического диска зависит от операционной системы и может быть 4КБ, 8КБ, 16КБ, 32КБ, 64КБ или даже больше. Кроме того, в зависимости от обстоятельств, СУБД может выдавать одноблочный запрос на чтение или многоблочный запрос на чтение.
- Работа с данными в кэше данных во много раз быстрее, чем работа с данными в файлах данных, поскольку СУБД не нужно ждать, пока жесткий диск извлечет данные; для работы в кэше данных не требуются операции ввода-вывода жесткого диска.
- Большинство действий по настройке производительности направлены на минимизацию количества операций ввода-вывода, поскольку использование операций ввода-вывода во много раз медленнее, чем чтение данных из кэша данных. Например, на момент написания статьи время доступа к ОЗУ варьировалось от 5 до 50 наносекунд, а время доступа к магнитному жесткому диску – от 5 до 15 миллисекунд, а время доступа к SSD – от 35 до 100 микросекунд. Это означает, что жесткие диски на несколько порядков медленнее, чем ОЗУ.

На рисунке 27 также показаны некоторые типичные процессы СУБД. Хотя число процессов и их имена варьируются от поставщика к поставщику, функциональность аналогична. Следующие процессы представлены на рисунке 27:

- *Обработчик запросов*. Обработчик принимает запросы клиентов и обрабатывает их. Как только запрос получен, обработчик передает запрос соответствующему пользовательскому процессу.
- *Процесс пользователя*. СУБД создает пользовательский процесс для управления каждым сеансом клиента. Поэтому при входе в СУБД назначается пользовательский процесс. Этот процесс обрабатывает все запросы, которые клиент отправляет на сервер. Существует много пользовательских процессов, по крайней мере, один на каждого зарегистрированного клиента.
- *Планировщик*. Процесс планировщика организует параллельное выполнение запросов SQL.

- *Менеджер блокировок*. Этот процесс управляет всеми блокировками, размещенными на объектах базы данных, включая страницы на диске.
- *Оптимизатор*. Процесс оптимизатора анализирует запросы SQL и находит наиболее эффективный способ доступа к данным.

Режимы оптимизации запросов к БД. Большинство алгоритмов, предложенных для оптимизации запросов, основано на двух принципах:

- Выбор оптимального порядка исполнения для достижения максимально быстрого времени исполнения.
- Выбор источников данных для доступа, чтобы минимизировать расходы на связь.

В рамках этих двух принципов алгоритм оптимизации запросов может оцениваться на основе режима его работы или времени его оптимизации.

Режимы работы можно классифицировать как ручные или автоматические. *Автоматическая оптимизация запросов* означает, что СУБД находит наиболее эффективный путь доступа без вмешательства пользователя. *Ручная оптимизация запросов* требует, чтобы оптимизация была выбрана и запланирована пользователем. С точки зрения пользователя, автоматическая оптимизация запросов явно более желательна, но цена такого удобства заключается в повышенных накладных расходах, которые она накладывает на СУБД.

Алгоритмы оптимизации запросов также могут быть классифицированы в зависимости от того, когда выполняется оптимизация. В рамках этой временной классификации алгоритмы оптимизации запросов могут быть статическими или динамическими.

- *Оптимизация статического запроса* происходит во время компиляции. Другими словами, лучшая стратегия оптимизации выбирается при составлении запроса СУБД. Этот подход распространен, когда операторы SQL встроены в процедурные языки программирования, такие как C# или Java. Когда программа передается в СУБД для компиляции, она создает план, необходимый для доступа к базе данных. Когда программа выполняется, СУБД использует этот план для доступа к базе данных.
- *Динамическая оптимизация запросов* происходит во время выполнения. Стратегия доступа к базе данных определяется при запуске программы. Таким образом, стратегия доступа динамически определяется СУБД во время выполнения, используя самую актуальную информацию о базе данных. Хотя динамическая оптимизация запросов эффективна, ее стоимость измеряется издержками обработки во время выполнения. Лучшая стратегия определяется каждый раз, когда выполняется запрос; это может происходить несколько раз в одной и той же программе.

Наконец, методы оптимизации запросов могут быть классифицированы в соответствии с типом информации, которая используется для оптимизации запроса. Например, запросы могут быть основаны на статистике или на правилах алгоритмов.

- *Алгоритм оптимизации запросов на основе статистики* использует статистическую информацию о БД. Статистика предоставляет информацию о свойствах БД, таких как размер, количество записей, среднее время доступа, количество обслуживаемых запросов и количество пользователей с правами доступа. Эти статистические данные затем используются СУБД для определения наилучшей стратегии доступа. В оптимизаторах, основанных на статистике, некоторые СУБД позволяют установить цель, чтобы указать, что оптимизатор должен пытаться минимизировать время для извлечения первой или последней строки. Минимизация времени получения первой строки часто используется в системах транзакций и интерактивных клиентских средах. В этих случаях цель состоит в том, чтобы представить первые несколько строк пользователю как можно быстрее. Затем, пока СУБД ожидает от пользователя прокрутки данных, она может извлечь другие строки для запроса. Установка цели оптимизатора для минимизации извлечения последней строки обычно выполняется во встроенном SQL и внутри хранимых процедур. В этих случаях элемент управления не будет возвращаться вызывающему приложению, пока не будут получены все данные; поэтому важно, как можно быстрее извлечь все данные из последней строки, чтобы можно было вернуть управление.
- Статистическая информация управляется СУБД и генерируется в одном из двух разных режимов: динамическом или ручном. В *режиме динамического генерирования статистики* СУБД автоматически оценивает и обновляет статистику после каждой операции доступа к данным. В *режиме генерации статистики вручную* статистика должна периодически обновляться с помощью выбранной пользователем утилиты, такой как команда IBM RUNSTAT, которая используется СУБД DB2.
- *Алгоритм оптимизации запросов на основе правил* основан на наборе пользовательских правил для определения наилучшей стратегии доступа к запросам. Правила вводятся пользователем или администратором БД, и они обычно носят общий характер.

Статистика базы данных. Еще один процесс СУБД, который играет важную роль в оптимизации запросов, – сбор статистики БД. Термин *статистика базы данных* относится к ряду измерений об объектах БД, таких как число используемых процессоров, скорость процессора и доступное временное пространство. Такая статистика предоставляет снимок характеристик БД.

СУБД использует эту статистику для принятия важных решений по повышению эффективности обработки запросов. Статистика БД может быть собрана вручную администратором или автоматически. Например, многие поставщики СУБД поддерживают команду ANALYZE в SQL для сбора статистики. Кроме того, многие поставщики имеют свои собственные процедуры для сбора статистики. Например, IBM DB2 использует процедуру RUNSTATS, в то время как Microsoft SQL Server использует процедуру UPDATE STATISTICS

и предоставляет параметры автоматического обновления и автоматического создания статистики в своих параметрах инициализации. Примеры параметров, которые СУБД может собирать для разных объектов базы данных, показан в таблице 27.

Таблица 27

Примеры параметров для статистики БД

Объект БД	Примерные параметры
Таблица	Количество строк, количество используемых дисковых блоков, длина строки, количество столбцов в каждой строке, количество отдельных значений в каждом столбце, максимальное и минимальное значение в каждом столбце, и столбцы, имеющие индексы.
Индекс	Количество и имя столбцов в ключе индекса, количество значений ключа в индексе, количество отдельных значений ключа в ключе индекса, гистограмма значений ключа в индексе и количество страниц диска, используемых индексом.
Ресурс	Размер блока логического и физического диска, расположение и размер файлов данных, а также количество расширений на файл данных.

Если статистика объектов существует, СУБД будет использовать ее при обработке запросов. Большинство новых СУБД (таких как Oracle, MySQL, SQL Server и DB2) автоматически собирают статистику; другие требуют, чтобы администратор БД собирал статистику вручную. Для генерации статистики объектов базы данных вручную каждая СУБД имеет свои собственные команды.

В Oracle: ANALYZE <TABLE/INDEX> имя_объекта COMPUTE STATISTICS;

В MySQL: ANALYZE TABLE <имя_таблицы>

В SQL Server: UPDATE STATISTICS <имя_объекта>, где имя объекта относится к таблице или представлению.

Например, чтобы сгенерировать статистику для таблицы ПОСТАВЩИК, необходимо использовать:

В Oracle: ANALYZE TABLE ПОСТАВЩИК COMPUTE STATISTICS;

В MySQL: ANALYZE TABLE ПОСТАВЩИК;

В SQL Server: UPDATE STATISTICS ПОСТАВЩИК.

Когда генерируется статистика для таблицы, все связанные индексы также анализируются. Однако можно сгенерировать статистику для одного индекса, используя следующую команду, где ПОСТ_NDX – это имя индекса:

ANALYZE INDEX ПОСТ_NDX COMPUTE STATISTICS;

В SQL Server нужно использовать UPDATE STATISTICS. Примером команды может быть UPDATE STATISTICS ПОСТАВЩИК ПОСТ_NDX.

Статистика БД хранится в системном каталоге в специально отведенных таблицах. Обычно периодически восстанавливают статистику для объектов базы данных, особенно для тех, которые часто изменяются. Например, если есть СУБД для проката видео, система, вероятно, будет использовать таблицу АРЕНДА для хранения ежедневных прокатов видео. Эта таблица, и связанные с ней индексы, будут подвергаться постоянным вставкам и

обновлениям, когда регистрируется ежедневные арендные платы и возвраты. Таким образом, статистика таблицы АРЕНДА, которую сгенерировали на прошлой неделе, точно не отображает таблицу в том виде, в каком она существует сегодня. Чем актуальнее статистика, тем больше шансов, что СУБД правильно выберет самый быстрый способ выполнения данного запроса.

8.2. Обработка запросов

При получении клиентской команды SQL, СУБД обрабатывает ее в три этапа:

1. *Разбор*. СУБД анализирует запрос SQL и выбирает наиболее эффективный план доступа/выполнения.
2. *Исполнение*. СУБД выполняет запрос SQL с использованием выбранного плана выполнения.
3. *Доставка*. СУБД выбирает данные и отправляет набор результатов обратно клиенту.

Обработка операторов SQL DDL (таких как CREATE TABLE) отличается от обработки, требуемой операторами DML. Разница в том, что оператор DDL фактически обновляет таблицы словаря данных или системный каталог, тогда как оператор DML (SELECT, INSERT, UPDATE или DELETE) в основном манипулирует данными пользователя. На рисунке 28 показаны общие шаги, необходимые для обработки запросов.

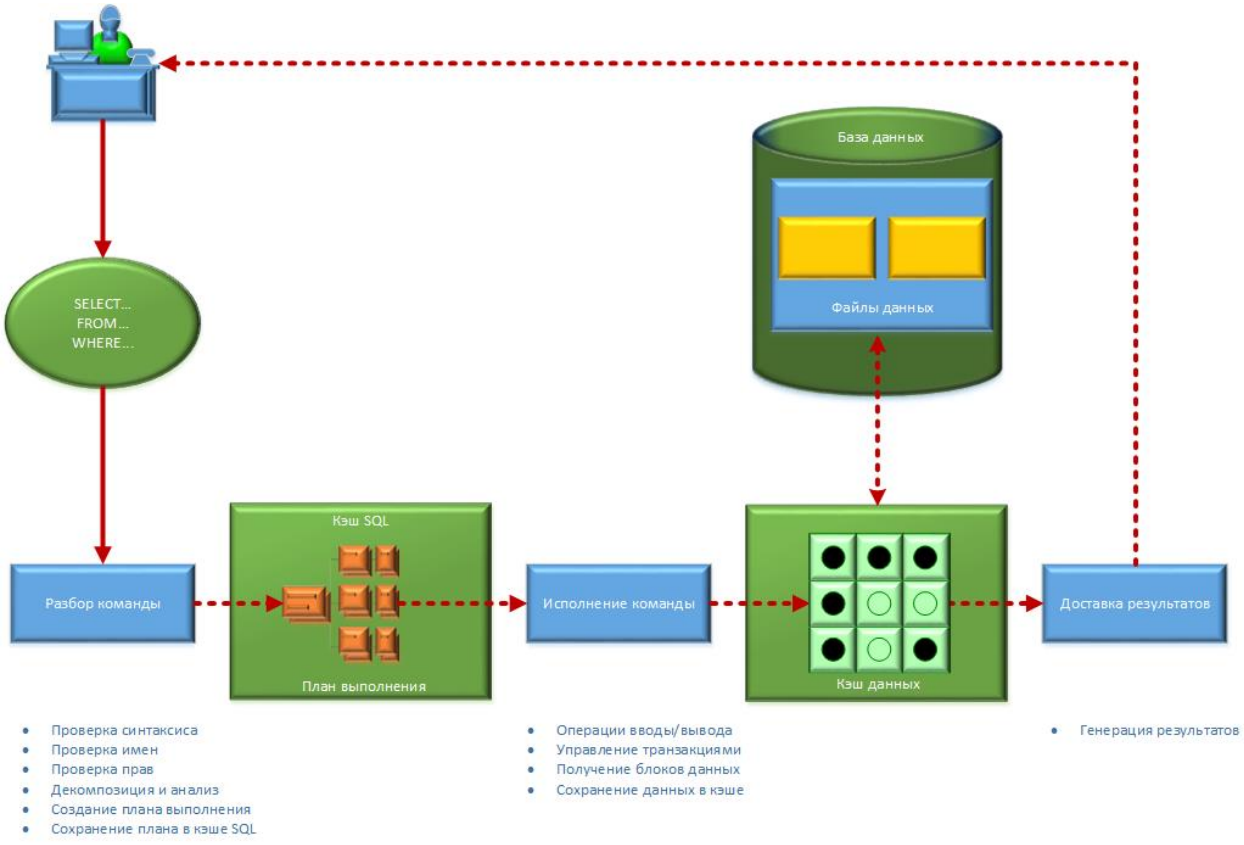


Рис. 28. Выполнение запроса

Фаза синтаксического анализа. Процесс оптимизации включает в себя разбор запроса на более мелкие единицы и преобразование исходного запроса SQL в несколько иную версию исходного кода SQL, но полностью эквивалентную и более эффективную. Полностью эквивалентную означает, что оптимизированные результаты запроса всегда совпадают с исходным запросом. Более эффективную означает, что оптимизированный запрос почти всегда будет выполняться быстрее, чем исходный запрос. Чтобы определить наиболее эффективный способ для выполнения запроса СУБД, можно использовать статистику базы данных.

Действия синтаксического анализа SQL выполняются *оптимизатором запросов*, который анализирует запрос и находит наиболее эффективный способ доступа к данным. Этот процесс является наиболее трудоемким этапом в обработке запросов. Разбор SQL-запроса требует нескольких шагов, в которых SQL-запрос:

- Проверяется на соответствие синтаксису;
- Проверяется по словарю данных, чтобы убедиться, что имена таблиц и столбцов правильные;
- Проверяется по словарю данных, чтобы гарантировать, что пользователь имеет надлежащие права доступа;
- Анализируются и разлагаются на более атомарные компоненты;
- Оптимизируется за счет преобразования в полностью эквивалентный, но более эффективный SQL-запрос;
- Подготавливается к исполнению путем определения наиболее эффективного плана выполнения или доступа

После преобразования оператора SQL СУБД создает план выполнения. *План выполнения* – результат анализа оператора SQL; он содержит последовательность шагов, которые СУБД будет использовать для выполнения запроса и возврата набора результатов наиболее эффективным способом. Сначала СУБД проверяет, существует ли уже план доступа для запроса в кэше SQL. Если это так, СУБД повторно использует план доступа для экономии времени. Если это не так, оптимизатор оценивает разные планы, а затем решает, какие индексы использовать и как лучше всего выполнять операции объединения. Выбранный план выполнения для запроса затем помещается в кэш SQL, и становится доступным для использования и последующего повторного использования.

Планы выполнения специфичны для СУБД и переводят SQL-запрос клиента в серию сложных операций ввода-вывода, необходимых для чтения данных из физических файлов данных и генерации набора результатов.

Фаза выполнения. На этом этапе выполняются все операции ввода-вывода, указанные в плане выполнения. Когда выполняется план, для доступа к данным приобретаются надлежащие блокировки – при необходимости – данные извлекаются из файлов и помещаются в кэш. Все команды управления транзакциями обрабатываются на этапах анализа и выполнения обработки запросов.

Фаза доставки результатов. После завершения этапов синтаксического анализа и выполнения все строки, соответствующие указанным условиям, извлекаются, сортируются, группируются и, если требуется, агрегируются. СУБД может использовать временное табличное пространство для хранения временных данных. На этом этапе сервер БД координирует перемещение строк результирующего набора из кэша сервера в кэш клиента. Например, данный набор результатов запроса может содержать 9 000 строк; сервер будет отправлять первые 100 строк клиенту, а затем ждать, пока клиент запросит следующий набор строк, пока весь набор результатов не будет отправлен клиенту.

Узкие места обработки запросов. Основная задача обработки запросов – выполнить данный запрос максимально быстрым способом с наименьшим количеством ресурсов. Выполнение запроса требует, чтобы СУБД разбивала запрос на серию операций ввода-вывода, которые должны выполняться совместно. Чем сложнее запрос, тем сложнее операции, что означает, что узкие места более вероятны. Узким местом обработки запросов является задержка, возникающая при обработке операции ввода-вывода, которая приводит к замедлению работы всей системы. Таким же образом, чем больше компонентов в системе, тем больше требуется взаимодействия между компонентами, что увеличивает вероятность узких мест. В СУБД пять компонентов обычно вызывают узкие места:

- *ЦП*. Мощность центрального процессора СУБД должна соответствовать ожидаемой рабочей нагрузке системы. Высокая загрузка ЦП может указывать на то, что скорость процессора слишком мала для объема выполненной работы. Однако высокая загрузка ЦП может быть вызвана другими факторами, такими как неисправный компонент, нехватка ОЗУ (ЦП тратит слишком много времени на перестановку блоков памяти), плохо написанный драйвер устройства или вредоносный процесс. Узкое место ЦП повлияет не только на СУБД, но и на все процессы, работающие в системе.
- *ОЗУ*. СУБД выделяет память для конкретного использования, такого как кэш данных и кэш SQL. Оперативная память должна быть разделена между всеми запущенными процессами, включая операционную систему и СУБД. Если недостаточно ОЗУ, перемещение данных между компонентами, конкурирующими за дефицит ОЗУ, может создать узкое место.
- *Жесткий диск*. Другими распространенными причинами узких мест являются скорость жесткого диска и скорость передачи данных. Современная технология хранения на жестком диске обеспечивает большую емкость, чем в прошлом; однако пространство на жестком диске используется не только для хранения данных конечного пользователя. Современные операционные системы также используют жесткий диск для виртуальной памяти, что означает копирование областей оперативной памяти на жесткий диск по мере необходимости, чтобы освободить место в оперативной памяти для более неотложных задач. Следовательно, больше места на жестком диске и более высокая скорость передачи данных уменьшают вероятность узких мест.

- *Сеть.* В среде базы данных сервер БД и клиенты связаны через сеть. Все сети имеют ограниченную пропускную способность, которая распределяется между всеми клиентами. Когда множество сетевых узлов одновременно получают доступ к сети, возможны узкие места.
- *Код приложения.* Не все узкие места вызваны ограниченными аппаратными ресурсами. Двумя наиболее распространенными источниками узких мест являются неполноценный код приложения и плохо спроектированные БД. Неправильный код может быть улучшен с помощью методов оптимизации кода, при условии, что основной проект БД является надежным. Однако никакое количество кодирования не сделает плохо спроектированную БД более эффективной.

Узкие места являются результатом многочисленных транзакций БД, конкурирующих за использование ресурсов базы данных (ЦП, ОЗУ, жесткий диск, индексы, блокировки, буферы и т.д.). СУБД использует множество компонентов и структур для выполнения своих операций, таких как процессы, буферы, блокировки, табличные пространства, индексы и файлы журналов. Эти ресурсы используются всеми транзакциями, выполняемыми в базе данных, и, следовательно, транзакции часто конкурируют за такие ресурсы. Поскольку большинство (если не все) транзакции работают со строками данных в таблицах, одним из наиболее типичных узких мест являются транзакции, конкурирующие за одни и те же строки данных. Другим распространенным источником разногласий являются ресурсы общей памяти, в частности, общие буферы и блокировки. Чтобы ускорить операции обновления данных, СУБД использует буферы для кэширования данных. В то же время для управления доступом к данным СУБД использует блокировки.

8.3. Индексы и оптимизация запросов

Индексы имеют решающее значение для ускорения доступа к данным, поскольку они облегчают поиск, сортировку и использование агрегатных функций и даже операции объединения. Повышение скорости доступа к данным происходит потому, что индекс представляет собой упорядоченный набор значений, который содержит ключ индекса и указатели. Указатели – это идентификаторы строк для фактических строк таблицы.

Сканирование индекса более эффективно, чем полное сканирование таблицы, поскольку данные индекса предварительно упорядочены, а объем данных обычно намного меньше. Поэтому при выполнении поиска СУБД почти всегда лучше использовать индекс для доступа к таблице, чем последовательно сканировать все строки в таблице. Например, на рисунке 29 показано представление индекса таблицы КЛИЕНТ с 40000 строками и индексом ГРД_NDX для атрибута КЛН_ГРД.

Рис. 29. Индекс для таблицы КЛИЕНТ

Примерный запрос к таблице КЛИЕНТ:

```
SELECT КЛН_ФАМ, КЛН_АДР, КЛН_ТЕЛ
FROM КЛИЕНТ
WHERE КЛН_ГРД = 'Нижневартовск'.
```

Если индекс отсутствует, СУБД выполнит полное сканирование таблицы и прочитает все 40000 строк клиентов. Предполагая, что индекс ГРД_NDX создан и проанализирован, СУБД автоматически использует индекс для определения местоположения первого клиента из города Нижневартовск, а затем приступает к чтению всех последующих строк таблицы КЛИЕНТ, используя идентификаторы строк в индексе. Предполагая, что только пять строк удовлетворяют условию КЛН_ГРД = 'Нижневартовск', существует пять обращений к индексу и пять обращений к данным, всего 10 обращений ввода-вывода. СУБД будет спасена от чтения приблизительно 40000 запросов ввода-вывода, которые не соответствуют критериям. Это много циклов процессора!

Если индексы так важны, почему бы не индексировать каждый столбец в каждой таблице? Простой ответ заключается в том, что это непрактично. Индексирование каждого столбца в каждой таблице перегружает СУБД с точки зрения обработки обслуживания индекса, особенно если таблица имеет много атрибутов и строк или требует много вставок, обновлений и удалений.

Одной из мер, которая определяет необходимость индекса, является разреженность данных столбца, который надо проиндексировать. *Разреженность данных* относится к числу разных значений, которые может иметь столбец. Например, столбец СТД_ПОЛ в таблице СТУДЕНТ может иметь только два возможных значения «М» или «Ж»; следовательно, этот столбец имеет низкую разреженность. Напротив, столбец СТД_ДР, в котором хранится дата рождения студентов, может иметь много разных значений дат; следовательно, этот столбец имеет высокую разреженность. Знание разреженности помогает решить, подходит ли использование индекса.

Большинство СУБД реализуют индексы, используя одну из следующих структур данных:

- ***Хеш-индекс***. Индекс хеша основан на упорядоченном списке значений хеша. Алгоритм хеширования используется для создания значения хеш-функции из ключевого столбца. Это значение указывает на запись в хеш-таблице, которая, в свою очередь, указывает на фактическое расположение строки данных. Хеш-индекс подходит для простых и быстрых операций поиска, основанных на условиях равенства.
- ***Индекс B-дерева***. Индекс B-дерева – упорядоченная структура данных, организованная как перевернутое дерево. Индексное дерево хранится отдельно от данных. Листы нижнего уровня индекса B-дерева содержат указатели на фактические строки данных. Индексы B-дерева являются «самоуравновешенными», что означает, что для доступа к любой строке в индексе требуется примерно одинаковое количество времени. Это стандартный и самый распространенный тип индекса, используемый в базах данных. Индекс B-дерева используется главным образом в таблицах, в которых значения столбцов повторяются относительно небольшое количество раз.
- ***Битовый индекс.*** В битовом индексе используется битовый массив (0 и 1) для представления существования значения или условия. Эти индексы используются главным образом в приложениях хранилища данных в таблицах с большим количеством строк, в которых небольшое количество значений столбцов повторяется много раз. Битовые индексы, как правило, используют меньше места, чем индексы B-дерева, потому что они используют биты вместо байтов для хранения своих данных.

Используя предыдущие характеристики индекса, разработчик БД может определить наилучший тип индекса для использования. Например, таблица КЛИЕНТ имеет несколько тысяч строк. У таблицы два столбца широко используются для запросов: КЛН_ФАМ, который представляет фамилию клиента, и КЛН_ГРД, который может иметь одно из четырех значений (Нижневартовск, Мегион, Радужный и Лангепас). Основываясь на этой информации, можно сделать вывод, что:

- Поскольку столбец КЛН_ФАМ содержит много разных значений, которые повторяются относительно небольшое количество раз по сравнению с общим числом строк в таблице, будет использоваться индекс B-дерева.
- Поскольку столбец КЛН_ГРД содержит только несколько разных значений, которые повторяются относительно большое количество раз по сравнению с общим числом строк в таблице, будет использоваться битовый индекс.

СУБД текущего поколения достаточно интеллектуальны, чтобы определить оптимальный тип индекса для использования при определенных обстоятельствах, при условии, что обновлена статистика БД. Независимо от того, какой индекс выбран, СУБД

определяет лучший план для выполнения данного запроса. В следующем разделе приведен упрощенный пример типа выбора, который должен сделать оптимизатор запросов.

8.4. Выбор оптимизатора

Оптимизация запросов – это главное действие на этапе анализа при обработке запросов. На этом этапе СУБД должна выбрать, какие индексы использовать, как выполнять операции соединения, какую таблицу использовать первой и т.д. Каждая СУБД имеет свои алгоритмы определения наиболее эффективного способа доступа к данным. Оптимизатор запросов может работать в одном из двух режимов:

- *Оптимизатор на основе правил* использует заданные правила и точки для определения наилучшего подхода к выполнению запроса. Правила назначают «фиксированную стоимость» каждой операции SQL; Затем добавляются затраты, чтобы получить стоимость плана выполнения.
- *Оптимизатор на основе затрат* использует сложные алгоритмы, основанные на статистике об объектах, к которым осуществляется доступ, для определения наилучшего подхода к выполнению запроса. В этом случае процесс оптимизатора складывает стоимость обработки, затраты на ввод-вывод и затраты ресурсов (ОЗУ и временное пространство), чтобы определить общую стоимость заданного плана выполнения.

Задача оптимизатора – найти альтернативные способы выполнения запроса – оценить «стоимость» каждой альтернативы, а затем выбрать наименьшую.

Хотя оптимизатор обычно работает очень хорошо в большинстве случаев, в некоторых случаях оптимизатор может не выбрать лучший план выполнения. Оптимизатор принимает решения на основе существующей статистики. Если статистика старая, оптимизатор может не справиться с выбором лучшего плана выполнения. Даже с текущей статистикой выбор оптимизатора может быть не самым эффективным. Иногда пользователь хотел бы изменить режим оптимизатора для текущего оператора SQL. Для этого нужно использовать подсказки. Подсказки оптимизатора – это специальные инструкции, которые встроены в текст команды.

8.5. Настройка производительности SQL

Настройка производительности SQL оценивается с точки зрения клиента. Поэтому цель состоит в том, чтобы проиллюстрировать некоторые распространенные практики, используемые для написания эффективного кода SQL. Несколько слов предостережения уместны:

- Большинство реляционных СУБД текущего поколения выполняют автоматическую оптимизацию запросов на стороне сервера.
- Большинство методов оптимизации производительности SQL зависят от СУБД и, следовательно, редко переносимы даже в разные версии одной и той же СУБД. Одной из причин такого поведения является постоянное развитие технологий БД.

Означает ли это, что не нужно беспокоиться о том, как пишется SQL-запрос, потому что СУБД всегда будет его оптимизировать? Нет, потому что есть значительные возможности для улучшения. СУБД использует общие методы оптимизации, а не фокусируется на конкретных методах, продиктованных особыми обстоятельствами выполнения запроса. Плохо написанный SQL-запрос может и, как правило, ставит под угрозу систему базы данных с точки зрения производительности. Большинство текущих проблем с производительностью БД связаны с плохо написанным кодом SQL. Следовательно, хотя СУБД предоставляет общие службы оптимизации, тщательно написанный запрос почти всегда превосходит плохо написанный.

Селективность индекса. Индексы являются наиболее важной техникой, используемой при оптимизации производительности SQL. Как правило, индексы могут использоваться:

- Когда индексируемый столбец появляется в критериях поиска предложения WHERE или HAVING;
- Когда индексируемый столбец появляется в предложении GROUP BY или ORDER BY;
- Когда функция MAX или MIN применяется к индексированному столбцу;
- Когда разрежённость данных в индексированном столбце высока.

Индексы очень полезны, когда необходимо выбрать небольшое подмножество строк из большой таблицы на основе заданного условия. Если для столбца, используемого при выборе, существует индекс, СУБД может выбрать его использование. Целью является создание индексов с высокой селективностью. *Селективность индекса* – мера вероятности того, что индекс будет использоваться при обработке запроса. Несколько общих рекомендаций по созданию и использованию индексов:

- *Создавать индексы для каждого отдельного атрибута, используемого в предложении WHERE, HAVING, ORDER BY или GROUP BY.* Если создать индексы по всем отдельным атрибутам, используемым в условиях поиска, СУБД будет обращаться к таблице, используя сканирование индекса вместо полного сканирования таблицы. Например, если есть индекс для ТОВ_ЦЕНА, условие ТОВ_ЦЕНА > 1000 может быть решено путем доступа к индексу вместо последовательного сканирования и проверки всех строк таблицы.
- *Не использовать индексы в небольших таблицах или таблицах с низкой разреженностью.* Маленькие таблицы и таблицы с низким разрежением – это не одно и то же. Условие поиска в таблице с низкой разреженностью может в любом случае вернуть большой процент строк таблицы, что делает операцию индекса слишком дорогостоящей и делает просмотр всей таблицы жизнеспособным вариантом. Используя ту же логику, не нужно создавать индексы для таблиц с несколькими строками и несколькими атрибутами – если только не нужно обеспечить наличие уникальных значений в столбце.
- *Объявлять первичные и внешние ключи, чтобы оптимизатор мог использовать индексы в операциях соединения.* Все естественные объединения и объединения

старого стиля выиграют, если объявить первичные ключи и внешние ключи, потому что оптимизатор будет использовать доступные индексы во время соединения. Объявление первичного или внешнего ключа автоматически создаст индекс для столбца. Кроме того, по той же причине лучше писать соединения, используя синтаксис SQL JOIN.

- *Объявлять индексы в столбцах соединения, отличных от PK или FK.* Если выполняются операции соединения со столбцами, отличными от первичного и внешнего ключей, лучше объявить индексы в этих столбцах.

Не всегда можно использовать индекс для повышения производительности. В некоторых СУБД индексы игнорируются при использовании функций в атрибутах таблицы. Основные СУБД, такие как Oracle, SQL Server и DB2, поддерживают индексы на основе функций. *Индекс на основе функций* – это индекс, основанный на конкретной функции или выражении SQL. Например, можно создать индекс на YEAR(ЧЕК_ДАТА). Основанные на функциях индексы особенно полезны при работе с производными атрибутами. Например, можно создать индекс для СОТР_ЗРП + СОТР_ВЗНС.

Сколько индексов необходимо создать? Следует повторить, что не стоит создавать индекс для каждого столбца в таблице. Слишком большое количество индексов замедлит операции INSERT, UPDATE и DELETE, особенно если таблица содержит много тысяч строк. Кроме того, некоторые оптимизаторы запросов выбирают только один индекс в качестве движущего индекса для запроса, даже если запрос использует условия во многих разных индексированных столбцах. Какой индекс использует оптимизатор? Если используется оптимизатор на основе затрат, ответ будет меняться со временем по мере добавления или удаления новых строк из таблиц. В любом случае нужно создать индексы во всех столбцах поиска и затем позволить оптимизатору выбирать. Важно постоянно оценивать использование индекса – отслеживать, тестировать, оценивать и улучшать его, если производительность недостаточна.

Условные выражения. Условное выражение обычно помещается в предложения WHERE или HAVING оператора SQL. Также как условные критерии, условное выражение ограничивает вывод запроса только теми строками, которые соответствуют условным критериям. Как правило, условные критерии имеют форму, показанную в таблице 28.

Таблица 28

Условные выражения

Первый операнд	Оператор сравнения	Второй операнд
ТОВ_ЦЕНА	>	10.0
КЛН_ГРД	=	'Нижневартовск'
ТОВ_НАЗВ	LIKE	'Ген%'
ТОВ_ОСТ	<=	ТОВ_МИН * 1.1

В таблице 28 операндом может быть:

- Простое имя столбца, например, ТОВ_ЦЕНА или КЛН_ГРД.

- Литерал или константа, например, 10.0 или «Нижневартовск».
- Выражение, например, ТОВ_МИН * 1.1.

Большинство методов оптимизации запросов, упомянутых ниже, предназначены для упрощения работы оптимизатора. Следующие общие практики используются для написания эффективных условных выражений в коде SQL.

- *Использование простых столбцов или литералов в качестве операндов в условном выражении* – по возможности следует избегать использования условных выражений с функциями. Сравнение содержимого одного столбца с литералом происходит быстрее, чем сравнение с выражениями. Например, ТОВ_ЦЕНА > 10.0 быстрее, чем ТОВ_ОСТ > ТОВ_МИН * 1.1, потому что СУБД должна сначала вычислить выражение ТОВ_МИН * 1.1. Использование функций в выражениях также увеличивает общее время выполнения запроса. Например, если условие UPPER(КЛН_ИМЯ) = 'НИКОЛАЙ', лучше использовать КЛН_ИМЯ = 'Николай', если все имена в столбце КЛН_ИМЯ хранятся с правильной заглавной буквой.
- *Сравнения числовых полей выполняются быстрее, чем сравнения символов, дат и значений NULL.* В условиях поиска сравнение числового атрибута с числовым литералом происходит быстрее, чем сравнение символьного атрибута с символьным литералом. Как правило, процессор обрабатывает числовые сравнения (целые и десятичные) быстрее, чем сравнения символов и дат. Поскольку индексы не хранят ссылки на нулевые значения, условия NULL включают дополнительную обработку и, следовательно, имеют тенденцию быть самым медленным из всех условных операндов.
- *Сравнения равенства обычно быстрее, чем сравнения неравенства.* Например, ТОВ_ЦЕНА = 10.0 обрабатывается быстрее, потому что СУБД может выполнять прямой поиск с использованием индекса в столбце. Если нет точных совпадений, условие оценивается как ложное. Однако, если использовать символ неравенства (>, >=, <=, <), СУБД должна выполнить дополнительную обработку для выполнения запроса, потому что в индексе почти всегда будет больше значений «больше чем» или «меньше чем», чем «равные» значения. За исключением NULL, самый медленный из всех операторов сравнения – LIKE с подстановочными символами, как в ТОВ_НАЗВ LIKE «%Ноутбук%». Кроме того, использование символа «не равно» (<>) приводит к более медленным поискам, особенно, когда разреженность данных высока, то есть, когда существует намного больше разных значений, чем равных значений.
- *По возможности следует преобразовывать условные выражения в литералы.* Например, если условие ТОВ_ОСТ – 10 = 7, изменить его на ТОВ_ОСТ = 17. Кроме того, если есть составное условие, такое как:

  ТОВ_ОСТ < ТОВ_МИН AND ТОВ_МИН = ТОВ_ЗКЗ AND ТОВ_ОСТ = 10

  можно изменить его на условие

ТОВ_ОСТ = 10 AND ТОВ_МИН = ТОВ_ЗКЗ AND ТОВ_ОСТ > 10.

- *При использовании нескольких условных выражений сначала указывать условия равенства.* Нужно обратить внимание, что это было сделано в предыдущем примере. Условия равенства обрабатываются быстрее, чем условия неравенства. Хотя большинство СУРБД автоматически сделают это, уделение внимания этой детали облегчает загрузку оптимизатора запросов. Оптимизатору не придется делать то, что уже сделано.
- *Если используется несколько условий AND, сначала написать условие, которое, скорее всего, будет ложным.* Если использовать эту технику, СУБД прекратит оценку остальных условий, как только найдет условное выражение, которое оценивается как ложное. Для того, чтобы результаты нескольких условий AND были истинными, все условия должны истинными. Если одно из условий оценивается как ложное, весь набор условий будет оцениваться как ложный. Если использовать эту технику, СУБД не будет тратить время на ненужную оценку дополнительных условий. Естественно, использование этого метода подразумевает знание разреженности набора данных. Например, следующий список условий:
  ТОВ_ЦЕНА > 10 AND ПОСТ_ГРД = 'Нижневартовск'.
  Если известно, что в Нижневартовске находится всего несколько поставщиков, можно переписать это условие следующим образом:
  ПОСТ_ГРД = 'Нижневартовск' AND ТОВ_ЦЕНА > 10.
- *При использовании нескольких условий «OR» сначала ставит наиболее вероятное условие.* Таким образом, СУБД прекратит оценку оставшихся условий, как только найдет условное выражение, которое оценивается как истинное. Для нескольких условий OR, чтобы результат был истинным, только одно из условий должно быть истинным.
- *По возможности стараться избегать использования логического оператора NOT.* Лучше всего преобразовать выражение SQL, содержащее логический оператор NOT, в эквивалентное выражение. Например: NOT (ТОВ_ЦЕНА > 10.0) можно записать как ТОВ_ЦЕНА <= 10.0. Кроме того, NOT (СОТР_ПОЛ = 'М') можно записать как СОТР_ПОЛ = 'Ж'.

8.6. Формулировка запроса

Запросы обычно пишутся для ответа на вопросы. Например, если пользователь дает пример выходных данных и предлагает соответствовать этому выходному формату, необходимо написать соответствующий SQL. Чтобы выполнить работу, нужно тщательно оценить, какие столбцы, таблицы и вычисления необходимы для получения желаемого результата. Для этого надо хорошо понимать базу данных.

Чтобы сформулировать запрос, обычно выполняются следующие шаги:

1. *Определение необходимых столбцов и вычислений.* Первый шаг необходим, чтобы четко определить, какие значения данных надо вернуть. Нужно вернуть только имена и адреса, или нужно также включить некоторые вычисления? Все столбцы в операторе SELECT должны возвращать отдельные значения.
   a. Нужны простые выражения? Например, нужно ли умножить цену на имеющееся количество, чтобы получить общую стоимость запасов? Могут понадобиться некоторые функции отдельных атрибутов, такие как DATE(), SYSDATE() или ROUND().
   b. Нужны агрегатные функции? Если нужно рассчитать общий объем продаж по товарам, нужно использовать предложение GROUP BY. В некоторых случаях может понадобиться использовать подзапрос.
   c. Определить гранулярность исходных данных, необходимых для запроса. Иногда может понадобиться обобщить данные, которые не всегда доступны ни в одной таблице. В таких случаях можно разбить запрос на несколько подзапросов и сохранить эти подзапросы в виде представлений. Затем можно создать запрос верхнего уровня, который объединяет эти представления и генерирует окончательный результат.
2. *Определение исходных таблиц.* Как только станет известно, какие столбцы требуются, можно будет определить исходные таблицы, используемые в запросе. Некоторые атрибуты появляются в более чем одной таблице. В этих случаях нужно использовать наименьшее количество таблиц в запросе, чтобы минимизировать количество операций объединения.
3. *Определение способа соединения таблиц.* Как только станет известно, какие таблицы нужны в запросе, надо правильно определить, как соединить таблицы. В большинстве случаев используется внутреннее соединение, но в некоторых случаях может понадобиться внешнее соединение.
4. *Определение необходимых критериев выбора.* В большинстве запросов используются критерии выбора определенного типа. В этом случае нужно определить, какие операнды и операторы необходимы в критериях. Убедиться в правильности типа данных и степени детализации данных в критериях сравнения.
   a. *Простое сравнение*. В большинстве случаев сравниваются отдельные значения, например, ТОВ_ЦЕНА > 10.
   b. *Один атрибут с несколькими значениями*. Если сравнивается один атрибут с несколькими значениями, может потребоваться использовать оператор сравнения IN, например, КЛН_ГРД IN ('Нижневартовск', 'Мегион', 'Лангепас').
   c. *Вложенные сравнения*. Может потребоваться наличие вложенных критериев выбора, включающих подзапросы – например, ТОВ_ЦЕНА >= (SELECT AVG (ТОВ_ЦЕНА) FROM ТОВАР).

d. *Выборка сгруппированных данных.* В других случаях критерии выбора могут применяться не к исходным данным, а к совокупным данным. В этих случаях нужно использовать предложение HAVING.

5. *Определение порядка отображения выходных данных.* Наконец, требуемый вывод может быть упорядочен одним или несколькими столбцами. В этих случаях нужно использовать предложение ORDER BY. Предложение ORDER BY является одной из наиболее ресурсоемких операций для СУБД.

8.7. Настройка производительности СУБД

Настройка производительности СУБД включает задачи, такие как управление процессами СУБД в первичной памяти (выделение памяти для целей кэширования) и управление структурами в физической памяти (выделение пространства для файлов данных).

Настройка производительности СУБД на стороне сервера направлена на настройку параметров, используемых для:

- *Кэш данных*. Размер кэша данных должен быть достаточно большим, чтобы можно было обслуживать как можно больше запросов данных. Каждая СУБД имеет настройки, которые контролируют размер кэша данных. Этот кэш используется всеми пользователями базы данных.
- *Кэш SQL*. В кэше SQL хранятся последние выполненные операторы SQL (после того, как оптимизатор проанализировал операторы SQL). Как правило, если есть приложение с несколькими пользователями, обращающимися к базе данных, один и тот же запрос, вероятно, будет отправлен многими разными пользователями. В этих случаях СУБД будет анализировать запрос только один раз и выполнять его много раз, используя один и тот же план доступа. Таким образом, второй и последующие запросы SQL для того же запроса обслуживаются из кэша SQL, пропуская этап синтаксического анализа.
- *Кэш сортировки*. Кэш сортировки используется как область временного хранения для операций ORDER BY или GROUP BY, а также для функций создания индекса.
- *Режим оптимизатора*. Большинство СУБД работают в одном из двух режимов оптимизации: на основе затрат или на основе правил. Другие автоматически определяют режим оптимизации в зависимости от того, доступна ли статистика базы данных. Например, администратор БД отвечает за генерацию статистики, которая используется оптимизатором на основе затрат. Если статистика недоступна, СУБД использует оптимизатор на основе правил.

С точки зрения производительности было бы оптимальным хранить всю БД в первичной памяти, чтобы минимизировать медленный доступ к диску. Вот почему несколько поставщиков баз данных предлагают варианты *баз данных в оперативной памяти* для своих продуктов. Системы БД в оперативной памяти оптимизированы для хранения больших частей (если не всех) данных в оперативной памяти (RAM), а не во внешней (дисковой) памяти. Эти

системы становятся популярными из-за растущих требований к производительности современных приложений, снижения затрат и технологических достижений компонентов (таких как флэш-память и твердотельные накопители). Даже если эти типы баз данных устраняют узкие места в доступе к диску, они по-прежнему подлежат правилам оптимизации запросов и настройки производительности, особенно когда сталкиваются с плохо разработанными БД или плохо написанными операторами SQL.

Хотя БД в памяти занимают нишу на отдельных рынках, большинство реализаций БД по-прежнему полагается на данные, хранящиеся на дисках. Вот почему управление деталями физического хранения файлов данных играет важную роль в настройке производительности СУБД. Нужно обратить внимание на следующие общие рекомендации для физического хранения баз данных:

- Использовать *ускорители ввода/вывода*. Устройство этого типа использует флэш-накопители (SSD) для хранения базы данных. SSD не имеет движущихся частей и, следовательно, выполняет операции ввода-вывода с более высокой скоростью, чем традиционные вращающиеся дисководы. Ускорители ввода/вывода обеспечивают высокую производительность транзакций и снижают конкуренцию, вызванную типичными накопителями.
- Использовать *RAID* (избыточный массив независимых дисков) для обеспечения повышения производительности, отказоустойчивости, а также баланса между ними. Отказоустойчивость означает, что в случае сбоя данные могут быть восстановлены. Системы RAID используют несколько дисков для создания виртуальных дисков (томов хранения), состоящих из нескольких отдельных дисков. Таблица 29 описывает наиболее распространенные конфигурации RAID.

Таблица 29

Основные RAID уровни

RAID уровень	Описание
0	Блоки данных распределены по отдельным дискам. Также известен как полосатый массив. Обеспечивает повышенную производительность, но не отказоустойчивость. Требуется минимум два диска.
1	Одинаковые блоки данных записываются (дублируются) на отдельные диски. Также называется зеркалированием или дуплексом. Обеспечивает повышенную производительность чтения и отказоустойчивость благодаря избыточности данных. Требуется минимум два диска.
3	Данные распределяются по разным дискам, а данные четности вычисляются и сохраняются на выделенном диске. (Данные четности – специально сгенерированные данные, которые позволяют восстанавливать поврежденные или отсутствующие данные). Обеспечивает хорошую производительность чтения и отказоустойчивость благодаря проверке четности. Требуется минимум три диска.

RAID уровень	Описание
5	Данные и информация четности распределяются между отдельными дисками. Обеспечивает хорошую производительность чтения и отказоустойчивость благодаря четности. Требуется минимум три диска.
1 + 0	Блоки данных распределяются по отдельным дискам и зеркально отражаются (дублируются). Такое расположение обеспечивает как скорость, так и отказоустойчивость. Это рекомендуемая конфигурация RAID для большинства установок баз данных (если стоимость не является проблемой).

- Минимизировать конкуренцию на диске. Использовать несколько независимых томов хранения с независимыми шпинделями (вращающимися дисками), чтобы минимизировать циклы жесткого диска. База данных состоит из множества табличных пространств, каждое из которых имеет определенную функцию. В свою очередь, каждое табличное пространство состоит из нескольких файлов данных, в которых данные фактически хранятся. База данных должна иметь как минимум следующие табличные пространства:
  - *Системное табличное пространство*. Используется для хранения таблиц метаданных. Это наиболее часто используемое табличное пространство, и его следует хранить в своем собственном томе.
  - *Табличное пространство пользовательских данных*. Используется для хранения данных пользователя. Необходимо создать столько табличных пространств пользовательских данных и файлов данных, сколько требуется для обеспечения баланса между производительностью и удобством использования. Например, можно создать и назначить отдельное табличное пространство пользовательских данных для каждого приложения и каждой отдельной группы пользователей, но не обязательно для каждого пользователя.
  - *Индекс табличного пространства*. Используется для хранения индексов. Можно создать и назначить отдельное табличное пространство индекса для каждого приложения и каждой группы пользователей. Файлы данных табличного пространства индекса должны храниться в томе хранилища, отдельном от файлов пользовательских данных или файлов системных данных.
  - *Временное табличное пространство*. Используется как временное хранилище для операций соединения, сортировки или результатов агрегации. Можно создать и назначить другое временное табличное пространство для каждого приложения и каждой группы пользователей.
  - *Откат сегмента табличного пространства*. Используется для восстановления транзакций.

- Поместить таблицы с высокой нагрузкой в свои собственные табличные пространства, чтобы база данных минимизировала конфликт с другими таблицами.
- Назначить отдельные файлы данных в отдельных томах хранения для индексов, системных таблиц и таблиц высокого уровня. Это гарантирует, что индексные операции не будут конфликтовать с данными пользователя или операциями доступа к таблице словаря данных. Еще одним преимуществом этого подхода является то, что можно использовать диски разных размеров в разных томах. Например, объем данных может использовать размер блока 16КБ, в то время как индексный объем может использовать размер блока 8КБ. Размер записи индекса, как правило, меньше, и, изменяя размер блока, уменьшается конфликт и минимизируется операции ввода-вывода. Это очень важно; многие администраторы баз данных игнорируют индексы как источник разногласий. Используя отдельные тома хранения и блоки разных размеров, операции ввода-вывода для данных и индексов будут выполняться асинхронно (в разное время); более важно то, что вероятность операций записи, блокирующих операции чтения, уменьшается, поскольку блокировки страниц имеют тенденцию блокировать меньшее количество записей.
- Воспользоваться преимуществами различных организаций хранения таблиц, доступных в БД. Например, в Oracle использование таблиц, организованных по индексу (IOT); в SQL Server таблицы кластерного индекса. *Упорядоченная по индексу таблица* (или *таблица с кластерным индексом*) представляет собой таблицу, в которой данные пользователя и данные индекса хранятся в последовательных расположениях в постоянном хранилище. Этот тип организации хранения обеспечивает преимущество в производительности для таблиц, к которым обычно обращаются через заданный порядок индексов, поскольку индекс содержит ключ индекса, а также строки данных. Поэтому СУБД выполняет меньше операций ввода-вывода.
- Таблицы разделов на основе использования. Некоторые СУРБД поддерживают горизонтальное разбиение таблиц на основе атрибутов. Таким образом, один запрос SQL может обрабатываться несколькими обработчиками данных. Разместить разделы таблицы ближе к месту их использования.
- При необходимости использовать денормализованные таблицы. Другими словами, можно улучшить производительность, переместив таблицу из более высокой нормальной формы в более низкую нормальную форму – обычно из третьей во вторую нормальную форму. Этот метод добавляет дублирование данных, но минимизирует операции соединения.
- Хранить вычисленные и агрегированные атрибуты в таблицах. Использовать производные атрибуты в таблицах. Например, можно добавить промежуточную сумму чека, сумму налога и итоговую сумму в таблице ЧЕК. Использование

производных атрибутов минимизирует вычисления в запросах и операциях соединения, особенно во время выполнения агрегатных запросов.

8.8. Итоги

- Настройка производительности базы данных относится к набору действий и процедур, предназначенных для обеспечения того, чтобы запрос конечного пользователя обрабатывался СУБД в наименьшее количество времени. Настройка производительности SQL относится к действиям на стороне клиента, которые предназначены для генерации кода SQL, который возвращает правильный ответ за наименьшее количество времени, используя минимальное количество ресурсов на стороне сервера. Настройка производительности СУБД относится к действиям на стороне сервера, которые ориентированы таким образом, чтобы СУБД была должным образом сконфигурирована для быстрого реагирования на запросы клиентов при оптимальном использовании существующих ресурсов.
- Статистика базы данных относится к количеству измерений, собранных СУБД, которые описывают моментальный снимок характеристик объектов базы данных. СУБД собирает статистику об объектах, таких как таблицы, индексы и доступные ресурсы, которые включают число используемых процессоров, скорость процессора и доступное временное пространство. СУБД использует статистику для принятия критических решений по повышению эффективности обработки запросов.
- СУБД обрабатывают запросы в три этапа. На этапе синтаксического анализа СУБД анализирует запрос SQL и выбирает наиболее эффективный план доступа/выполнения. На этапе выполнения СУБД выполняет запрос SQL, используя выбранный план выполнения. На этапе выборки СУБД выбирает данные и отправляет набор результатов обратно клиенту.
- Индексы имеют решающее значение в процессе, который ускоряет доступ к данным. Индексы облегчают поиск, сортировку и использование агрегатных функций и операций объединения. Повышение скорости доступа к данным происходит потому, что индекс представляет собой упорядоченный набор значений, который содержит ключ индекса и указатели. Разреженность данных относится к числу различных значений, которые может иметь столбец. Индексы рекомендуются в столбцах высокой разреженности, используемых в условиях поиска.
- Во время оптимизации запросов СУБД должна выбирать, какие индексы использовать, как выполнять операции объединения, какую таблицу использовать первой и т.д. Каждая СУБД имеет свои алгоритмы определения наиболее эффективного способа доступа к данным. Двумя наиболее распространенными подходами являются оптимизация на основе правил и затрат.

- Оптимизатор на основе правил использует заданные правила и точки для определения наилучшего подхода к выполнению запроса. Оптимизатор на основе затрат использует сложные алгоритмы, основанные на статистике об объектах, к которым осуществляется доступ, для определения наилучшего подхода к выполнению запроса. В этом случае процесс оптимизатора складывает стоимость обработки, затраты на ввод/вывод и затраты ресурсов (ОЗУ и кэш), чтобы определить общую стоимость заданного плана выполнения.
- Настройка производительности SQL связана с написанием запросов, которые эффективно используют статистику. В частности, запросы должны эффективно использовать индексы. Индексы очень полезны, когда необходимо выбрать небольшое подмножество строк из большой таблицы на основе условия.
- Формулировка запроса имеет дело с тем, как преобразовать деловые вопросы в конкретный код SQL для получения требуемых результатов. Чтобы сделать это, нужно тщательно оценить, какие столбцы, таблицы и вычисления необходимы для создания желаемого результата.
- Настройка производительности СУБД включает такие задачи, как управление процессами СУБД в оперативной памяти (выделение памяти для целей кэширования) и управление структурами в физической памяти (выделение пространства для файлов данных).

Лекция 9. Администрирование БД и безопасность

9.1. Введение

Данные – это сырье, из которого производится информация. Поэтому в современной информационной среде данные являются ценным активом, который требует тщательного управления.

Чтобы оценить ценность данных, надо рассмотреть, что хранится в БД предприятия: данные о клиентах, поставщиках, запасах, операциях и так далее. Сколько возможностей будет потеряно в случае потери данных? Какова фактическая стоимость потери данных? Например, бухгалтерская фирма, потерявшая всю свою базу данных, понесет значительные прямые и косвенные расходы. Проблемы фирмы будут усугублены, если потеря данных произошла в течение налогового сезона. Потеря данных ставит любую организацию в сложное положение. Предприятие не может эффективно выполнять повседневные операции, оно может потерять клиентов, которым требуется быстрое и эффективное обслуживание, и возможность привлечь новых клиентов.

Данные являются ценным ресурсом, который можно перевести в информацию. Если информация является точной и своевременной, это может повысить конкурентоспособность и генерировать богатство. В сущности, организация подвергается циклу «данные-информация-решение»; то есть пользователь применяет аналитику к данным для получения информации, которая является основой знаний, используемых при принятии решений. Этот цикл показан на рисунке 30.

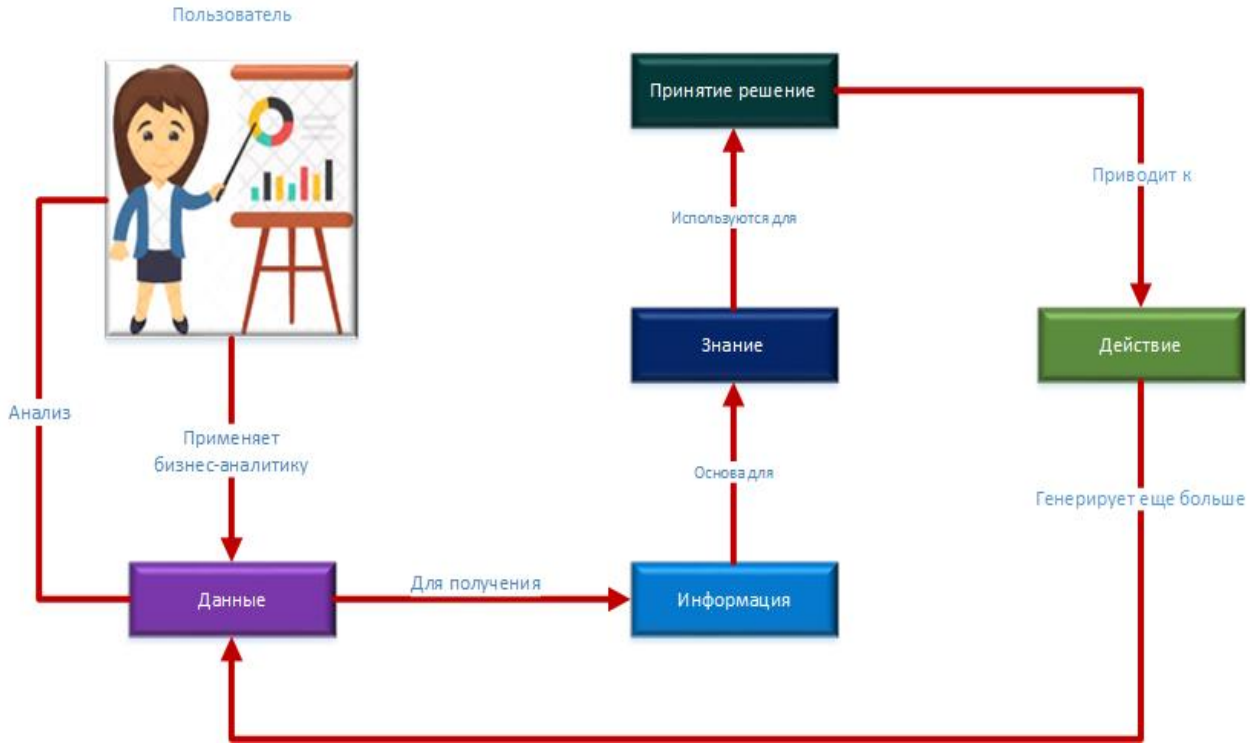


Рис. 30. Цикл данные-информация-принятие решений

Решения, принимаемые менеджерами высокого уровня, инициируют действия на нижних уровнях организации. Такие действия создают дополнительные данные, которые будут использоваться для мониторинга деятельности организации. В свою очередь, дополнительные данные должны быть переработаны в рамках решения «данные-информация-решение». Таким образом, данные образуют основу для принятия решений, стратегического планирования, контроля и мониторинга операций.

Эффективное управление активами имеет решающее значение для успеха организации. Чтобы управлять данными как корпоративным активом, менеджеры должны понимать ценность информации. Для некоторых компаний, таких как кредитные агентства, их единственным продуктом является информация, а их успех является исключительно функцией управления информацией.

Большинство организаций постоянно ищут новые способы использования своих ресурсов данных для получения большей прибыли. Этот рычаг может принимать различные формы: от хранилищ данных, которые поддерживают улучшение отношений с клиентами, до более тесной интеграции с клиентами и поставщиками в поддержку электронной цепочки поставок. Поскольку организации становятся все более зависимыми от информации, точность этой информации становится все более важной. *Грязные данные* или данные, которые страдают от неточностей и несоответствий, становятся еще большей угрозой. Данные могут стать грязными по многим причинам:

- Отсутствие применения ограничений целостности, таких как NOT NULL, уникальность и ссылочная целостность;
- Ошибки ввода данных и опечатки;
- Использование синонимов и омонимов в разных системах;
- Нестандартное использование сокращений в символьных данных;
- Разное разложение составных атрибутов на простые атрибуты в разных системах.

Некоторые причины «грязных» данных, такие как неправильная реализация ограничений, могут быть устранены в отдельной базе данных. Однако устранение других причин является более сложным. Некоторая грязная информация возникает в результате перемещения данных по системам, например, при создании хранилища данных. Усилия по контролю грязных данных обычно называют инициативами по обеспечению качества данных.

Качество данных – комплексный подход к обеспечению точности, достоверности и своевременности данных. Этот комплексный подход важен, потому что качество данных подразумевает не только очистку грязных данных; он также фокусируется на предотвращении будущих неточностей и повышении доверия пользователей к данным. Крупномасштабные инициативы по обеспечению качества данных, как правило, являются сложными и дорогостоящими проектами, поэтому согласование этих инициатив с бизнес-целями является обязательным, равно как и участие высшего руководства. Хотя усилия по обеспечению качества данных сильно различаются в разных организациях, большинство из них включают следующее:

- Структура управления данными, отвечает за качество данных;
- Измерения текущего качества данных;
- Определение стандартов качества данных в соответствии с бизнес-целями;
- Внедрение инструментов и процессов для обеспечения качества данных в будущем.

Ряд инструментов может помочь в реализации инициатив по обеспечению качества данных. В частности, программное обеспечение для профилирования данных и управления основными данными доступно у многих поставщиков. *Программное обеспечение для профилирования данных* собирает статистику, анализирует существующие источники данных и метаданные для определения шаблонов данных и сравнивает шаблоны с установленными в организации стандартами. Этот анализ может помочь оценить качество существующих данных и определить источники грязных данных. *Управление основными данными (MDM)* помогает предотвратить загрязнение данных, координируя общие данные в нескольких системах. Программное обеспечение MDM предоставляет «главную» копию сущностей, таких как клиенты, которые появляются в многочисленных системах по всей организации.

Хотя эти технологические подходы обеспечивают важную часть качества данных, общее решение для высококачественных данных в организации все еще в значительной степени зависит от администрирования и управления данными.

9.2. Необходимость базы данных и ее роль в организации

Данные используются разными людьми в разных отделах по разным причинам, поэтому управление данными должно учитывать концепцию общих данных. Необходимость совместного использования данных делает использование СУБД практически неизбежным. При правильном использовании СУБД облегчает:

- Интерпретацию и представление данных в полезных форматах путем преобразования необработанных данных в информацию;
- Передачу данных и информации нужным людям в нужное время;
- Сохранение данных и использование данных мониторинга в течение приемлемого промежутка времени;
- Контроль за дублированием и использованием данных, как внутри, так и вне организации.

Независимо от организации доминирующая роль БД заключается в поддержке принятия управленческих решений на всех уровнях организации, при сохранении конфиденциальности и безопасности данных.

Управленческая структура организации может быть разделена на три уровня: руководство высшего уровня принимает стратегические решения, руководство среднего звена – тактические решения, а оперативное управление отвечает за ежедневные рабочие решения. Оперативные решения являются краткосрочными: так, менеджер может изменить цену товара, чтобы убрать его из запасов. Тактические решения включают более длительные временные рамки и влияют на более масштабные операции, например, изменение цены продукта в ответ

на конкурентное давление. Стратегические решения влияют на долгосрочное благополучие компании или даже на ее выживание – например, изменение стратегии ценообразования по всем товарным линиям для захвата доли рынка.

СУБД должна предоставлять каждому уровню управления полезное представление о данных и поддерживать необходимый уровень принятия решений. Следующие действия являются типичными для каждого уровня управления.

На высшем уровне управления БД должна быть в состоянии:

- Предоставлять информацию, необходимую для принятия стратегических решений, стратегического планирования, разработки политики и определения целей.
- Обеспечивать доступ к внешним и внутренним данным, чтобы определить возможности роста и наметить его направление. Направление относится к характеру операций: станет ли компания сервисной организацией, производственной организацией или их комбинацией?
- Обеспечивать основу для определения и соблюдения организационных политик, которые переводятся в бизнес-правила на более низких уровнях организации.
- Повышать вероятности положительного возврата инвестиций путем поиска новых способов снижения затрат и повышения производительности труда в компании.
- Обеспечивать обратную связь, чтобы отслеживать достижение компанией своих целей.

На среднем уровне управления БД должна иметь возможность:

- Предоставлять данные, необходимые для тактических решений и планирования.
- Наблюдать и контролировать распределение и использование ресурсов компании и оценивать работу различных отделов.
- Обеспечивать основу для обеспечения безопасности и конфиденциальности данных в БД. *Безопасность* означает защиту данных от случайного или преднамеренного использования посторонними пользователями. В контексте администрирования базы данных *конфиденциальность* – степень, в которой отдельные лица и организации имеют право определять детали использования данных (кто, что, когда, где и как).

На уровне оперативного управления БД должна иметь возможность:

- Представлять и поддерживать деятельность компании как можно ближе. Модель данных должна быть достаточно гибкой, чтобы включать все текущие и будущие данные.
- Производить результаты запроса в пределах указанных уровней производительности. Требования к производительности возрастают для более низких уровней управления и операций. Таким образом, БД должна поддерживать быстрые ответы на большее количество транзакций на уровне оперативного управления.

- Расширять краткосрочные операции компании путем предоставления своевременной информации для поддержки клиентов, а также для разработки приложений и компьютерных операций.

Общей целью любой базы данных является обеспечение бесперебойного потока информации по всей организации.

База данных компании также известна как корпоративная или корпоративная БД. *Корпоративная база данных* может быть определена как представление данных предприятия, которое обеспечивает поддержку всех текущих и ожидаемых будущих операций. Большинство сегодняшних успешных организаций полагаются на базу данных предприятия для обеспечения поддержки всех своих операций – от проектирования до внедрения, от продаж до услуг, и от ежедневного принятия решений до стратегического планирования.

9.3. Внедрение БД

Наличие компьютеризированной системы управления базами данных не гарантирует, что данные будут использованы должным образом для обеспечения лучших решений, требуемых менеджерами. СУБД – инструмент для управления данными; как и любой инструмент, она должна эффективно использоваться для получения желаемых результатов. В руках плотника молоток может помочь в изготовлении мебели, но в руках ребенка он может нанести ущерб. Решением проблем компании является не просто существование компьютерной системы или ее базы данных, а ее эффективное управление и использование.

Внедрение СУБД представляет собой большие изменения и проблемы. На всю организацию СУБД может оказать глубокое влияние, которое может быть положительным или отрицательным в зависимости от администрирования. Например, одним из ключевых соображений является адаптация СУБД к организации, а не принуждение организации к адаптации к СУБД. Основной проблемой должны быть потребности организации, а не технические возможности СУБД. Тем не менее, внедрение СУБД (внутреннее размещение или аутсорсинг в облачной службе) не может быть выполнено без влияния на организацию. Поток новой информации оказывает глубокое влияние на функционирование организации и, следовательно, на ее корпоративную культуру.

Внедрение СУБД было описано как процесс, который включает в себя три важных аспекта:

- *Технологический* – программное и аппаратное обеспечение СУБД;
- *Управленческий* – административные функции;
- *Культурно-корпоративное* сопротивление изменениям.

Технологический аспект включает в себя выбор, установку, настройку и мониторинг СУБД, чтобы обеспечить ее эффективное управление хранением данных, доступом и безопасностью. Персонал, отвечающий за установку СУБД, должен обладать техническими навыками для обеспечения адекватной поддержки разных пользователей системы: программистов, менеджеров и операторов. Таким образом, кадровое обеспечение

администрации БД является ключевым технологическим фактором. Выбранный персонал должен обладать правильным сочетанием технических и управленческих навыков, чтобы обеспечить плавный переход к новой среде. В современном мире ИТ технологические аспекты будут применяться как к внутренним СУБД, так и к облачным средам данных.

Управленческий аспект внедрения СУБД не следует воспринимать легкомысленно. Высококачественная СУБД не гарантирует качественную информационную систему, как и наличие лучшего гоночного автомобиля не гарантирует победу в гонке. Такие управленческие аспекты также включают управление услугами и отношения с поставщиком облачных услуг передачи данных.

Внедрение СУБД требует тщательного планирования для создания соответствующей организационной структуры и размещения персонала, отвечающего за администрирование системы. Эта структура также должна быть предметом тщательно разработанного мониторинга и контроля. Административный персонал должен обладать отличными навыками межличностного общения в сочетании с широким пониманием организационных и деловых вопросов. Высшее руководство должно быть привержено новой системе, должно определять и поддерживать функции администрирования данных, цели и роли в организации.

Культурное влияние новой системы баз данных должно оцениваться тщательно. СУБД может влиять на людей, функции и взаимодействия. Например, дополнительный персонал может быть нанят, новые роли могут быть распределены существующему персоналу, а эффективность работы сотрудников может оцениваться с использованием новых стандартов.

Культурное воздействие вероятно, потому что подход базы данных создает более контролируемый и структурированный поток информации. Руководители отделов, которые привыкли обращаться с собственными данными, должны отказаться от права собственности и поделиться своими данными с остальной частью компании. Прикладные программисты должны учиться и следовать новым стандартам проектирования и разработки. Менеджеры могут воспринимать информационную перегрузку и требовать времени для адаптации к новой среде.

Когда новая БД появится в сети, люди могут не захотеть использовать ее информацию и могут поставить под сомнение ее ценность или точность. Многие могут быть разочарованы тем, что информация не соответствует их предвзятым представлениям и убеждениям. Администраторы БД должны быть готовы открыть свои двери для пользователей, выслушать их проблемы, по возможности принять меры по их решению и объяснить использование и преимущества системы.

9.4. Эволюция управления базами данных

Администрирование данных уходит своими корнями в старый, децентрализованный мир файловой системы. Стоимость данных и дублирование данных в этих системах привело к централизованному администрированию данных, известному как отдел электронной обработки данных. Задача данного отдела заключалась в объединении всех компьютерных ресурсов для поддержки всех департаментов на оперативном уровне. Администраторы

получили полномочия управлять всеми файловыми системами компании, а также разрешать конфликты данных и управления, возникающие в результате дублирования и неправильного использования данных.

Появление СУБД и ее общего представления о данных породило новый уровень сложности управления данными и привело к созданию *отдела информационных систем (ИС)*. Обязанности отдела ИС были расширены и теперь включают следующие:

- *Сервисная функция* для предоставления пользователям поддержки управления данными;
- *Производственная функция* для предоставления пользователям решений для их информационных потребностей через интегрированные прикладные или управленческие информационные системы.

Функция отдела ИС отразилась на его внутренней организационной структуре. Современная организационная структура отдела ИС в средних и крупных компаниях показана на рисунке 31.

Рис. 31. Организационная структура отдела ИС

По мере роста спроса, сегмент разработки приложений ИС был подразделен по типу систем, которые он поддерживал: учет, инвентаризация, маркетинг, хранение данных, бизнес-аналитика и т.д. Однако это означает, что обязанности администратора БД были разделены. Сегмент разработки приложений отвечал за сбор требований к базе данных и за разработку логической базы данных, тогда как сегмент операций с базами данных включал в себя внедрение, мониторинг и управление операциями СУБД.

По мере роста числа приложений баз данных управление данными становилось все более сложным, что приводило к развитию администрирования БД. Человек, ответственный за управление централизованной и общей базой данных, стал *администратором базы данных (АБД – DBA)*.

Размер и роль функции АБД варьируются от компании к компании, равно как и ее размещение в организационной структуре. На организационной схеме АБД может быть определен как консультант или руководитель. В должности консультанта АБД часто берет на себя консультативную роль; АБД может разрабатывать стратегию администрирования данных, но не имеет полномочий для ее применения или разрешения возможных конфликтов. В позиции руководителя АБД несет как ответственность, так и полномочия по планированию, определению, реализации и применению политик, стандартов, и процедур, используемых при администрировании данных. Два возможных положения АБД показаны на рисунке 32.

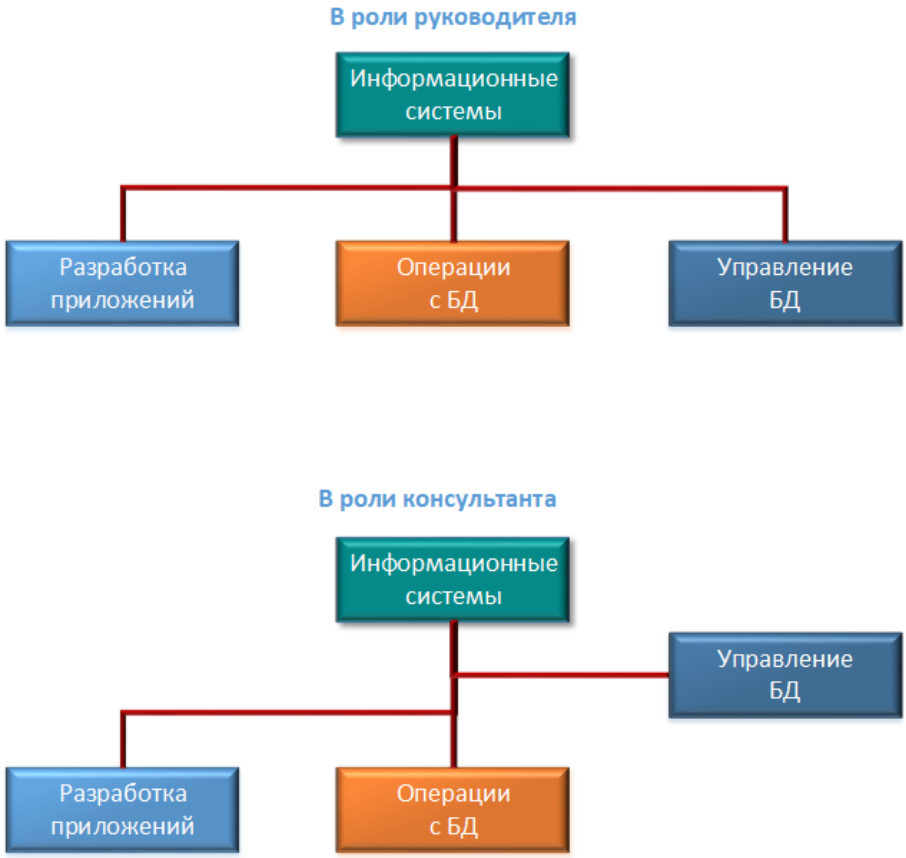


Рис. 32. Функции администратора базы данных

Не существует стандарта для определения того, как АБД должен вписываться в структуру организации, отчасти потому, что сама роль, вероятно, является самой динамичной из всех в организации. На самом деле, быстрые изменения в технологии СУБД диктуют необходимость изменения организационных стилей. Например:

- Развитие распределенных БД может вынудить организацию к дальнейшей децентрализации управления данными. Распределенная БД требует, чтобы системный АБД определял и делегировал обязанности каждого локального АБД, тем самым налагая новые и более сложные координирующие действия на него.
- Растущее использование данных, доступных через Интернет, и растущее число приложений хранилищ данных, вероятно, расширят деятельность АБД по моделированию и проектированию данных.
- Растущая изощренность и мощь пакетов СУБД на основе персональных компьютеров обеспечивают простую платформу для разработки удобных, экономичных и эффективных решений. Однако такая среда также допускает дублирование данных, не говоря уже о проблемах, создаваемых людьми, не имеющими технической квалификации для создания хороших проектов БД. Новая вычислительная среда требует от АБД нового набора технических и управленческих навыков.

- Растущее использование облачных услуг передачи данных толкает многие платформы БД и инфраструктуры в облако. Это может освободить АБД от многих низкоуровневых технологических задач, позволяя им сосредоточиться на более важных стратегических вопросах. В таких средах АБД становится поставщиком услуг по использованию данных и консультантом для организации.
- И наоборот, растущее использование больших данных в организациях может заставить АБД стать более технологически ориентированным. Продолжающиеся усилия по интеграции систем хранения Hadoop как с NoSQL, так и с реляционными базами данных требуют, чтобы АБД был знаком с проблемами хранения и доступа низкого уровня, которые все еще доминируют в этих новых дисциплинах.

Действия АБД обычно определяются и делятся в соответствии с фазами жизненного цикла базы данных (ЖЦ БД). Если используется такой подход, функция АБД требует, чтобы персонал выполнял следующие действия:

- Планирование БД, включая определение стандартов, процедур и правоприменения;
- Сбор требований к БД и создание концептуальной модели;
- Создание логической и транзакционной модели БД;
- Создание физической модели БД и ее реализации;
- Тестирование и отладка БД;
- Операции с БД, ее обслуживание, включая установку, конвертацию и миграцию данных;
- Обучение, поддерживание БД;
- Мониторинг и управление качеством данных.

В организации может быть установлено несколько несовместимых СУБД для поддержки разных операций. Например, некоторые корпорации имеют иерархическую СУБД для поддержки ежедневных транзакций на операционном уровне и реляционную для поддержки специальных информационных потребностей среднего и высшего руководства. Разные СУБД для персональных компьютеров могут быть установлены в разных отделах. В такой среде для каждой СУБД компании может быть назначен один АБД. Генерального координатора всех АБД иногда называют *системным администратором*.

Растет тенденция к специализации в управлении данными. Например, организационные диаграммы, используемые некоторыми крупными корпорациями, проводят различие между АБД и *администратором данных (АД)*. АД, также известный как *менеджер информационных ресурсов (МИР)*, обычно подчиняется непосредственно высшему руководству и получает более высокую степень ответственности и полномочий, чем АБД, хотя эти две роли могут перекрываться.

АД отвечает за управление общими ресурсами корпоративных данных, как компьютеризированными, так и ручными. Таким образом, работа АД охватывает больше операций, чем АБД, поскольку АД контролирует данные, выходящие за рамки СУБД, в дополнение к компьютеризированным данным. В зависимости от структуры организации АБД

может отчитываться перед АД, МИР, менеджером ИС или непосредственно перед генеральным директором компании.

9.5. Человеческий фактор среды БД

Эффективное администрирование данных требует как технических, так и управленческих навыков. Например, работа АД обычно имеет сильную управленческую ориентацию с охватом всей компании, а также техническую ориентацию, которая имеет более узкую, специфичную для СУБД область. Тем не менее, АБД также должен иметь значительные навыки управления людьми. Например, как АД, так и АБД направляют и контролируют кадры, а также обучение персонала в соответствующих департаментах.

АД обеспечивает глобальную и всеобъемлющую административную стратегию для данных организации. Он отвечает за консолидацию и согласованность как ручных, так и компьютеризированных данных.

АД должен установить цели администрирования данных. Эти цели определяются такими вопросами, как:

- Доступность данных;
- Согласованность и целостность данных;
- Безопасность и конфиденциальность данных;
- Стандарты качества данных;
- Степень и тип использования данных.

Естественно, список можно расширить, чтобы соответствовать конкретным потребностям организации в данных. Независимо от того, как осуществляется управление данными – и несмотря на тот факт, что у АД или АБД большие полномочия для определения и контроля способа использования данных компании – они не владеют данными. Вместо этого их функции определены, чтобы подчеркнуть, что данные являются общим активом компании.

Не существует общепринятых административных стандартов АД и АБД. Стиль, обязанности, организационное размещение и внутренняя структура обеих функций варьируются от компании к компании. Например, многие компании распределяют обязанности АД между АБД и менеджером ИС.

Арбитраж взаимодействий между двумя наиболее важными активами любой организации – людьми и данными, помещает АБД в динамическую среду, изображенную на рисунке 33.

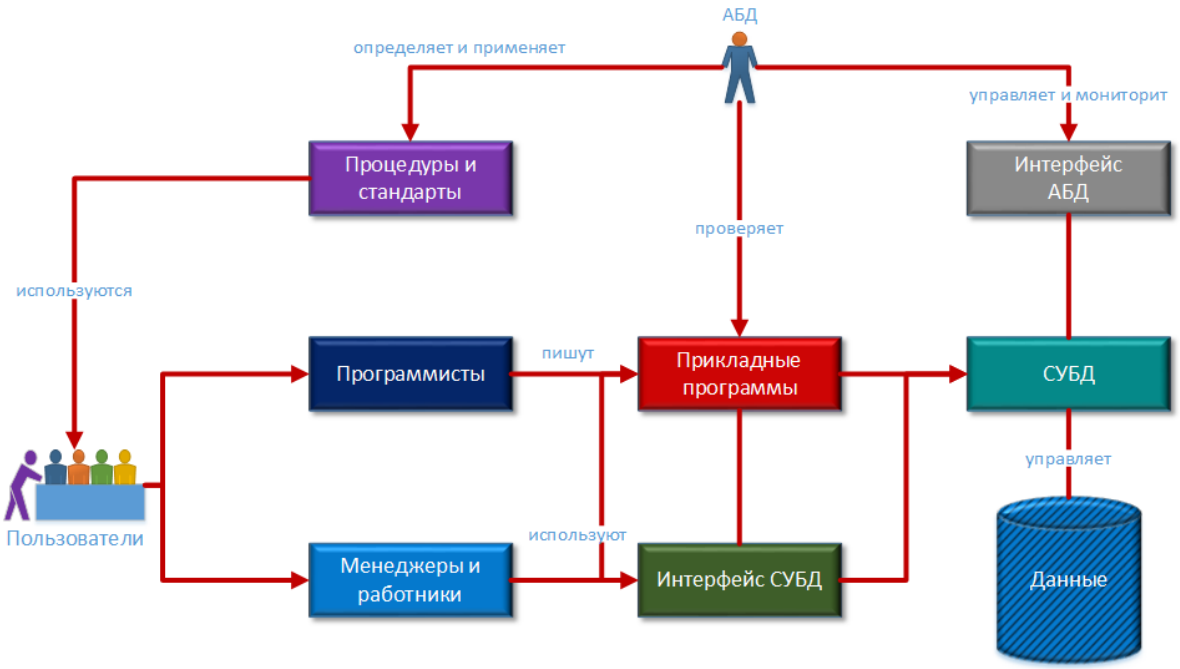


Рис. 33. Роль АБД в организации

Изучая рисунок 33, необходимо обратить внимание, что администратор баз данных является центром взаимодействия данных и пользователей. АБД определяет и обеспечивает соблюдение процедур и стандартов, которые должны использоваться программистами и пользователями при работе с СУБД. АБД также проверяет, что доступ программиста и пользователя соответствует требуемым стандартам качества и безопасности.

Пользователи БД могут быть классифицированы по следующим критериям:

- Тип необходимой поддержки принятия решений (операционный, тактический или стратегический);
- Степень компьютерных знаний (новичок, опытный или эксперт);
- Частота доступа (случайный, периодический или частый).

Эти классификации не являются исключительными и обычно совпадают. Например, оперативный пользователь может быть экспертом со случайным доступом к БД, или менеджер верхнего уровня может быть стратегическим начинающим пользователем с периодическим доступом к БД. С другой стороны, прикладной программист БД является экспертом по эксплуатации. Таким образом, в каждой организации работают люди, чей уровень знаний в области БД охватывает весь спектр. АБД должен уметь взаимодействовать со всеми, понимать их разные потребности, отвечать на вопросы на всех уровнях знаний и эффективно общаться с этими людьми.

Действия АБД, изображенные на рисунке 33, указывают на необходимость разнообразного сочетания навыков. В крупных компаниях такие навыки, вероятно, будут распределены между несколькими АБД, а в небольших компаниях навыки могут быть предметом только одного АБД. Навыки можно разделить на две категории – управленческие и технические.

АБД должен выполнять две разные роли. Управляющая роль его сосредоточена на управлении персоналом и взаимодействии с пользователями. Техническая роль заключается в использовании СУБД – проектировании, разработке и внедрении баз данных, а также в производстве, разработке и использовании прикладных программ.

Управляющая роль АБД. Как менеджер, АБД должен сосредоточиться на контроле и планировании администрирования БД. Поэтому администратор базы данных отвечает за следующее:

- Координация, мониторинг и распределение ресурсов администрирования БД: люди и данные.
- Определение целей и формулирование стратегических планов для администрирования БД.

АБД отвечает за планирование, организацию, тестирование, мониторинг и предоставление довольно большого количества услуг.

Поддержка пользователей. АБД взаимодействует с пользователями, предоставляя данные и информационную поддержку. Поскольку пользователи обычно имеют разный компьютерный опыт, служба поддержки включает в себя следующее:

- *Сбор требований пользователя.* АБД должен работать с пользователями, чтобы помочь собрать данные, необходимые для идентификации и описания их настоящих и будущих информационных потребностей. Коммуникационные навыки АБД важны в тесном сотрудничестве с людьми, которые имеют различный компьютерный фон и стиль общения.
- *Укрепление доверия пользователя*. Поиск адекватных решений проблем пользователей повышает их доверие к АБД. Он также должен информировать пользователей о предоставляемых услугах и о том, как они улучшают управление данными и их безопасность.
- *Разрешение конфликтов и проблем*. Поиск решений проблем пользователей в одном отделе может вызвать конфликты с другими отделами. Пользователи обычно озабочены своими собственными потребностями в данных, а не потребностями других, и они могут не учитывать, как их данные могут повлиять на другие отделы внутри организации. Когда возникают конфликты, АБД должен иметь полномочия и ответственность за их разрешение.
- *Поиск решений для информационных потребностей*. Способность и полномочия разрешать конфликты данных позволяет АБД разрабатывать решения, которые будут должным образом соответствовать структуре управления данными и удовлетворять информационные потребности пользователей. Учитывая растущую важность Интернета, эти решения, вероятно, потребуют разработки и управления веб-серверами для взаимодействия с БД. На самом деле, взрывной рост электронной коммерции требует использования динамических интерфейсов для упрощения интерактивных запросов и продаж продуктов.

- *Обеспечение качества и целостности данных и приложений.* Как только правильное решение найдено, оно должно быть правильно реализовано и использовано. АБД должен работать с прикладными программистами и пользователями, чтобы обучить их стандартам и процедурам БД, необходимым для качества данных, доступа к ним и манипулирования данными. АБД также должен убедиться, что транзакции БД не влияют отрицательно на качество данных. Аналогичным образом, сертификация качества прикладных программ, обращающихся к базе данных, является важной функцией АБД. Особое внимание следует уделить интернет-интерфейсам СУБД, поскольку они подвержены проблемам с безопасностью, особенно при использовании облачных служб данных.
- *Управление обучением и поддержкой пользователей СУБД*. Одним из наиболее трудоемких действий АБД является обучение конечных пользователей работе с базой данных. АБД должен убедиться, что все пользователи понимают основные функции программного обеспечения СУБД. АБД координирует и контролирует все учебные мероприятия по СУБД.

Политики, процедуры и стандарты. Успешная стратегия администрирования данных требует постоянного применения политик, процедур и стандартов для правильного создания, использования и распространения данных в базе данных. АБД должен определить, задокументировать и сообщить следующее, прежде чем их можно будет применять:

- *Политики* – общие положения о направлении или действиях, которые сообщают и поддерживают цели АБД.
- *Стандарты* описывают минимальные требования для конкретной деятельности АБД; они более подробны и конкретны, чем политики. По сути, стандарты – это правила, которые оценивают качество деятельности. Например, стандарты определяют структуру прикладных программ и соглашения об именах, которые должны использовать программисты.
- *Процедуры* – инструкции, описывающие последовательность шагов, которые необходимо выполнить во время выполнения определенного действия. Процедуры должны разрабатываться в существующих условиях труда, и они должны поддерживать и улучшать рабочую среду.

Следующие примеры проиллюстрируют различия между политикой, стандартами и процедурами:

Политика

- Все пользователи должны иметь пароли.
- Пароли должны меняться каждые шесть месяцев.

Стандарты

- Пароль должен содержать не менее 5 символов.
- Пароль должен содержать не более 12 символов.

- Номера паспортов, имена и даты рождения не могут использоваться в качестве паролей.

Процедуры

Чтобы создать пароль:

1) пользователь отправляет АБД письменный запрос на создание учетной записи;
2) АБД утверждает запрос и направляет его оператору;
3) оператор создает учетную запись, назначает временный пароль и отправляет информацию об учетной записи пользователю;
4) копия информации об учетной записи отправляется АБД;
5) пользователь меняет временный пароль на постоянный.

Стандарты и процедуры, определенные АБД, применяются ко всем пользователям, которые хотят использовать БД. Стандарты и процедуры должны дополнять друг друга и представлять собой расширение политик администрирования данных. Процедуры должны облегчать работу пользователей и АБД. АБД должен определить, сообщить и обеспечить выполнение процедур, которые охватывают такие области, как:

- *Сбор требований к БД пользователя*. Какая документация требуется? Какие формы должны быть использованы?
- *Проектирование БД и моделирование*. Какая методология проектирования БД будет использоваться (реляционная или объектно-ориентированная)? Какие инструменты будут использоваться (инструменты CASE, словари данных, диаграммы UML или ER)?
- *Документация и соглашения об именах*. Какая документация должна использоваться при определении всех элементов данных, наборов и программ, которые обращаются к базе данных?
- *Разработка, кодирование и тестирование прикладных программ баз данных*. АБД должен определить стандарты кодирования прикладной программы, документации и тестирования. Стандарты и процедуры предоставляются программистам приложений, и АБД должен обеспечивать соблюдение этих стандартов.
- *Выбор программного обеспечения БД*. Выбранная СУБД должна должным образом взаимодействовать с существующим ПО, иметь функции, необходимые организации, и обеспечивать положительный возврат инвестиций. В современной интернет-среде АБД также должен работать с веб- и сетевыми администраторами для обеспечения эффективного и безопасного подключения к веб- и облачным БД.
- *Безопасность и целостность БД*. АБД должен определить политику, управляющую безопасностью и целостностью. Безопасность базы данных особенно важна. Стандарты безопасности должны быть четко определены и строго соблюдаться. Процедуры безопасности должны обрабатывать множество сценариев, чтобы обеспечить минимизацию проблем. Хотя ни одна система не может быть полностью безопасной, процедуры должны соответствовать критическим стандартам.

Растущее использование интернет-интерфейсов для БД открывает двери для новых угроз безопасности, которые намного сложнее в управлении, чем в традиционных интерфейсах – это особенно важно при работе с облачными службами данных. Поэтому АБД должен тесно сотрудничать со специалистами по безопасности в Интернете, чтобы обеспечить надлежащую защиту БД от атак.

- *Резервное копирование и восстановление БД*. Процедуры резервного копирования и восстановления БД должны включать информацию, гарантирующую правильное выполнение и управление резервными копиями. АБД должен тесно сотрудничать с любым поставщиком облачных услуг передачи данных, чтобы обеспечить наличие надлежащих процедур для управления резервным копированием и восстановлением данных, а также для обеспечения владения и безопасности данных.
- *Обслуживание и эксплуатация БД*. Ежедневные операции СУБД должны быть четко документированы. Операторы должны вести журналы заданий и должны писать инструкции и примечания для операторов. Такие заметки помогают точно определить причины и решения проблем. Операционные процедуры также должны включать точные инструкции для процедур резервного копирования и восстановления.
- *Обучение пользователей*. В организации должна быть установлена полнофункциональная программа обучения, которая должна быть четко определена. Каждый пользователь должен знать о доступности обучения.

Процедуры и стандарты должны пересматриваться не реже одного раза в год, чтобы поддерживать их в актуальном состоянии и гарантировать, что организация сможет быстро адаптироваться к изменениям в рабочей среде. Естественно, внедрение нового программного обеспечения СУБД, обнаружение нарушений безопасности или целостности, реорганизация компаний и подобные изменения требуют пересмотра процедур и стандартов.

Безопасность данных, конфиденциальность и целостность. Безопасность данных, конфиденциальность и целостность имеют большое значение для АБД, которые управляют установками СУБД. Технология указала путь к повышению производительности за счет управления информацией и позволила распределять данные по нескольким сайтам, что затрудняет поддержание контроля, безопасности и целостности данных. Таким образом, АБД должен использовать механизмы безопасности и целостности, предоставляемые СУБД, для обеспечения соблюдения политик администрирования БД. Кроме того, АБД должны объединиться с экспертами по безопасности в Интернете для создания механизмов безопасности, которые защищают данные от возможных атак или несанкционированного доступа.

Резервное копирование и восстановление данных. Когда данные недоступны, компании несут потенциально разрушительные потери. Поэтому процедуры резервного копирования и восстановления данных имеют решающее значение во всех установках БД. АБД также должен обеспечить полное восстановление данных в случае потери данных или потери целостности БД. Эти потери могут быть частичными или общими; поэтому процедуры резервного

копирования и восстановления являются самой дешевой страховкой БД, которую можно купить.

Управление безопасностью, целостностью, резервным копированием и восстановлением БД настолько важно, что многие отделы АБД создали должность, называемую *сотрудником безопасности баз данных (СББД).* Единственной задачей СББД является обеспечение безопасности и целостности базы данных. В крупных организациях деятельность СББД часто классифицируется как борьба с аварийными ситуациями.

Управление аварийными ситуациями включает все действия АБД, предназначенные для обеспечения доступности данных после физического сбоя или нарушения целостности базы данных. Управление бедствиями включает в себя все планирование, организацию и тестирование планов действий в чрезвычайных ситуациях и процедур восстановления базы данных. Меры резервного копирования и восстановления должны включать как минимум следующее:

- *Периодические резервные копии данных и приложений*. Некоторые СУБД включают инструменты для автоматического резервного копирования и восстановления БД. Такие продукты, как IBM DB2, допускают разные типы резервного копирования: полное, инкрементное и параллельное. *Полная резервная копия*, также известная как *дамп базы данных*, создает полную копию всей базы данных. *Инкрементное резервное копирование* создает резервную копию всех данных с момента последнего резервного копирования. *Параллельное резервное копирование* происходит, пока пользователь работает с БД.
- *Правильная идентификация резервной копии*. Резервные копии должны быть четко идентифицированы посредством подробных описаний и информации о дате, что позволяет АБД гарантировать, что правильные резервные копии используются для восстановления базы данных. Самым распространенным носителем резервного копирования традиционно была магнитная лента; операторы компьютеров старательно хранили и маркировали ленты, а АБД отслеживали текущее местоположение ленты. Однако организации, достаточно большие для найма АБД, обычно не используют ленты для резервного копирования предприятия. Другое решение – оптические и дисковые устройства резервного копирования, а также онлайн-хранилище на основе сетевого хранилища, сетей хранения данных и облачного хранилища данных. Для резервного копирования предприятия используют многоуровневый подход, при котором данные сначала резервируются на быстрые дисковые носители для промежуточного хранения и быстрого восстановления, позже передаются на ленту для архивного хранения.
- *Удобное и безопасное резервное хранилище*. Требуется несколько резервных копий одних и тех же данных, которые должны храниться в разных местах, например, сайтах организации и за ее пределами. (Хранение разных резервных копий в одном и том же месте отрицательно сказывается на необходимости создания нескольких резервных копий). Места хранения должны быть надлежащим образом

подготовлены, они могут включать в себя пожаробезопасные и сейсмостойкие хранилища, а также средства контроля влажности и температуры. АБД должен установить политику для ответа на два вопроса: (1) Где хранятся резервные копии? (2) Как долго должны храниться резервные копии?

- *Физическая защита как аппаратного, так и программного обеспечения*. Защита может включать использование закрытых установок с ограниченным доступом, а также подготовку компьютерных площадок для обеспечения кондиционирования воздуха, резервного питания и противопожарной защиты. Физическая защита также включает в себя резервный компьютер и СУБД, которые будут использоваться в случае чрезвычайной ситуации.
- *Персональный контроль доступа к программному обеспечению установки базы данных*. Многоуровневые пароли и привилегии, а также аппаратные и программные токены запроса/ответа могут использоваться для идентификации авторизованных пользователей ресурсов.
- *Страховое покрытие данных в базе данных*. АБД или сотрудник службы безопасности должен приобрести страховой полис для обеспечения финансовой защиты в случае сбоя базы данных. Страхование может быть дорогим, но оно дешевле, чем катастрофа, вызванная массовой потерей данных.

Стоит сделать два дополнительных замечания:

- Планы восстановления данных и действия в чрезвычайных ситуациях должны быть тщательно протестированы и оценены, и их необходимо часто применять на практике. Не следует занижать отработку пожарных учений, они требуют поддержки и обеспечения проведения со стороны высшего руководства.
- Программа резервного копирования и восстановления вряд ли будет охватывать все компоненты информационной системы. Поэтому целесообразно установить приоритеты для характера и степени восстановления данных.

Распределение и использование данных. Данные полезны только тогда, когда они своевременно доходят до нужных пользователей. АБД отвечает за обеспечение того, чтобы данные дошли до нужных людей в нужное время и в нужном формате. Эти задачи могут занять очень много времени, особенно когда емкость доставки данных основана на типичной среде разработки приложений, где пользователи зависят от программистов, предоставляющих программы для работы с БД. Хотя Интернет, его внутренние и внешние расширения открыли базы данных для корпоративных пользователей, они также создали новый набор проблем для АБД.

Современная философия распространения данных позволяет авторизованным пользователям легко получить доступ к БД. Одним из способов решения этой задачи является облегчение использования новых, более сложных инструментов запросов и новых веб-интерфейсов. Они позволяют АБД обучать пользователей производить необходимую информацию, не завися от программистов. Естественно, АБД должен гарантировать, что пользователи придерживаются соответствующих стандартов и процедур.

Эта философия совместного использования данных сегодня распространена, и, вероятно, она станет более распространенной по мере развития технологий БД. Такая среда является более гибкой для пользователей; становясь более самостоятельными в получении и использовании данных, они могут принимать более правильные решения. Тем не менее, эта «демократия данных» может также вызвать некоторые неприятные побочные эффекты. Предоставление пользователям возможности управлять своими подмножествами данных может непреднамеренно разорвать связь между этими пользователями и администраторами данных. Работа АБД может стать более сложной, а эффективность администрирования данных – поставлена под угрозу. Дублирование данных может снова процветать без проверок на организационном уровне, чтобы обеспечить уникальность элементов данных. Таким образом, пользователи, которые не полностью понимают природу и источники данных, могут неправильно использовать элементы данных.

Техническая роль АБД. Техническая роль АБД требует широкого понимания функций СУБД, конфигурации, языков программирования, а также методологий моделирования и проектирования данных. Например, техническая деятельность АБД включает выбор, установку, эксплуатацию, обслуживание и обновление СУБД и служебного программного обеспечения, а также проектирование, разработку, внедрение и обслуживание прикладных программ, взаимодействующих с базой данных.

Многие из технических действий АБД являются логическим продолжением управленческой деятельности АБД. Например, он занимается безопасностью и целостностью БД, резервным копированием и восстановлением, а также обучением и поддержкой. Технические аспекты работы администратора базируются на следующих областях деятельности:

- Оценка, выбор и установка СУБД и связанных утилит.
- Разработка и внедрение баз данных и приложений.
- Тестирование и оценка баз данных и приложений.
- Работа с СУБД, утилитами и приложениями.
- Обучение и поддержка пользователей.
- Поддержка СУБД, утилит и приложений.

В следующих разделах рассматриваются детали каждой области.

Оценка, выбор и установка СУБД и утилит. Одной из первых и наиболее важных технических обязанностей АБД является выбор СУБД, служебного программного обеспечения и вспомогательного оборудования, которое будет использоваться в организации. Выбор СУБД может также включать рассмотрение облачных услуг передачи данных. Эта задача требует тщательного планирования, которое должно основываться на потребностях организации, а не на конкретных функциях программного и аппаратного обеспечения. АБД должен признать, что целью является решение проблем, а не покупка компьютера или программного обеспечения СУБД. СУБД – это важный инструмент управления, а не технологическая игрушка.

Первый и самый важный шаг плана – определить потребности компании. АБД должен убедиться, что в этом процессе участвуют все пользователи, включая менеджеров высшего и среднего звена. Как только потребности определены, цели администрирования данных могут быть четко установлены, тогда функции СУБД и критерии выбора могут быть определены.

Чтобы согласовать возможности СУБД с потребностями организации, администратору базы данных было бы целесообразно разработать контрольный список требуемых функций СУБД, который решает по меньшей мере следующие проблемы:

- *Модель СУБД*. Удовлетворяют ли потребностям компании реляционная, объектно-ориентированная, объектно-реляционная СУБД или СУБД NoSQL? Если требуется приложение хранилища данных, следует использовать реляционную или многомерную СУБД? СУБД поддерживает схемы типа «звезда»? Чтобы определить, какая модель лучше, нужно определить основную цель приложения: высокая доступность, высокая производительность, точность транзакций (применение ACID) или возможность управлять разными типами данных и сложными отношениями.
- *Объем хранилища СУБД*. Какой максимальный размер диска и базы данных требуется? Какое минимальное количество независимых дисков требуется для «рекомендуемой» установки? Каковы другие потребности хранения? При использовании услуги облачного хранилища, помимо начального размера данных, особое внимание следует уделить ожидаемым темпам роста данных из-за сокращенных дополнительных затрат на хранение данных. Облачное хранилище предоставляет такие проблемы, как местоположение, безопасность, репликация, избыточность и синхронизация данных.
- *Поддержка разработки приложений*. Какие языки программирования поддерживаются? Какие инструменты разработки приложений доступны? (Варианты включают схемы базы данных, словарь данных, мониторинг производительности и средства рисования экрана и меню). Предоставляются ли инструменты запросов пользователю? Предоставляет ли СУБД доступ к веб-интерфейсу?
- *Безопасность и целостность*. Поддерживает ли СУБД правила ссылочной целостности и целостности сущностей, права доступа и т.д.? Поддерживает ли СУБД использование контрольных журналов для выявления ошибок и нарушений безопасности? Можно ли изменить размер контрольного журнала? Если данные хранятся в общедоступном облаке, насколько они безопасны?
- *Резервное копирование и восстановление*. Предоставляет ли СУБД инструменты автоматического резервного копирования и восстановления? Поддерживает ли СУБД резервное копирование на магнитную ленту, оптический диск или в сеть? Создает ли СУБД автоматическое резервное копирование журналов транзакций?

- *Контроль параллелизма*. Поддерживает ли СУБД многопользовательский режим? Какие уровни изоляции (таблица, страница, строка) предлагает СУБД? Сколько ручного кодирования необходимо в прикладных программах?
- *Производительность*. Сколько транзакций в секунду поддерживает СУБД? Нужны ли дополнительные процессоры транзакций? Требуется ли база данных в памяти для обеспечения максимальной производительности?
- *Инструменты администрирования базы данных*. Предлагает ли СУБД какой-либо интерфейс управления АБД? Какой тип информации предоставляет интерфейс АБД? Предоставляет ли СУБД оповещения АБД при возникновении ошибок или нарушений безопасности?
- *Совместимость и распространение данных*. Может ли СУБД работать с другими типами СУБД в той же среде? Какой уровень сосуществования или взаимодействия достигнут? Поддерживает ли СУБД операции чтения и записи в другие пакеты СУБД? Поддерживает ли СУБД архитектуру «клиент/сервер»? Будет ли облачная служба данных лучшим выбором для данной системы?
- *Переносимость и стандарты*. Может ли СУБД работать на разных операционных системах и платформах? Может ли СУБД работать на мэйнфреймах, компьютерах среднего уровня и персональных компьютерах? Могут ли приложения СУБД работать без изменений на всех платформах? Каким отраслевым стандартам соответствует СУБД?
- *Аппаратное обеспечение*. Какое оборудование требуется СУБД? Может ли СУБД работать на виртуальной машине? Требует ли реализация СУБД использования аппаратных кластеров или распределенной среды?
- *Словарь данных*. Есть ли в СУБД «доступный» словарь данных? Взаимодействует ли СУБД с каким-либо инструментом словаря данных? Поддерживает ли СУБД какие-либо открытые инструменты управления?
- *Поддержка поставщика и обучение*. Предлагает ли продавец внутреннее обучение? Какой тип и уровень поддержки предоставляет поставщик? Легко ли читать документацию по СУБД? Какова политика обновления поставщика?
- *Сторонние инструменты*. Какие дополнительные инструменты предлагают сторонние поставщики? Включают ли они инструменты запросов, словарь данных, средства управления доступом и контролем, а также средства управления распределением памяти?
- *Расходы*. Какие затраты связаны с приобретением программного и аппаратного обеспечения? Сколько дополнительного персонала требуется, и какой уровень знаний требуется от них? Каковы текущие расходы? Каков ожидаемый срок окупаемости?

Если рассматриваются облачные сервисы данных, существуют дополнительные проблемы, которые необходимо решить любому потенциальному поставщику облачных

услуг. Использование облачных баз данных освобождает клиентскую организацию от затрат на приобретение и внедрение инфраструктуры, а также от ежедневных затрат на обслуживание. Однако эти услуги приходят с потерей контроля над данными и инфраструктурой. Любые потенциальные облачные поставщики должны быть оценены на основе нескольких факторов, в том числе:

- *История простоя*. Как часто службы облачного провайдера недоступны, и какие меры они будут принимать, чтобы ваши данные всегда были доступны?
- *Безопасность*. Как поставщик защищает данные с помощью брандмауэров, аутентификации, аудита безопасности и шифрования? Кто в облачной компании будет иметь доступ к файлам данных?
- *Служба поддержки*. Какие варианты поддержки клиентов доступны, если у клиента есть проблемы с предоставляемыми услугами передачи данных?
- *Непредвиденные потери данных*. Ожидается, что облачный провайдер сохранит данные в безопасности. Однако, что произойдет, если они потеряют данные клиента? Какой вид компенсации или страховки от потери данных предоставляется? Какие типы избыточностей и резервных копий используются для предотвращения потери данных? Где хранятся резервные копии и избыточности, чтобы гарантировать, что стихийное бедствие в одном географическом регионе не может привести к потере всех копий данных?

Плюсы и минусы нескольких альтернативных решений должны оцениваться в процессе отбора. Доступные альтернативы часто ограничены, потому что программное обеспечение должно быть совместимо с существующей компьютерной системой организации. СУБД является лишь частью решения; для этого требуется поддержка со стороны сопутствующего оборудования, прикладного программного обеспечения и служебных программ. Например, использование СУБД, скорее всего, будет ограничено доступными ЦП сервера, процессорами клиентских машин, вспомогательными запоминающими устройствами, устройствами передачи данных, операционной системой, системой обработки транзакций и так далее. Затраты, связанные с аппаратными и программными компонентами, должны быть включены в оценки.

АБД должен контролировать установку всего программного и аппаратного обеспечения, которое поддерживает стратегию администрирования данных, и должен тщательно понимать устанавливаемые компоненты, включая процедуры их установки, настройки и запуска. Процедуры установки включают расположение файлов резервных копий и журналов транзакций, информацию о конфигурации сети и сведения о физическом хранилище.

Проектирование и реализация баз данных и приложений. АБД также предоставляет пользователям услуги моделирования и проектирования. Такие службы часто координируются с группой разработчиков приложений в отделе обработки данных. Поэтому одним из основных видов деятельности АБД является определение и обеспечение соблюдения стандартов и процедур, которые будут использоваться. Как только структура

соответствующих стандартов и процедур будет создана, АБД должен обеспечить выполнение действий по моделированию и проектированию БД в рамках этой структуры. Затем АБД предоставляет необходимую помощь и поддержку при проектировании базы данных на концептуальном, логическом и физическом уровнях.

Функция АБД обычно требует, чтобы несколько человек были посвящены в моделирование базы данных и проектирование. Эти люди могут быть сгруппированы в соответствии с организационными областями, охватываемыми приложением. Например, персонал по моделированию и проектированию баз данных может быть назначен на производственные системы, финансовые и управленческие системы или на системы управления и поддержки принятия решений. АБД планирует задания на проектирование для координации действий по проектированию данных и моделированию. Такая координация может потребовать перераспределения имеющихся ресурсов на основе внешних приоритетов.

АБД также работает с прикладными программистами, чтобы обеспечить качество и целостность проекта базы данных и транзакций. Такие службы поддержки включают проверку проекта базы данных, чтобы убедиться, что транзакции:

- *Верные*. Транзакции отражают реальные события.
- *Эффективные*. Транзакции не перегружают СУБД.
- *Соответствующие*. Транзакции соответствуют правилам и стандартам целостности.

Эти действия требуют персонала с широкими навыками проектирования баз данных и программирования.

Реализация приложений требует реализации физической базы данных. Следовательно, АБД должен оказывать помощь и осуществлять надзор во время физического проектирования, включая определение и создание пространства хранения, службы загрузки, преобразования и миграции баз данных. Задачи внедрения АБД также включают создание, компиляцию и хранение плана доступа приложения. *План доступа* – набор инструкций, генерируемых при компиляции приложения, который предопределяет, как приложение будет обращаться к БД во время выполнения. Чтобы иметь возможность создавать и проверять план доступа, пользователь должен обладать необходимыми правами для доступа к базе данных.

Прежде чем приложение будет подключено к сети, АБД должен разработать, протестировать и внедрить рабочие процедуры, необходимые для новой системы. Такие процедуры включают планы обучения, безопасности, резервного копирования и восстановления, а также распределение ответственности за контроль и обслуживание БД. Наконец, АБД должен разрешить пользователям приложений доступ к БД, из которой приложения получают необходимые данные.

Добавление новой БД может потребовать тонкой настройки или перенастройки СУБД. Помните, что СУБД помогает всем приложениям, управляя общим корпоративным хранилищем данных. Поэтому, когда структуры данных добавляются или изменяются, СУБД может потребовать назначения дополнительных ресурсов для обслуживания новых и оригинальных пользователей с равной эффективностью.

Тестирование и оценка баз данных и приложений. АБД также должен предоставлять услуги тестирования и оценки для всех приложений баз данных и пользователей. Эти сервисы являются логическим продолжением сервисов проектирования, разработки и внедрения. Процедуры и стандарты тестирования уже должны быть в наличии, прежде чем любая прикладная программа будет утверждена для использования в компании.

Хотя службы тестирования и оценки тесно связаны со службами проектирования и реализации БД, они обычно поддерживаются независимо. Причина такого разделения заключается в том, что программисты и разработчики БД часто слишком близки к изучаемой проблеме, чтобы обнаружить ошибки и упущения.

Тестирование обычно начинается с загрузки БД «испытательного стенда», который содержит тестовые данные для приложений. Его целью является проверка определения данных и правил целостности БД и прикладных программ.

Тестирование и оценка приложения БД охватывают все аспекты системы, от простого сбора и создания данных до их использования и вывода из эксплуатации. Процесс оценки охватывает следующее:

- Технические аспекты как приложений, так и базы данных; резервное копирование и восстановление, безопасность и целостность, использование SQL и производительность приложений должны быть оценены.
- Оценка письменной документации и процедур, чтобы убедиться, что они точны и просты в применении.
- Соблюдение стандартов именования, документирования и кодирования.
- Проверка на дублирование данных конфликтующих с существующими данными.
- Соблюдение всех правил проверки данных.

После тщательного тестирования всех приложений, базы данных и процедур система объявляется работоспособной и может быть предоставлена конечным пользователям.

Работа с СУБД, утилитами и приложениями. Операции с СУБД можно разделить на четыре основные области:

- Поддержка системы;
- Мониторинг и настройка производительности;
- Резервное копирование и восстановление;
- Аудит и мониторинг безопасности.

Действия по поддержке системы охватывают все задачи, непосредственно связанные с повседневной работой СУБД и ее приложений. Эти действия включают заполнение журналов заданий, смену ленты и проверку состояния компьютерного оборудования, дисковых пакетов и аварийных источников питания. Системные действия включают периодические задачи, такие как запуск специальных программ и конфигурации ресурсов для новых и обновленных версий приложений БД.

Мониторинг и настройка производительности требуют значительных усилий и времени АБД. Эти действия предназначены для обеспечения удовлетворительного уровня

производительности СУБД, утилит и приложений. Для выполнения задач мониторинга и настройки производительности АБД должен:

- Установить цели производительности СУБД.
- Вести мониторинг СУБД для оценки достижения целей производительности.
- Изолировать проблему и найти решения, если цели производительности не достигнуты.
- Внедрить выбранные решения производительности.

СУБД часто включают инструменты мониторинга производительности, которые позволяют АБД запрашивать информацию об использовании БД. Инструменты мониторинга производительности доступны из разных источников: утилиты СУБД предоставляются сторонними поставщиками, или могут быть включены в утилиты операционной системы или средства обработки транзакций. Большинство инструментов мониторинга производительности позволяют АБД сосредоточиться на определенных узких местах системы. Наиболее распространенные узкие места в настройке производительности СУБД связаны с использованием индексов, алгоритмов оптимизации запросов и управления ресурсами хранения.

Поскольку неправильный выбор индекса может отрицательно повлиять на производительность системы, большинство установок СУБД придерживается тщательно определенного плана создания и использования индекса. Такой план особенно важен в среде реляционных БД.

Чтобы обеспечить удовлетворительную производительность, АБД может обучить программистов и пользователей правильному использованию операторов SQL. Как правило, руководства по программированию СУБД и руководства по администрированию содержат полезные рекомендации по производительности и примеры, которые демонстрируют правильное использование операторов SQL как в командной строке, так и в прикладных программах. Поскольку реляционные системы не предоставляют пользователю выбор индекса в запросе, СУБД делает выбор индекса для пользователя. Поэтому АБД должен создавать индексы, которые можно использовать для повышения производительности системы.

Процедуры оптимизации запросов обычно интегрированы в СУБД, что позволяет использовать параметры настройки. Процедуры оптимизации запросов ориентированы на улучшение одновременного доступа к БД. Параллельность также зависит от типов блокировок, используемых СУБД и запрашиваемых приложениями. Поскольку параллелизм важен для эффективной работы системы, АБД должен быть знаком с факторами, влияющими на параллелизм.

При настройке производительности СУБД АБД также должен учитывать доступные ресурсы хранения как с точки зрения основной, так и дополнительной памяти. Распределение ресурсов хранения определяется при настройке СУБД. Параметры конфигурации хранилища могут использоваться для определения:

- Количества баз данных, которые могут быть открыты одновременно.

- Количества прикладных программ или пользователей, поддерживаемых одновременно.
- Объема оперативной памяти (размер пула буферов), назначенного каждой базе данных и каждому процессу.
- Размера и расположения файла журнала.

Проблемы мониторинга производительности зависят от СУБД. Поэтому АБД должен ознакомиться с руководствами по СУБД, чтобы изучить технические детали, связанные с мониторингом производительности.

Поскольку потеря данных может иметь разрушительные последствия для организации, *операции резервного копирования и восстановления* имеют первостепенное значение во время работы СУБД. АБД должен установить расписание для резервного копирования базы данных и файлов журналов через соответствующие промежутки времени. Частота резервного копирования зависит от типа приложения и относительной важности данных. Все критические компоненты системы – БД, приложения базы данных и журналы транзакций – должны периодически архивироваться.

Большинство СУБД содержат утилиты, которые планируют автоматическое резервное копирование БД, полное или инкрементное. Несмотря на то, что инкрементные резервные копии выполняются быстрее, чем полные, для инкрементного резервного копирования требуется наличие периодического полного резервного копирования, что полезно для целей восстановления.

Восстановление БД после сбоя носителя или системы требует применения журнала транзакций к правильной копии. АБД должен спланировать, внедрить, протестировать и внедрить «пуленепробиваемую» процедуру резервного копирования и восстановления.

Аудит и мониторинг безопасности предполагают соответствующее назначение прав доступа и правильное использование прав доступа программистами и пользователями. Технические аспекты аудита и мониторинга безопасности включают создание пользователей, назначение прав доступа и использование команд SQL для предоставления и отзыва прав доступа для пользователей и объектов базы данных. АБД также должен периодически генерировать отчет журнала аудита, чтобы найти фактические или предпринятые нарушения безопасности. Если таковые обнаружены, АБД должен выяснить, где произошли нарушения и, если возможно, кто их совершил.

Обучение и поддержка пользователей. Обучение людей использованию СУБД и ее инструментов является частью технической деятельности АБД. Кроме того, АБД обеспечивает техническое обучение для прикладных программистов по использованию СУБД и ее утилит. Обучение программиста приложений охватывает использование инструментов СУБД, а также процедуры и стандарты, необходимые для программирования БД.

Незапланированная техническая поддержка по требованию для пользователей и программистов также является частью деятельности АБД. Техническая процедура устранения неполадок может быть разработана для облегчения такой поддержки. Процедура может

включать разработку технической базы данных для поиска решений общих технических проблем.

Часть поддержки обеспечивается взаимодействием с поставщиками СУБД. Установление хороших отношений с поставщиками программного обеспечения – один из способов обеспечить компанию хорошим источником внешней поддержки. Продавцы являются источником актуальной информации о новых продуктах и переподготовке персонала. Хорошие отношения между поставщиком и компанией также могут дать организациям преимущества в определении будущего направления развития БД.

Обслуживание СУБД, утилит и приложений. Обслуживание является продолжением операционной деятельности АБД. Деятельность по обслуживанию посвящена сохранению среды СУБД.

Периодическое обслуживание СУБД включает управление физическими устройствами хранения. Одним из наиболее распространенных действий по обслуживанию является реорганизация физического расположения данных в БД. Реорганизация БД может быть предназначена для выделения смежных расположений страниц на диске для СУБД для повышения производительности. Процесс реорганизации также может освободить пространство, выделенное для удаленных данных, что даст больше места на диске для новых данных.

Действия по обслуживанию также включают обновление СУБД и служебного программного обеспечения. Для обновления может потребоваться установка новой версии программного обеспечения СУБД или интерфейсного средства в Интернете; или создание дополнительного шлюза СУБД, чтобы разрешить доступ к СУБД хоста, работающей на другом хост-компьютере. Услуги шлюза СУБД очень распространены в распределенных приложениях СУБД, работающих в среде клиент/сервер. Кроме того, БД нового поколения включают такие функции, как поддержка пространственных данных, хранилище данных и поддержка запросов типа «звезда», а также поддержка интерфейсов программирования Java для доступа в Интернет.

Довольно часто компании сталкиваются с необходимостью обмена данными в разных форматах или между базами данных. Усилия АБД по обслуживанию включают в себя услуги по миграции и конвертацию данных в несовместимых форматах или для другого программного обеспечения СУБД. Такие условия распространены, когда система обновляется с одной версии на другую, или когда существующая СУБД заменяется совершенно новой СУБД. Услуги по миграции и конвертации могут быть выполнены на логическом уровне (специфично для СУБД или программного обеспечения) или на физическом уровне (носитель данных или операционная система). СУБД текущего поколения поддерживают XML как стандартный формат для обмена данными между системами баз данных и приложениями.

9.6. Безопасность

Безопасность информационной системы относится к действиям и мерам, которые обеспечивают конфиденциальность, целостность и доступность информационной системы и

ее основного актива, данных. Защита данных требует комплексного подхода в масштабах всей компании. То есть нельзя защитить данные, если не защитить все процессы и системы вокруг них, включая аппаратные системы, программные приложения, сеть и ее устройства, внутренних и внешних пользователей, процедуры и сами данные. Чтобы понять сферу безопасности данных, рассмотрим каждую из трех целей безопасности более подробно:

- *Конфиденциальность* означает, что данные защищены от несанкционированного доступа, а если к данным обращается авторизованный пользователь, они используются только для авторизованных целей. Другими словами, конфиденциальность влечет за собой защиту данных от разглашения любой информации, которая нарушает права человека или организации на неприкосновенность частной жизни. Данные должны оцениваться и классифицироваться в соответствии с уровнем конфиденциальности: строго ограничены (очень мало людей имеют доступ), конфиденциальны (только определенные группы имеют доступ) и открыты (доступ для всех пользователей). Сотрудник по защите данных тратит много времени на то, чтобы обеспечить соблюдение организацией требуемых уровней конфиденциальности. Под *соответствием* понимаются действия, которые соответствуют правилам составления отчетов о конфиденциальности и безопасности данных.
- *Целостность* в рамках системы защиты данных связана с сохранением данных согласованными и свободными от ошибок или аномалий. СУБД играет ключевую роль в обеспечении целостности данных в базе данных. Однако с точки зрения безопасности, организационные процессы, пользователи и шаблоны использования также должны поддерживать целостность. Например, работающий на дому сотрудник, использует Интернет для доступа к стоимости продукта – такое использование считается приемлемым; однако стандарты безопасности могут требовать от сотрудника использования защищенного соединения и соблюдения строгих процедур для управления данными в домашних условиях, таких как уничтожение печатных отчетов и использование шифрования для копирования данных на локальный жесткий диск. Поддержание целостности данных – процесс, который начинается со сбора данных и продолжается с хранения, обработки, использования и архивирования данных. Обоснование целостности заключается в том, чтобы рассматривать данные как наиболее ценный актив в организации и обеспечивать тщательную проверку данных на всех уровнях внутри организации.
- *Доступность* означает доступность данных, когда это требуется авторизованным пользователям и для авторизованных целей. Чтобы обеспечить доступность данных, вся система должна быть защищена от ухудшения качества обслуживания или прерывания работы, вызванного любым внутренним или внешним источником. Перерывы в обслуживании могут быть очень дорогостоящими как для компаний, так и для пользователей. Доступность системы является важной целью безопасности.

Политика безопасности. Обычно задачи по защите системы и данных выполняются сотрудником по безопасности БД и АБД, которые работают вместе, чтобы создать единую стратегию безопасности данных. Такая стратегия начинается с определения разумной и всеобъемлющей политики безопасности. *Политика безопасности* – набор стандартов, политик и процедур, созданных для обеспечения безопасности системы и обеспечения аудита и соответствия требованиям. Процесс аудита безопасности начинается с выявления уязвимостей в инфраструктуре информационной системы организации и определения мер по защите системы и данных от этих уязвимостей.

Уязвимость безопасности. *Уязвимость в системе безопасности* – это слабость в системном компоненте, которую можно использовать для несанкционированного доступа или нарушения работы службы. Такие уязвимости могут быть отнесены к одной из следующих категорий:

- *Технический*. Примером может служить ошибка в операционной системе или веб-браузере.
- *Управленческий*. Например, организация может не информировать пользователей о критических проблемах безопасности.
- *Культурный*. Пользователи могут скрыть пароли под своими клавиатурами или забыть уничтожить конфиденциальные отчеты.
- *Процедурный*. Процедуры компании могут не требовать сложных паролей или проверки идентификаторов пользователей.

Если оставить уязвимость безопасности без внимания, она может стать угрозой безопасности. *Угроза безопасности* является неизбежным нарушением безопасности.

Нарушение безопасности происходит, когда угроза безопасности используется для того, чтобы поставить под угрозу целостность, конфиденциальность или доступность системы. Нарушения безопасности могут привести к проблемам в базе данных, целостность которой либо сохранена, либо повреждена:

- *Сохранена*. В этих случаях необходимо предпринять действия, чтобы избежать повторения подобных проблем безопасности, но восстановление данных может не потребоваться. На самом деле, большинство нарушений безопасности вызваны несанкционированным и незамеченным доступом в информационных целях, но такое отслеживание не нарушает работу базы данных.
- *Повреждена*. Необходимо предпринять действия, чтобы избежать повторения подобных проблем безопасности, и база данных должна быть восстановлена до согласованного состояния. Уязвимые нарушения безопасности включают доступ к базе данных компьютерными вирусами и хакерами, которые намереваются уничтожить или изменить данные.

Таблица 30 иллюстрирует некоторые уязвимости в безопасности компонентов системы и типичные меры защиты от них.

Таблица 30

Типовые уязвимости безопасности и связанные с ними защитные меры

Компонент системы	Уязвимость безопасности	Меры безопасности
Люди	• Пользователь устанавливает пустой пароль. • Пароль короткий или содержит дату рождения. • Пользователь оставляет дверь офиса открытой все время. • Пользователь оставляет информацию о заработной плате на экране в течение длительных периодов времени.	• Применение сложных политик паролей. • Использовать многоуровневую аутентификацию. • Использовать защитные экраны и заставки. • Обучать пользователей конфиденциальным данным. • Установить камеры безопасности. • Использовать автоматические дверные замки.
Рабочая станция и серверы	• Пользователь копирует данные на флешку. • Рабочая станция используется несколькими пользователями. • Сбой питания приводит к сбою компьютера. • Несанкционированный персонал может использовать компьютер. • Чувствительные данные хранятся на ноутбуке. • Данные теряются из-за кражи жесткого диска или ноутбука. • Происходит стихийное бедствие.	• Использовать групповые политики для ограничения использования флешек. • Назначить права доступа пользователей к рабочим станциям. • Установить источники бесперебойного питания (ИБП). • Добавить защитные блокировки на компьютеры. • Реализовать переключатель уничтожения для украденных ноутбуков. • Создание и тестирование планов резервного копирования и восстановления данных. • Защитить систему от стихийных бедствий – использовать стратегии совместного размещения.
Операционная система	• Атаки переполнения буфера. • Вирусные атаки. • Руткиты и атаки червей. • Атаки типа «отказ в обслуживании». • Троянские кони. • Шпионские программы. • Взломщики паролей.	• Применять исправления и обновления безопасности ОС. • Применять исправления сервера приложений. • Установить антивирусное и антишпионское программное обеспечение. • Применять контрольные журналы на компьютерах.

Компонент системы	Уязвимость безопасности	Меры безопасности
		• Выполнять периодическое резервное копирование системы. • Устанавливать только авторизованные приложения. • Использовать групповые политики для предотвращения несанкционированных установок.
Приложения	• Ошибки приложения – переполнение буфера. • Инъекция SQL, перехват сеанса и т.д. • Уязвимости приложения – межсайтовый скриптинг, неподтвержденные данные. • Атаки по электронной почте – спам, фишинг и т.д. • Электронные письма социальной инженерии.	• Тщательно тестировать прикладные программы. • Встроить меры защиты в код. • Проводить обширное тестирование уязвимостей в приложениях. • Установить спам-фильтры и антивирусное программное обеспечение для систем электронной почты. • Использовать методы безопасного кодирования. • Обучать пользователей атакам социальной инженерии.
Сеть	• IP-спуфинг. • Пакетные снифферы. • Хакерские атаки. • Удаление паролей в сети.	• Установить брандмауэры. • Использовать виртуальные частные сети (VPN). • Использовать системы обнаружения вторжений (IDS). • Использовать контроль доступа к сети (NAC). • Использовать мониторинг сетевой активности.
Данные	• Обмен данными открыт для всех пользователей. • Данные могут быть доступны удаленно. • Данные могут быть удалены из общего ресурса.	• Реализовать безопасность файловой системы. • Реализовать безопасность доступа к общим ресурсам. • Использовать разрешение на доступ. • Шифровать данные на уровне файловой системы или базы данных.

Безопасность базы данных. *Безопасность базы данных* относится к функциям СУБД и другим связанным мерам, которые соответствуют требованиям безопасности организации.

С точки зрения администратора баз данных, должны быть приняты меры безопасности для защиты СУБД от ухудшения качества обслуживания и защиты базы данных от потери, повреждения или неправильного обращения. АБД должен защищать СУБД от начала установки до эксплуатации и обслуживания.

Чтобы защитить СУБД от снижения качества обслуживания, рекомендуются некоторые меры безопасности. Например:

- Изменить системные пароли по умолчанию.
- Изменить пути установки по умолчанию.
- Применять последние патчи.
- Безопасные установочные папки с соответствующими правами доступа.
- Убедиться, что работают только необходимые службы.
- Настроить журналы аудита.
- Настроить ведение журнала сеанса.
- Требовать шифрование сеанса.

Кроме того, АБД должен тесно сотрудничать с сетевым администратором для обеспечения безопасности сети, которая защищает СУБД и все службы, работающие в сети. В современных организациях одним из наиболее важных компонентов информационной архитектуры является сеть.

Защита данных в базе данных является функцией управления авторизацией. *Управление авторизацией* определяет процедуры для защиты и обеспечения безопасности, и целостности БД. Эти процедуры включают в себя следующее:

- *Управление доступом пользователей*. Эта функция предназначена для ограничения доступа к БД; она включает следующие процедуры:
  - *Определить каждого пользователя в базе данных*. АБД выполняет эту функцию на уровне операционной системы и на уровне СУБД. На уровне операционной системы АБД может запросить создание уникального идентификатора для каждого пользователя, который входит в систему компьютера. На уровне СУБД АБД может либо создать другой идентификатор, либо использовать тот же для предоставления пользователю доступа к СУБД.
  - *Назначить пароли каждому пользователю*. АБД также выполняет эту функцию как на уровне операционной системы, так и на уровне СУБД. Пароли базы данных могут быть назначены с заранее установленными датами истечения срока действия, что позволяет АБД периодически проверять пользователей и напоминать им об изменении их паролей, что снижает вероятность несанкционированного доступа.
  - *Определить группы пользователей*. Классификация пользователей по группам в соответствии с общими потребностями доступа может помочь АБД контролировать и управлять привилегиями доступа отдельных пользователей.

Кроме того, АБД может использовать роли базы данных и ограничения ресурсов для минимизации влияния мошеннических пользователей в системе.

- *Назначить права доступа*. АБД назначает права доступа конкретным пользователям для доступа к определенным базам данных. Права доступа могут быть ограничены только для чтения, или авторизованный доступ может включать в себя права на чтение, запись и удаление. Права доступа в реляционных базах данных назначаются с помощью команд SQL GRANT и REVOKE.
- *Контроль физического доступа*. Физическая безопасность может предотвратить доступ неавторизованных пользователей к установке и средствам СУБД. Общая физическая защита для больших баз данных включает защищенные входы, рабочие станции, защищенные паролем, электронные пропуска для персонала, видео с замкнутым контуром, распознавание голоса и биометрические технологии.

- *Просмотр определения*. АБД должен определить представления данных для защиты и контроля объема данных, доступных для авторизованного пользователя. СУБД должна предоставлять инструменты, позволяющие определять представления, состоящие из одной или нескольких таблиц, и назначать права доступа пользователям. Команда SQL CREATE VIEW используется в реляционных базах данных для определения представлений. СУБД Oracle предлагает виртуальную частную базу данных (VPD), которая позволяет создавать настраиваемые представления данных для разных пользователей. С помощью этой функции АБД может ограничить обычных пользователей, которые запрашивают базу данных заработной платы, чтобы видеть только необходимые строки и столбцы, в то время как руководители отделов будут видеть только строки и столбцы, относящиеся к их отделам.
- *Контроль доступа к СУБД*. Доступ к базе данных можно контролировать, устанавливая ограничения на использование инструментов, запросов и отчетов СУБД. АБД должен убедиться, что инструменты используются правильно и только уполномоченным персоналом.
- *Мониторинг использования СУБД*. АБД также должен проверить использование данных в БД. Многие СУБД содержат функции, позволяющие создавать журнал аудита, который автоматически записывает краткое описание операций базы данных, выполняемых всеми пользователями. Такие контрольные журналы позволяют точно определять нарушения прав доступа. Журналы аудита могут быть адаптированы для записи всех обращений к базе данных или просто неудачных.

Целостность базы данных может быть нарушена из-за внешних факторов, неподконтрольных АБД. Например, БД может быть повреждена или разрушена в результате взрыва, пожара или землетрясения. Безотносительно причины, спектр повреждения или

разрушения базы данных делает процедуры резервного копирования и восстановления критически важными для любого АБД.

9.7. Инструменты администрирования базы данных

Высокий рост деятельности по управлению данными в организациях привел к необходимости улучшения стандартов управления, процессов и инструментов. За прошедшие годы возникла новая индустрия, посвященная исключительно инструментам администрирования данных. Эти инструменты охватывают весь спектр задач, включая выбор, развертывание, миграцию и повседневные операции. Например, можно найти сложные инструменты администрирования данных для:

- Мониторинга;
- Нагрузочного тестирования;
- Настройки производительности;
- Оптимизации кода SQL;
- Определения и устранения узких мест в базе данных;
- Моделирования и проектирования БД;
- Извлечения, конвертации и загрузки данных.

Все вышеперечисленные инструменты имеют что-то общее. Все они расширяют базу метаданных или словарь данных. Важность словаря данных как инструмента АБД невозможно переоценить.

Словарь данных. Словарь данных компонент СУБД, который хранит определение свойства данных и отношений. Такие данные о данных называются метаданными. Словарь данных СУБД обеспечивает ее свойством самоописания.

Существуют два основных типа словарей данных: интегрированные и автономные. Интегрированный словарь данных входит в состав СУБД. Например, все реляционные СУБД включают в себя встроенный словарь данных или системный каталог, к которому СУРБД часто обращаются и обновляют. Другие СУБД, особенно старые типы, не имеют встроенного словаря данных; вместо этого АБД может использовать сторонние автономные системы.

Словари данных также могут быть классифицированы как активные или пассивные. *Активный словарь данных* автоматически обновляется при каждом доступе к базе данных. *Пассивный словарь данных* не обновляется автоматически и обычно требует запуска пакетного процесса. Информация о доступе к словарю данных обычно используется СУБД для оптимизации запросов.

Основная функция словаря данных – хранить описание всех объектов, которые взаимодействуют с базой данных. Интегрированные словари данных, как правило, ограничивают свои метаданные данными, управляемыми СУБД. Автономные системы словарей данных обычно более гибкие и позволяют АБД описывать и управлять всеми данными организации, независимо от того, компьютеризированы они или нет. Каким бы ни был формат словаря данных, он предоставляет разработчикам баз данных и пользователям

значительно улучшенную способность общаться. Кроме того, словарь данных является инструментом, который помогает АБД разрешать конфликты данных.

Хотя нет стандартного формата для информации, хранящейся в словаре данных, некоторые функции являются общими. Например, словарь данных обычно хранит описания следующего:

- *Элементы данных, определенные во всех таблицах всех баз данных.* В частности, в словаре данных хранятся имена элементов, типы данных, формат отображения, формат внутреннего хранения и правила проверки. Словарь данных объясняет, где используется элемент, кто его использовал и так далее.
- *Таблицы, определенные во всех базах данных*. Например, словарь данных, вероятно, будет хранить имя создателя таблицы, дату создания, права доступа и количество столбцов.
- *Индексы, определенные для каждой таблицы базы данных*. Для каждого индекса СУБД хранит как минимум имя индекса, используемые атрибуты, местоположение, конкретные характеристики индекса и дату создания.
- *Определение базы данных*. Эта информация включает сведения о том, кто создал каждую базу данных, когда она была создана, где находится база данных, имя АБД и т.д.
- *Пользователи и администраторы базы данных*. Эта информация определяет пользователей базы данных.
- *Программы, которые обращаются к базе данных*. Эта информация включает в себя форматы экрана, форматы отчетов, прикладные программы и запросы SQL.
- *Доступ к авторизации для всех пользователей всех баз данных*. Эта информация определяет, кто может манипулировать, какие объекты и какие типы операций могут быть выполнены.
- *Связи между элементами данных*. Эта информация включает сведения о том, какие элементы задействованы, являются ли связи обязательными или необязательными, а также требования к связности и количеству элементов.

Если словарь данных может быть организован так, чтобы включать данные, внешние по отношению к самой СУБД, он становится особенно гибким инструментом для более общего управления корпоративными ресурсами. Такой обширный словарь данных, таким образом, позволяет управлять использованием и распределением всей информации организации, независимо от того, имеет ли она свои корни в данных базы данных. По этой причине некоторые менеджеры считают словарь данных ключевым элементом управления информационными ресурсами, поэтому словарь данных можно описать как *словарь информационных ресурсов*.

Метаданные, хранящиеся в словаре данных, часто являются основой для мониторинга использования базы данных и для назначения прав доступа пользователям. Информация, хранящаяся в словаре данных, обычно основана на формате реляционных таблиц, что

позволяет АБД запрашивать базу данных с помощью команд SQL. Например, команды SQL могут использоваться для извлечения информации о пользователях конкретной таблицы или правах доступа конкретного пользователя.

АБД может использовать словарь данных для анализа и проектирования данных. Например, АБД может создать отчет, в котором перечислены все элементы данных, которые будут использоваться в конкретном приложении; список всех пользователей, имеющих доступ к определенной программе; отчет, который проверяет избыточность данных, дублирование и использование омонимов и синонимов; и ряд других отчетов, которые описывают пользователей данных, доступ к данным и структуру данных. Словарь данных также можно использовать для обеспечения того, чтобы прикладные программисты соблюдали стандарты именования элементов данных в базе данных, и чтобы правила проверки данных были верными. Таким образом, словарь данных может использоваться для поддержки широкого спектра действий по администрированию данных и для облегчения проектирования и внедрения информационных систем. Интегрированные словари данных также важны для использования инструментов системного проектирования.

Case средства. *CASE* – набор инструментов и методов программной инженерии для проектирования программного обеспечения, который помогает обеспечить высокое качество программ, отсутствие ошибок и простоту в обслуживании программных продуктов. Также под CASE понимают совокупность методов и средств проектирования информационных систем с использованием CASE-инструментов. Инструмент CASE обеспечивает автоматизированную среду жизненного цикла разработки систем (ЖЦ ИС). CASE использует структурированные методологии и мощные графические интерфейсы. Поскольку они автоматизируют многие утомительные действия по проектированию и внедрению систем, инструменты CASE играют все более важную роль в разработке информационных систем.

Инструменты CASE обычно классифицируются в зависимости от степени поддержки, которую они предоставляют ЖЦ ИС. Например, *интерфейсные инструменты CASE* обеспечивают поддержку на этапах планирования, анализа и проектирования; *Внутренние инструменты CASE* обеспечивают поддержку на этапах кодирования и реализации. Преимущества, связанные с инструментами CASE, включают в себя:

- Сокращение времени разработки и затрат;
- Автоматизацию ЖЦ ИС;
- Стандартизацию методологий разработки систем;
- Более простое обслуживание прикладных систем, разработанных с помощью инструментов CASE.

Одним из наиболее важных компонентов инструментов CASE является обширный словарь данных, который отслеживает все объекты, созданные разработчиком системы. Например, словарь данных CASE хранит диаграммы потоков данных, структурные схемы, описания всех внешних и внутренних объектов, хранилища данных, элементы данных, форматы отчетов и форматы экрана. Словарь данных CASE также описывает отношения между компонентами системы.

Несколько инструментов CASE предоставляют интерфейсы, которые работают с СУБД, и позволяют инструменту CASE хранить информацию своего словаря данных с использованием СУБД. Такое взаимодействие демонстрирует взаимозависимость, существующую между разработкой систем и базой данных, и помогает создать полностью интегрированную среду разработки.

В среде разработки CASE разработчики баз данных и приложений используют инструмент CASE для хранения описания схемы базы данных, элементов данных, процессов приложения, экранов, отчетов и других данных, относящихся к разработке. Инструмент CASE объединяет всю информацию о разработке систем в едином хранилище, которое АБД может проверить на согласованность и точность.

В качестве дополнительного преимущества среда CASE имеет тенденцию улучшать степень и качество связи между АБД, разработчиками приложений и пользователями. АБД может использовать инструмент CASE для проверки определения схемы данных приложения, соблюдения соглашений об именах, дублирования элементов данных, правил проверки для элементов данных и множества других переменных развития и управления. Когда инструмент CASE обнаруживает конфликты, нарушения правил и несоответствия, он облегчает внесение исправлений. Более того, инструмент CASE может вносить исправления и затем каскадно воздействовать на всю среду приложений, что значительно упрощает работу АБД и разработчика приложений.

Типичный инструмент CASE содержит пять компонентов:

- Графику, предназначенную для создания структурированных диаграмм, таких как диаграммы потоков данных, диаграммы ER, диаграммы классов и диаграммы объектов;
- Создатели экранных форм для ввода и генераторы отчетов для вывода информационной системы;
- Интегрированный репозиторий для хранения и перекрестных ссылок на данные проектирования системы;
- Сегмент анализа для обеспечения полностью автоматической проверки согласованности, синтаксиса и полноты системы;
- Генератор программной документации.

Большинство инструментов CASE создают полностью документированные ER-диаграммы, которые могут отображаться на разных уровнях абстракции. Например, ERwin Data Modeler от Computer Associates может создавать подробные реляционные проекты. Пользователь указывает атрибуты и первичные ключи для каждого объекта и описывает отношения. Инструменты моделирования данных текущего поколения назначают внешние ключи на основе указанных отношений между объектами. Изменения в первичных ключах всегда обновляются автоматически по всей системе. В таблице 31 приведен краткий список доступных поставщиков инструментов для моделирования данных CASE.

Таблица 31

Инструменты CASE

Разработчик	Продукт	Сайт
Erwin Inc.	erwin Evolve	www.erwin.com
Idera Inc.	ER/Studio Data Architect	www.embarcadero.com/products/er-studio-data-architect
Microsoft	Visio	https://www.microsoft.com/ru-ru/microsoft-365/visio/flowchart-software
Oracle	SQL Developer Data Modeler	https://www.oracle.com/database/technologies/appdev/datamodeler.html
IBM	Engineering Systems Design Rhapsody	https://www.ibm.com/products/architect-for-software
SAP	Power Designer	www.sap.com/products/powerdesigner-data-modeling-tools.html
Visible Systems	Visible Analyst	http://www.visiblesystemscorp.com/Products/Analyst/index.htm
Visual Paradigm	Visual Paradigm Community Edition	https://www.visual-paradigm.com/download/community.jsp

Основные поставщики реляционных СУБД, такие как Oracle, в настоящее время предоставляют полностью интегрированные инструменты CASE для собственного программного обеспечения СУБД, а также для СУБД, поставляемых другими поставщиками. Например, инструменты Oracle CASE можно использовать с IBM DB2 и Microsoft SQL Server для создания полностью документированных проектов баз данных. Некоторые поставщики даже используют нереляционные СУБД, разрабатывают свои схемы и автоматически создают эквивалентные реляционные схемы.

Нет сомнений в том, что инструменты CASE повысили эффективность разработчиков баз данных и приложений. Однако независимо от того, насколько сложен инструмент CASE, его пользователи должны хорошо разбираться в концептуальном дизайне. В руках новичков в базе данных инструменты CASE производят впечатляющий, но плохой проект.

9.8. Разработка стратегии управления данными

Для успешной работы предприятия его деятельность должна соответствовать основным целям и задачам. Поэтому, независимо от масштаба компании, критически важным шагом для любой организации является обеспечение того, чтобы ее информационная система поддерживала стратегические планы для каждой области бизнеса.

Стратегия администрирования базы данных не должна противоречить планам информационных систем. В конце концов, эти планы основаны на подробном анализе целей компании, ее состояния или ситуации и потребностей бизнеса. Существует несколько методологий для обеспечения совместимости планов администрирования данных и информационных систем, а также для руководства разработкой стратегического плана. Наиболее часто используемая методология известна как информационная инженерия.

Информационная инженерия позволяет переводить стратегические цели компании в данные и приложения, которые помогут компании достичь этих целей. Она фокусируется на описании корпоративных данных вместо процессов. Обоснование простое: типы бизнес-данных обычно остаются достаточно стабильными, но процессы часто меняются и, следовательно, требуют частой модификации существующих систем. Делая упор на данные, информационная инженерия помогает уменьшить влияние на системы при изменении процессов.

Результатом процесса информационной инженерии является *архитектура информационных систем (АИС)*, которая служит основой для планирования, разработки и управления будущими информационными системами.

Внедрение информационной инженерии в организации дорогостоящий процесс, который включает планирование, выделение ресурсов, ответственность руководства, четко определенные цели, выявление критических факторов и контроль. АИС предоставляет платформу, которая включает компьютеризированные, автоматизированные и интегрированные инструменты, такие как инструменты СУБД и CASE.

Успех общей стратегии информационных систем и стратегии администрирования данных зависит от нескольких критических факторов успеха, которые АБД должен понимать. Критические факторы успеха включают следующие проблемы управленческой, технологической и корпоративной культуры:

- *Приверженность руководству*. Приверженность руководству высшего уровня необходима для обеспечения применения стандартов, процедур, планирования и контроля. Пример должен быть установлен сверху.
- *Тщательный анализ ситуации в компании*. Текущее состояние администрирования корпоративных данных должно быть проанализировано, чтобы понять позицию компании и иметь четкое представление о том, что должно быть сделано. Например, как обрабатываются анализ базы данных, проектирование, документация, реализация, стандарты, кодификация и другие вопросы? Сначала необходимо определить потребности и проблемы, а затем определить их приоритетность.
- *Участие пользователей*. Какова степень организационных изменений? Успешное изменение требует, чтобы люди были в состоянии приспособиться к нему. Пользователи должны иметь открытый канал связи с высшим руководством для обеспечения успеха внедрения. Хорошее общение является ключом к общему процессу.
- *Определенные стандарты*. Аналитики и программисты должны быть знакомы с соответствующими методологиями, процедурами и стандартами. Если это не так, им может потребоваться обучение.
- *Повышение квалификации*. Поставщик должен обучить персонал АБД использованию СУБД и других инструментов. Пользователи должны быть обучены использованию инструментов, стандартов и процедур. Ключевой персонал должен быть обучен в первую очередь, чтобы он мог обучать других.

- *Небольшой пилотный проект*. Небольшой проект рекомендуется для обеспечения того, чтобы СУБД работала в компании, чтобы она производила ожидаемый результат, и чтобы персонал был должным образом подготовлен.

Этот список факторов не является исчерпывающим, но он обеспечивает основу для разработки успешной стратегии. Независимо от того, насколько всеобъемлющим будет список, он должен основываться на разработке и реализации стратегии администрирования данных, которая тесно интегрирована с общим планированием информационных систем организации.

Разработка комплексной стратегии администрирования данных в организации – это большое начинание, включающее технические, операционные и управленческие роли. Сегодня у предприятий также есть возможность перенести целые вычислительные функции (например, серверы, хранилище, резервное копирование и даже базу данных) за пределы предприятия и в облако.

9.9. Роль АБД в облаке

Использование облачных сервисов данных оказывает существенное влияние на роль АБД. Такие сервисы, как Microsoft Azure и Amazon Web Services (AWS), позволяют использовать технологию баз данных на аутсорсинге как высоко масштабируемую услугу по требованию. В этом новом мире некоторые задачи, которые когда-то выполнялись в единой «внутренней» функции АБД, теперь разделены между внутренними администраторами и поставщиком облачных услуг. В результате, использование облачных служб данных изменяет и расширяет типичную роль АБД в техническом и в управленческом аспектах. В целом компания-партнер по облачным сервисам предоставляет:

- *Установку и обновление СУБД*. СУБД установлена на виртуальном сервере поставщиком услуг. Поскольку поставщик СУБД выпускает необходимые обновления и исправления безопасности для программного обеспечения СУБД, поставщик услуг управляет применением обновлений в пределах указанного окна обслуживания. Теперь роль АБД заключается в тщательной координации таких обновлений с внешним поставщиком облачных данных.
- *Управление сервером/сетью*. Поставщик услуг настраивает и управляет сервером, на котором находится СУБД, включая масштабирование базы данных по нескольким серверам по мере необходимости. Если база данных распределена по нескольким серверам, поставщик услуг может обеспечить балансировку нагрузки для обеспечения высокого уровня производительности. Однако АБД должен работать с сетевым отделом своей компании, чтобы убедиться, что сеть правильно настроена для обеспечения безопасности, производительности, доступности и управления.
- *Операции резервного копирования и восстановления*. Поставщик услуг выполняет регулярные резервные копии и хранит резервные копии в безопасных средствах.

АБД должен обеспечить соблюдение и поддержку внутренних политик конфиденциальности и хранения данных.

Хотя эти сервисы являются ценными и освобождают АБД от выполнения этих задач, основным преимуществом облачных сервисов данных является их способность предоставлять и управлять конфигурацией компьютерного оборудования и программного обеспечения при низких затратах. Предыдущие задачи – только малая часть обязанностей АБД; Управляющая роль АБД практически не меняется, а иногда даже дополняется новым измерением облачных сервисов данных. Пользовательские требования все еще должны быть собраны; решения для данных все еще должны быть разработаны; пользователи нуждаются в обучении; политики, стандарты и процедуры должны быть разработаны и внедрены.

Даже техническая роль АБД все еще существует с использованием облачных сервисов данных. Существует много поставщиков облачных услуг передачи данных, и некоторые предлагают различные продукты СУБД, включая проприетарные системы. Доступны только некоторые версии этих СУБД, включая несколько версий одной и той же СУБД. Например, данный поставщик услуг может поддерживать как MySQL 5.1, так и MySQL 5.5. В этой среде АБД оценивает разные СУБД, чтобы определить, какой программный продукт использовать, и оценивает, у какого поставщика приобрести СУБД. Кроме того, АБД должен работать с поставщиком услуг облачных данных, чтобы согласовать необходимые технические характеристики базы данных с теми, которые поддерживаются поставщиком услуг облачных данных, и обеспечить доступность, безопасность и целостность данных в расширенных границах сети компании.

Поставщики услуг облачных данных предлагают различные схемы ценообразования. Ценообразование обычно основывается на таких факторах, как объем памяти, вычислительные ресурсы (циклы ЦП и память) и размеры передаваемых данных. Пользователям сервиса ежемесячно выставляется счет за количество использованных ресурсов. Поставщики услуг заинтересованы в том, чтобы базы данных их клиентов были как можно большего размера; Кроме того, они заинтересованы в том, чтобы структуры баз данных были неэффективными при обработке запросов, поскольку клиентам придется покупать больше памяти и процессорных ресурсов. Поставщики услуг выигрывают, если база данных заполнена плохо спроектированными таблицами, которые содержат много ненужных избыточных данных, с каждым индексируемым атрибутом в каждой таблице и запросами, которые требуют много времени для запуска или возвращают тысячи строк данных, которые должны быть переданы в интерфейсное приложение для дополнительной обработки. Таким образом, АБД может сэкономить организации значительное время и деньги, обеспечивая правильное проектирование баз данных с минимальной избыточностью и эффективное кодирование баз данных. Очевидно, что техническая роль АБД по-прежнему важна для организаций, которые используют облачные службы данных. Усилия АБД по эффективному и результативному проектированию базы данных, кодированию, мониторингу производительности базы данных и настройке базы данных все еще влияют на способность

организации использовать данные и информацию в качестве ресурса, и они оказывают непосредственное видимое влияние на ежемесячный счет за обслуживание данных.

Независимо от того, хранится ли база данных на сервере предприятия или в облаке, администратор БД должен обеспечивать доступность, безопасность и целостность данных.

9.10. Итоги

- Управление данными является критически важным видом деятельности для любой организации, поэтому данные должны рассматриваться как корпоративный актив. Ценность набора данных измеряется полезностью информации, полученной из него. Хорошее управление данными, вероятно, даст хорошую информацию, которая является основой для более эффективного принятия решений.
- Качество данных – это комплексный подход к обеспечению точности, достоверности и своевременности данных. Качество данных фокусируется на исправлении грязных данных, предотвращении будущих неточностей в данных и укреплении доверия пользователей к данным.
- СУБД является наиболее часто используемым инструментом для управления корпоративными данными. СУБД поддерживает принятие стратегических, тактических и оперативных решений на всех уровнях организации. Внедрение СУБД в организацию – дело деликатное; влияние СУБД на управленческие и культурные рамки организации должно быть тщательно изучено.
- Администратор базы данных (АБД) отвечает за управление корпоративной БД. Внутренняя организация администрирования баз данных варьируется от компании к компании. Хотя стандартов не существует, обычной практикой является разделение операций АБД в соответствии с фазами жизненного цикла базы данных. Некоторые компании создали положение с более широкими полномочиями по управлению компьютеризированными данными и другими данными; это действие обрабатывается администратором данных (АД).
- Функции АД и АБД имеют тенденцию перекрываться. Вообще говоря, у АД есть больше управленческих задач, чем у технически ориентированных АБД. По сравнению с функцией АБД функция АД не зависит от СУБД с более широкой и долгосрочной направленностью. Однако, когда организация не включает должность АД, АБД выполняет все его функции. В этой объединенной роли АБД должен обладать разнообразным сочетанием технических и управленческих навыков.
- Управленческие услуги АБД включают поддержку пользователей; определение и обеспечение соблюдения политик, процедур и стандартов для БД; обеспечение безопасности, конфиденциальности и целостности данных; предоставление услуг резервного копирования и восстановления данных; и мониторинг распространения и использования данных.

- Техническая роль АБД требует участия по крайней мере в следующих действиях: оценка, выбор и установка СУБД; разработка и внедрение; тестирование и оценка БД и приложений; эксплуатация и обслуживание СУБД, утилит и приложений; обучение и поддержка пользователей.
- Безопасность относится к действиям и мерам, обеспечивающим конфиденциальность, целостность и доступность информационной системы и ее основного ресурса – данных. Политика безопасности – это набор стандартов, политик и практик, которые гарантируют безопасность системы и обеспечивают аудит и соответствие требованиям.
- Уязвимость в системе безопасности – это слабость в системном компоненте, которая может быть использована для несанкционированного доступа или нарушения работы службы. Угроза безопасности – это неизбежное нарушение безопасности, вызванное неконтролируемой уязвимостью. Уязвимости безопасности существуют во всех компонентах информационной системы: люди, оборудование, программное обеспечение, сеть, процедуры и данные. Поэтому очень важно иметь надежную защиту базы данных. Безопасность БД относится к функциям СУБД и соответствующим мерам, которые соответствуют требованиям безопасности организации.
- Разработка стратегии администрирования данных тесно связана с миссией и целями компании. Поэтому стратегический план требует детального анализа целей компании, ее ситуации и потребностей бизнеса. Для руководства разработкой этого плана администрирования данных требуется интегрирующая методология. Наиболее часто используемая методология интеграции известна как информационная инженерия.
- Чтобы помочь преобразовать стратегические планы в оперативные планы, АБД имеет доступ к арсеналу инструментов администрирования БД, включая словарь данных и инструменты автоматизированного проектирования систем (CASE).
- С введением надежных облачных сервисов данных роль АБД вышла за пределы корпоративных стен.

Библиографический список

Алапати С.Р. Oracle Database 11g: Руководство администратора баз данных / Пер. с англ. М.: Вильямс, 2010.

Андон Ф., Резниченко В. Язык запросов SQL: Учебный курс. СПб.: Питер; Киев: Издательская группа BHV, 2006.

Аносов А. Критерии выбора СУБД при создании информационных систем. URL: http://www.interface.ru/home.asp?artId=2147 (24.05.2020).

Бондарь А.Г. Microsoft SQL Server 2014. СПб.: БХВ-Петербург, 2015.

Борри Х. Fireblrd: Руководство разработчика баз данных / Пер. с англ. СПб.: БХВ-Петербург, 2006.

Виейра Р. Программирование баз данных Microsoft SQL Server 2005 для профессионалов / Пер. с англ. М.: Вильямс, 2008.

Голицына О.Л., Максимов Н.В., Попов И.И. Базы данных: Учеб. пособие. М.: Форум; ИНФРА-М, 2012.

Гринвальд Р., Стаковьяк Р., Стерн Дж. Oracle 11g. Основы / Пер. с англ. 4-е изд. СПб.: СимволПлюс, 2009.

Грофф Дж., Вайнберг П., Оппель Э. SQL: Полное руководство / Пер. с англ. 3-е изд. М.: Вильямс. 2015.

Дейт К. Введение в системы баз данных. М.: Наука, 1980. 464 с.

Дейт К.Дж. Введение в системы баз данных / Пер. с англ. 8-е изд. М.: Вильямс, 2005.

Ковязин А.Н., Востриков С.М. Мир InterBase: архитектура, администрирование и разработка приложений баз данных в InterBase/FireBird/Yafill. 4-е изд. М.: КУДИЦ-ОБРАЗ, 2006.

Коннолли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика: учеб. пособие; пер. с англ. 2-е изд. М.: Вильямс, 2000. 1120 с.

Кристофидес Н. Теория графов. Алгоритмический подход / Пер. с англ. М.: Мир, 1978. 432 с.

Логинов Д. Почему мне нравится FireBird. https://clck.ru/9EYVY (09.01.2020).

Мейер М. Теория реляционных баз данных. М.: Мир, 1987. 608 с.

Официальный сайт компании IBase.ru. http://www.ibase.ru (дата обращения 15.05.2020).

Официальный сайт компании Microsoft. http://www.microsoft. com (дата обращения 31.05.2020).

Пери Дж., Пост Дж. Введение в Oracle 10g / Пер. с англ. М.: Вильямс, 2006.

Петкович Д. Microsoft® SQL Server 2012. Руководство для начинающих / Пер. с англ. СПб.: БХВ-Петербург, 2013.

Полякова Л. Основы SQL: электронный курс. http://www.intuit.ru/studies/courses/5/5/info (19.04.2020).

Пржиялковский В. Контекст сеанса в Oracle. https://clck.ru/SxWSg (30.03.2020).

Руководство по языку SQL СУБД FireBird 4.0 / Под ред. Д. Симонова. URL: https://clck.ru/SxWRT (15.06.2020).

Селко Д. Стиль программирования Джо Селко на SQL / Пер. с англ. М.: Русская редакция; СПб.: Питер, 2006.

Словарик по FireBird. http://www firebirdsql.su (18.02.2020).

Ульман Дж. Основы систем баз данных. М.: Финансы и статистика, 1983. 334 с.

Функции по категориям. http://oracleplsql.ru (30.06.2020).

Bonie H. FireBird 3.0 Release Notes, 2013. https://clck.ru/SxbDJ

Chen P.P.-S. The Entity-relationship Model: Toward a Unified View of Data // SIGIR Forum. 1975. Vol. 10, no. 3. Pp. 9–9. https://doi.org/10.1145/1095277.1095279.

Codd E.F. A Relational Model of Data for Large Shared Data Banks // Commun. ACM. 1970. Vol. 13, no. 6. Pp. 377–387. https://doi.org/10.1145/362384.362685.

Fagin R. The Decomposition Versus Synthetic Approach to Relational Database Design // Proceedings of the 3rd International Conference on Very Large Data Bases. Vol. 3. VLDB Endowment, 1977. Pp. 441–446. (VLDB ’77).

Kossmann D., Stocker K. Iterative Dynamic Programming: A New Class of Query Optimization Algorithms // ACM Trans. Database Syst. 2000. Vol. 25, no. 1. Pp. 43–82. DOI: https://doi.org/10.1145/352958.352982.

Lane P. Oracle Database Globalization Support Guide, 11g Release 2 (11.2) / P. Lane. https://clck.ru/Sxcia (17.06.2020).

Lorentz D., Roeser M.B. Oracle Database SQL Language Reference, 11g Release 1 (11.1). https://clck.ru/SxchX (20.05.2020).

Moore S. Oracle Database Express Edition 2 Day Developer's Guide, 11g Release 2 (11.2). https://clck.ru/SxchH (20.05.2020).

Moore S. Oracle Database PL/SQL Language Reference, 11g Release 2 (11.2). https://clck.ru/Sxcvi (12.02.2020).

Murray C. Oracle Database Express Edition 2 Day DBA, 11g Release 2 (11.2). https://clck.ru/Sxcwy (17.06.2020).

Nanda A. Efficient PL/SQL Coding // Arup Nanda. Oracle Database 11g: The Top New Features for DBAs and Developers. https://clck.ru/Sxczr (17.04.2020).

Oracle Pipelined Table Functions. https://clck.ru/Sxd3N (06.06.2020).

The Notions of Consistency and Predicate Locks in a Database System / K.P. Eswaran [et al.] // Commun. ACM. 1976. Vol. 19, no. 11. Pp. 624–633. DOI: https://doi.org/10.1145/360363.360369

Учебное издание

Мамедли Рамиль Эльман оглы

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Учебное пособие

ISBN 978-5-00047-585-0

Редактор *Е.В.Вилявина*
Художник обложки *Д.В.Вилявин*

Изд. лиц. ЛР № 020742. Подписано в печать 25.01.2021
Формат 60×84/8
Гарнитура Times New Roman. Усл. изд. листов 12,43
Электронное издание. Объем 9,77 МБ. Заказ 2173

Издательство НВГУ
628615, Тюменская область, г.Нижневартовск, ул. Маршала Жукова, 4
Тел./факс: (3466) 24-50-51, E-mail: izdatelstvo@nggu.ru