Основные понятия и определения

Термин «база данных» (database) был введен в обиход в области вычислительной техники примерно в 1962 году. Этот термин страдает от обилия различных интерпретаций. Примеры определения термина «баз данных» из авторитетных источников:

База данных (БД) - совокупность данных, организованных по определенным правилам, предусматривающая общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ. Является информационной моделью предметной области.

[К. Дейт, Введение в системы баз данных, 1980]
База данных – совокупность хранимых операционных данных, используемых прикладными системами некоторого предприятия.

[К. Дейт, Введение в системы баз данных, 6-е издание, 2000]
База данных состоит из некоторого набора постоянных данных, которые используются прикладными системами для какого-либо предприятия.

[К. Дейт, Введение в системы баз данных, 7-е издание, 2001]
База данных – это некоторый набор перманентных (постоянных) данных, используемых прикладными системами какого-либо предприятия.

1. Операционные данные – мир OnLine Transaction Processing (Оперативная обработка транзакций) или **OLTP**.
2. Перманентные данные – мир OnLine Analitical Processing (Аналитическая обработка в реальном времени) или **OLAP**. OLAP используется в системах поддержки принятия решений (Decision Support System – DSS) и управленческих информационных системах (Executive Information System – EIS). Базу данных для поддержки принятия решения обычно называют хранилищем данных (data warehouse).

[Джеффри Д. Ульман, Дженнифер Уидом, Введение в системы баз данных, 2000]
По сути дела БД - это просто множество информации, существующее долгое время, часто в течении многих лет. Обычно термином "база данных" обозначается множество данных, управляемое СУБД, или просто СБД.

[Д. Кренке, Теория и практика построения баз данных, 2003]
База данных – это самодокументированное собрание интегрированных записей. Важно понять обе части этого определения.

1. БД является самодокументированной (self describing): она содержит описание собственной структуры. Это описание
   называется словарем данных (data dictionary), каталогом данных (data directory) или метаданными (metadata).

2. БД – это собрание интегрированных записей: она содержит:
   1. Файлы данных,
   2. Метаданные,
   3. Индексы (indexes), которые представляют связи между данными, а также служат для повышения производительности приложений базы данных.
   4. Может содержать метаданные приложений (application metadata).
3. БД является информационной моделью пользовательской модели (user model) предметной области.

База данных - совокупность взаимосвязанных данных некоторой предметной области, хранимых в памяти ЭВМ и организованных таким образом, что эти данные могут быть использованы для решения многих задач многими пользователями. Основные требования к организации данных:

1. Неизбыточность данных - каждое данное присутствует в БД в единственном экземпляре.
2. Совместное использование данных многими пользователями.
3. Эффективность доступа к БД - высокое быстродействие, т. е. малое время отклика на запрос.
4. Целостность данных - соответствие имеющейся в БД информации её внутренней логике, структуре и всем явно заданным правилам.
5. Безопасность данных – защита данных от преднамеренного или непреднамеренного искажения или разрушения данных.
6. Восстановление данных после программных и аппаратных сбоев.
7. Независимость данных от прикладных программ.

Существует множество других определений, отражающих скорее субъективное мнение тех или иных авторов о том, что означает база данных в их понимании, однако общепризнанная единая формулировка отсутствует.

Система управления базами данных (СУБД) - приложение, обеспечивающее создание, хранение, обновление и поиск информации в базах данных.

Система баз данных (или банк данных в отечественной терминологии) - совокупность одной или нескольких баз данных и комплекса информационных, программных и технических средств, обеспечивающих накопление, обновление, корректировку и многоаспектное использование данных в интересах пользователей. (См. Григорьев Ю.А., Ревунков Г.И. Банки данных. – М.: МГТУ, 2002.)

Состав системы баз данных

- Одна или несколько баз данных.
- Аппаратное обеспечение.
- Программное обеспечение.
  - СУБД.
  - Прочие компоненты: утилиты, средства разработки приложений, средства проектирования, генераторы отчетов, диспетчер транзакций и т. д.
- Пользователи.
  - Прикладные программисты, которые отвечают за написание прикладных пакетных и интерактивных программ.
  - Конечные пользователи, которые получают доступ к базе данных, применяя одно из интерактивных приложений, интерфейс, интегрированный в программное обеспечение самой СУБД, командный интерфейс, реализуемый процессором языка запросов или некомандный интерфейс, основанный на меню и формах.
  - Администраторы базы данных.

Основные функции СУБД

- **Непосредственное управление данными во внешней памяти**. В развитых СУБД пользователи не обязаны знать, как организованы файлы данных.
- **Управление буферами оперативной памяти**. Для увеличения скорости обмена данными с внешней памятью используется буферизация данных в оперативной памяти. В развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов, даже если операционная система

производит общесистемную буферизацию. Существует даже отдельное направление СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Это направление основывается на предположении, что объем оперативной памяти компьютеров настолько велик, что можно не беспокоиться о буферизации.

- **Управление транзакциями**. Транзакция – это последовательность операций над данными, рассматриваемая СУБД как единое целое. Реализуется принцип «либо все, либо ничего». Поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД. Но понятие транзакции гораздо более важно в многопользовательских СУБД. То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает удобным использование понятия транзакции как единицы активности пользователя по отношению к БД.
- **Журнализация**. Одним из основных требований к СУБД является требование надежного хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. В любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД. Журнал - это особая часть БД, недоступная пользователям СУБД (иногда поддерживаются две копии журнала, располагаемые на разных физических носителях), в которую поступают записи обо всех изменениях основной части БД.
- **Поддержка языков БД**. Для работы с БД используются специальные языки, называемые языками баз данных. В ранних СУБД (иерархических и сетевых) поддерживалось несколько специализированных по своим функциям языков. В современных СУБД (реляционных) поддерживается язык SQL (Structured Query Language), содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс.

Основные компонентами СУБД

- Ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию.
- Процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных, и создание, как правило, машинно-независимого исполняемого внутреннего кода,
- Подсистема поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД.
- Сервисные программы (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

Классификация СУБД

Классификация СУБД может выполняться по нескольким признакам:

1. По модели данных:
    1. **Дореляционные**
        1. **Инвертированные списки (файлы).** БД на основе инвертированных списков представляет собой совокупность файлов, содержащих записи (таблиц). Для записей в файле определен некоторый порядок, диктуемый физической организацией данных. Для каждого файла может быть определено произвольное число других упорядочений на основании значений некоторых полей записей (инвертированных списков). Обычно для этого используются индексы. В такой модели данных отсутствуют ограничения целостности как таковые. Все

ограничения на возможные экземпляры БД задаются теми программами, которые работают с БД. Одно из немногих ограничений, которое все-таки может присутствовать - это ограничение, задаваемое уникальным индексом.

2. **Иерархичекие**. Иерархическая модель БД состоит из объектов с указателями от родительских объектов к потомкам, соединяя вместе связанную информацию. Иерархические БД могут быть представлены как дерево.
3. **Сетевые**. К основным понятиям сетевой модели БД относятся: элемент (узел), связь. Узел — это совокупность атрибутов данных, описывающих некоторый объект. Сетевые БД могут быть представлены в виде графа. В сетевой БД логика процедуры выборки данных зависит от физической организации этих данных. Поэтому эта модель не является полностью независимой от приложения. Другими словами, если необходимо изменить структуру данных, то нужно изменить и приложение.

2. **Реляционные**. Реляционная модель данных включает следующие компоненты:
   1. Структурный (данные в базе данных представляют собой набор отношений),
   2. Целостностный (отношения (таблицы) отвечают определенным условиям целостности),
   3. Манипуляционный (манипулирования отношениями осуществляется средствами реляционной алгебры и/илиреляционного исчисления). Кроме того, в состав реляционной модели данных включают теорию нормализации. В 1985 году доктор Кодд сформулировал двенадцать правил, которым должна соответствовать настоящая реляционная база данных. Они являются полуофициальным определением понятия реляционная база данных. Строгое изложение теории реляционных баз данных (реляционной модели данных) в современном понимании можно найти в книге К. Дж. Дейта.
3. **Постреляционные**

2. <u>По архитектуре организации хранения данных</u>
   1. локальные СУБД (все части локальной СУБД размещаются на одном компьютере)
   2. распределенные СУБД (части СУБД могут размещаться на двух и более компьютерах)
3. <u>По способу доступа к БД (а может быть по местоположению БД):</u>
   1. **Файл-серверные.** При работе в архитектуре "файл-сервер" БД и приложение расположены на файловом сервере сети. Возможна многопользовательская работа с одной и той же БД, когда каждый пользователь со своего компьютера запускает приложение, расположенное на сетевом сервере. Тогда на компьютере пользователя запускается копия приложения. По каждому запросу к БД из приложения данные из таблиц БД перегоняются на компьютер пользователя, независимо от того, сколько реально нужно данных для выполнения запроса. После этого выполняется запрос. Каждый пользователь имеет на своем компьютере локальную копию данных, время от времени обновляемых из реальной БД, расположенной на сетевом сервере. При этом изменения, которые каждый пользователь вносит в БД, могут быть до определенного момента неизвестны другим пользователям, что делает актуальной задачу систематического обновления данных на компьютере пользователя из реальной базы данных. Другой актуальной задачей является блокирование записей, которые изменяются одним из пользователей; это необходимо для того, чтобы в это время другой пользователь не внес изменений в те же данные. В архитектуре "файл-сервер" вся тяжесть выполнения запросов к базе данных и управления целостностью базы данных ложится на приложение пользователя. База данных на сервере является пассивным источником данных.
   2. **Клиент-серверные.** Клиент-сервер - сетевая архитектура, в которой устройства являются либо клиентами, либо серверами. Клиентом (front end) является запрашивающая машина, сервером (back end) - машина, которая отвечает на запрос. Оба термина (клиент и сервер) могут применяться как к физическим устройствам, так и к программному обеспечению. Характерной особенностью архитектуры "клиент-сервер" является перенос вычислительной нагрузки на сервер базы данных (sql-сервер) и максимальная разгрузка приложения клиента от вычислительной работы, а также существенное укрепление безопасности данных – как от злонамеренных, так и

просто ошибочных изменений. БД в этом случае помещается на сетевом сервере, как и в архитектуре "файл-сервер", однако прямого доступа к базе данных из приложений не происходит. Функцию прямого обращения к базе данных осуществляет СУБД. При этом ресурсы клиентского компьютера не участвуют в физическом выполнении запроса; клиентский компьютер лишь отсылает запрос к СУБД и получает результат, после чего интерпретирует его необходимым образом и представляет пользователю. Так как клиентскому приложению посылается результат выполнения запроса, по сети "путешествуют" только те данные, которые необходимы клиенту. В итоге снижается нагрузка на сеть. Кроме того, СУБД, если это возможно, оптимизирует полученный запрос таким образом, чтобы он был выполнен в минимальное время с наименьшими накладными расходами. При выполнении запросов сервером существенно повышается степень безопасности данных, поскольку правила целостности данных определяются в базе данных на сервере и являются едиными для всех приложений, использующих эту БД. Таким образом, исключается возможность определения противоречивых правил поддержания целостности. Аппарат транзакций, поддерживаемый СУБД, позволяет исключить одновременное изменение одних и тех же данных различными пользователями и предоставляет возможность откатов к первоначальным значениям при внесении в БД изменений, закончившихся аварийно.

3. **Встраиваемые**. Встраиваемая СУБД — библиотека, которая позволяет унифицированным образом хранить большие объёмы данных на локальной машине. Доступ к данным может происходить через SQL либо через особые функции СУБД. Встраиваемые СУБД быстрее обычных клиент-серверных и не требуют установки сервера, поэтому востребованы в локальном ПО, которое имеет дело с большими объёмами данных.
4. **Сервисно-ориентированные**. БД является хранилищем сообщений, промежуточных состояний, метаинформации об очередях сообщений и сервисах. Отправка сообщений в очередь и прием сообщений из очереди производится в одной транзакции с изменением данных, что обеспечивает транзакционную целостность системы. Так как очереди сообщений и данные хранятся и обрабатываются в базе единообразно, это обеспечивает гарантированную доставку и обработку сообщений в случае сбоев оборудования или питания с таким же успехом, как и прочих данных, хранящихся в той же базе данных. Кроме этого, в базе данных хранится информация о самих сервисах и обрабатываемых ими очередях сообщений, что обеспечивает восстановление после сбоя состояний не только данных и сообщений, но и настроек сервисов и очередей сообщений.
5. **Прочие**. Пространственная(spatial database), Временная, или темпоральная (temporal database), Пространственно-временная (spatial-temporal database).

Семантическое моделирование данных

Реляционная модель данных достаточна для моделирования предметных областей. Однако проявляется ограниченность реляционной модели данных в следующих аспектах:

- Модель не предоставляет достаточных средств для представления смысла данных.
- Для многих приложений трудно моделировать предметную область на основе плоских таблиц.
- Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет каких-либо средств для представления этих зависимостей.
- Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области («сущностей») и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для разделения сущностей и связей.

Указанные ограничения вызвали к жизни направление семантических (концептуальных, инфологических) моделей данных. Любая развитая семантическая модель данных, как и реляционная модель, включает структурную, манипуляционную и целостную части. Главным назначением семантических моделей является обеспечение возможности выражения семантики данных. На практике семантическое моделирование используется на первой стадии проектирования базы данных. При этом в терминах семантической модели производится концептуальная схема базы данных, которая затем

- Либо вручную преобразуется к реляционной (или какой-либо другой) схеме.
- Либо реализуется автоматизированная компиляция концептуальной схемы в реляционную.
- Либо происходит работа с базой данных в семантической модели, т.е. под управлением СУБД, основанных на семантических моделях данных. (Третья возможность еще не вышла за пределы исследовательских и экспериментальных проектов.)

Наиболее известным представителем класса семантических моделей предметной области является модель «сущность- связь» или ER-модель, предложенная Питером Ченомв 1976 году. Модель сущность-связь — модель данных, позволяющая описывать концептуальные схемы предметной области. Предметная область — часть реального мира, рассматриваемая в пределах данного контекста. Под контекстом здесь может пониматься, например, область исследования или область, которая является объектом некоторой деятельности. ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями. Во время проектирования баз данных происходит преобразование ER-модели в конкретную схему базы данных на основе выбранной модели данных (реляционной, объектной, сетевой или др.). ER-модель представляет собой формальную конструкцию, которая сама по себе не предписывает никаких графических средств её визуализации. В качестве стандартной графической нотации, с помощью которой можно визуализировать ERM, была предложена диаграмма сущность-связь (entity-relationship diagram, ERD). На практике понятия ER-модель и ER-диаграмма часто не различают, хотя для визуализации ER-моделей предложены и другие графические нотации. Основными понятиями ER-модели являются сущность, связь и атрибут (свойство).

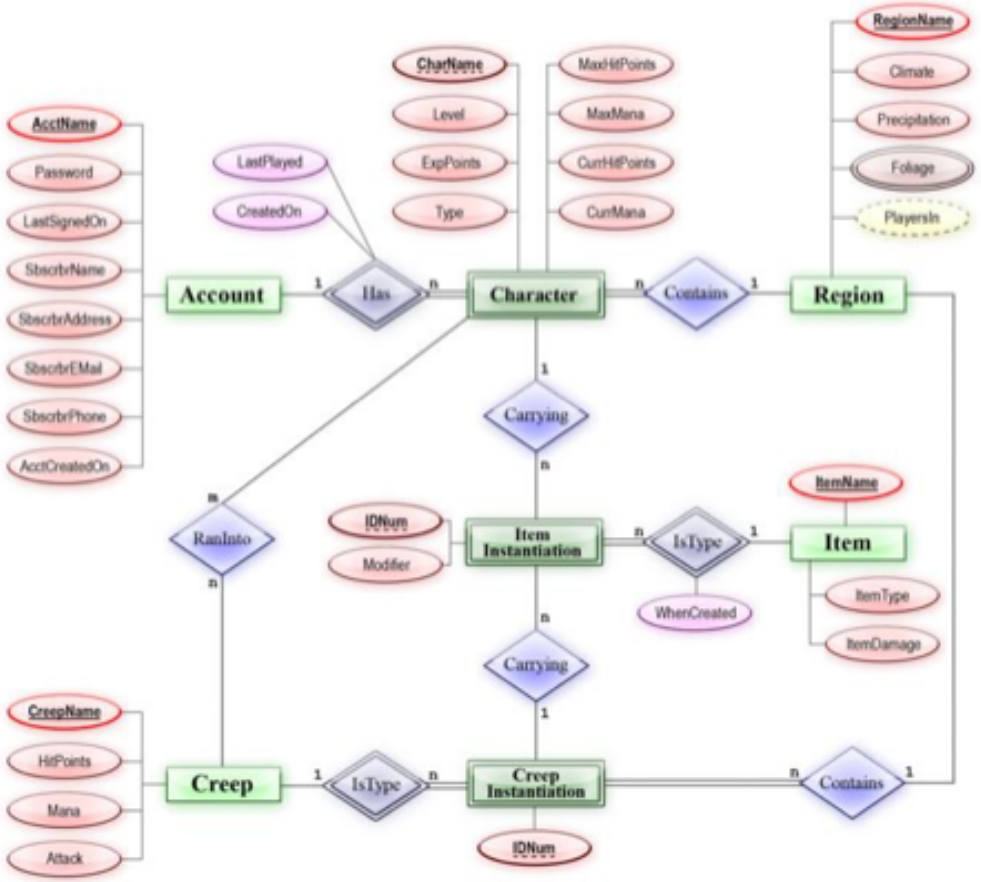
Сущность - это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности - это имя типа, а не некоторого конкретного экземпляра этого типа. Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных объектов этого типа. Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности (это требование в некотором роде аналогично требованию отсутствия кортежей-

дубликатов в реляционных таблицах). Сущности подразделяются на сильные и слабые. Сильные сущности существуют сами по себе, а существование слабых сущностей зависит от существования сильных.

Связь - это ассоциация, устанавливаемая между сущностями. Эта ассоциация может существовать между разными сущностями или между сущностью и ей же самой (рекурсивная связь). Сущности, включенные в связь, называются ее участниками, а количество участников связи называется ее степенью. Участие сущности в связи может быть как полным, так и частичным. Связи в ER-модели могут иметь тип «один к одному», «один ко многим», «многие ко многим». Именно тип связи «многие ко многим» является единственным типом, представляющим истинную связь, поскольку это единственным тип связи, который требует для своего представления отдельного отношения. Связи типа «один к одному» и «один ко многим» всегда могут быть представлены с помощью механизма внешнего ключа, помещаемого в одно из отношений.

Свойством сущности (и связи) является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности (или связи). Значения свойств каждого типа извлекаются из соответствующего множества значений, которое в реляционной терминологии называется доменом. Свойства могут быть простыми или составными, ключевыми, однозначными или многозначными, опущенными (т. е. «неизвестными» или «непредставленными»), базовыми или производными.

Более сложными элементами ER-модели являются подтипы и супертипы сущностей. Как в языках программирования с развитыми типовыми системами (например, в языках объектно-ориентированного программирования), вводится возможность наследования типа сущности, исходя из одного или нескольких супертипов.



На ER-диаграммах множества сущностей изображаются в виде прямоугольников, множества отношений изображаются в виде ромбов. Слабый тип сущности изображают в виде прямоугольника с двойным контуром. Слабый тип связи изображают в виде ромба с двойным контуром. Если сущность участвует в отношении, они связаны линией. Тип связи с частичным участием изображают двойной линией. Вид типа связи обозначается над линиями в виде соответствующих надписей возле типов сущностей. Например, если это вид бинарной связи «один ко многим», то делают надписи 1, n (или m), соответственно, возле соответствующих типов сущностей. Атрибуты изображаются в виде овалов и связываются линией с одним отношением или с одной сущностью. Именование сущности обычно выражается уникальным существительным, именование связи обычно выражается глаголом, именование атрибута обычно выражается существительным. Неизбыточный набор атрибутов, значения которых в совокупности являются уникальными для каждого экземпляра сущности, являются ключом сущности.

Существует множество инструментов для работы с ER-моделями, вот некоторые из них: Microsoft Visio, ERwin, Oracle Designer, PowerDesigner, Rational Rose. В справочниках приводятся сведения о 25 таких инструментах.

Получение реляционной схемы из ER-схемы осуществляется с помощью следующей пошаговой процедуры.

Шаг 1. Каждая простая сущность превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

Шаг 2. Каждый свойство (атрибут) становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут.

Шаг 3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификаторов, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

Шаг 4. Связи «многие к одному» (и «один к одному») становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи «один», и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.

Шаг 5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Шаг 6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа:

- все подтипы в одной таблице,
- для каждого подтипа - отдельная таблица.

Реляционная модель данных

Основоположником теории реляционных баз данных является британский учёный Эдгар Кодд, который в 1970 году опубликовал первую работу по реляционной модели данных. Наиболее распространенная трактовка реляционной модели данных принадлежит Кристоферу Дейту. Согласно Дейту, реляционная модель состоит из трех частей: структурной, целостностной и манипуляционной.

Структурная часть реляционной модели

Структурная часть реляционной модели описывает, из каких объектов состоит реляционная модель. Постулируется, что основной структурой данных, используемой в реляционной модели, являются нормализованные «n-арные» отношения. Основными понятиями структурной части реляционной модели являются тип данных, домен, атрибут, схема отношения, схема базы данных, кортеж, отношение, потенциальный, первичный и альтернативные ключи, реляционная база данных.

Понятие типа данных в реляционной модели полностью адекватно понятию типа данных в языках программирования.

Понятие домена можно считать уточнением типа данных. Домен можно рассматривать как подмножество значений некоторого типа данных, имеющих определенный смысл. Домен характеризуется следующими свойствами:

- домен имеет уникальное имя (в пределах базы данных),
- домен определен на некотором типе данных или на другом домене,
- домен может иметь некоторое логическое условие, позволяющее описать подмножество данных, допустимых для данного домена,
- домен несет определенную смысловую нагрузку.

Говорят, что домен отражает семантику, определенную предметной областью. Может быть несколько доменов, совпадающих как подмножества, но несущие различный смысл. Основное значение доменов состоит в том, что домены ограничивают сравнения. Некорректно, с логической точки зрения, сравнивать значения из различных доменов, даже если они имеют одинаковый тип.
Понятие домена помогает правильно моделировать предметную область. Не все домены обладают логическим условием, ограничивающим возможные значения домена. В таком случае множество возможных значений домена совпадает с множеством возможных значений типа данных.
Атрибут отношения – это пара вида <имя_атрибута, имя_домена >. Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.
Схема отношения – это именованное множество упорядоченных пар <имя_атрибута, имя_домена>. Степенью или «арностью» схемы отношения является мощность этого множества. Схема базы данных в реляционной модели – это множество именованных схем отношений. Понятие схемы отношения близко к понятию структурного типа в языках программирования (например, record в языке Pascal или struct в языке C).
Кортеж, соответствующий данной схеме отношения – это множество упорядоченных пар <имя_атрибута, значение_атрибута>, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. Значение атрибута должно быть допустимым значением домена, на котором определен данный атрибут. Степень или «арность» кортежа совпадает с «арностью» соответствующей схемы отношения.
Отношение, определенное на множестве из n доменов (не обязательно различных), содержит

две части: заголовок (схему отношения) и тело (множество из m кортежей). Значения n и m называются соответственно степенью и кардинальностью отношения. Отношения обладают следующими свойствами.

- В отношении нет одинаковых кортежей. Действительно, тело отношения есть множество кортежей и, как всякое множество, не может содержать неразличимые элементы.
- Кортежи не упорядочены (сверху вниз). Причина следующая – тело отношения есть множество, а множество не упорядочено.
- Атрибуты не упорядочены (слева направо). Т.к. каждый атрибут имеет уникальное имя в пределах отношения, то порядок атрибутов не имеет значения. В таблицах в отличие от отношений столбцы упорядочены.
- Каждый кортеж содержит ровно одно значение для каждого атрибута. Отношение, удовлетворяющее этому свойству, называется нормализованным или представленным в первой нормальной форме (1NF).
- Все значения атрибутов атомарны, т. е. не обладают структурой. Трактовка этого свойства в последнее время претерпела существенные изменения. Исторически в большинстве публикаций по базам данных считалось недопустимым использовать атрибуты со структурированными значениями. (А в большинстве изданий это считается таковым и поныне.) В настоящее время принимается следующая точка зрения: тип данных атрибута может быть произвольным, а, следовательно, возможно существование отношения с атрибутами, значениями которых также являются отношения. Именно так в некоторых постреляционных базах данных реализована работа со сколь угодно сложными типами данных, создаваемых пользователями.

В реляционной модели каждый кортеж любого отношения должен отличатся от любого другого кортежа этого отношения (т.е. любое отношение должно обладать уникальным ключом).

Непустое подмножество множества атрибутов схемы отношения будет потенциальным ключом тогда и только тогда, когда оно будет обладать свойствами уникальности (в отношении нет двух различных кортежей с одинаковыми значениями потенциального ключа) и неизбыточности (никакое из собственных подмножеств множества потенциального ключа не обладает свойством уникальности).

В реляционной модели по традиции один из потенциальных ключей должен быть выбран в качестве первичного ключа, а все остальные потенциальные ключи будут называться альтернативными.

Реляционная база данных – это набор отношений, имена которых совпадают с именами схем отношений в схеме базы данных.

Целостностная часть реляционной модели

В целостностной части реляционной модели фиксируются два базовых требования целостности, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и ссылочная целостность (или целостность внешних ключей).

Простой объект реального мира представляется в реляционной модели как кортеж некоторого отношения. Требование целостности сущностей заключается в следующем: каждый кортеж любого отношения должен отличатся от любого другого кортежа этого отношения (т.е. любое отношение должно обладать потенциальным ключом). Вполне очевидно, что если данное требование не соблюдается (т.е. кортежи в рамках одного отношения не уникальны), то в базе данных может храниться противоречивая информация об одном и том же объекте.

Поддержание целостности сущностей обеспечивается средствами СУБД. Это осуществляется с помощью двух ограничений:

- при добавлении записей в таблицу проверяется уникальность их первичных ключей,
- не позволяется изменение значений атрибутов, входящих в первичный ключ.

Сложные объекты реального мира представляются в реляционной модели данных в виде кортежей нескольких нормализованных отношений, связанных между собой. При этом

- связи между данными отношениями описываются в терминах функциональных зависимостей,
- для отражения функциональных зависимостей между кортежами разных отношений используется дублирование первичного ключа одного отношения (родительского) в другое (дочернее). Атрибуты, представляющие собой копии ключей родительских отношений, называются внешними ключами.

Внешний ключ в отношении R2 – это непустое подмножество множества атрибутов FK этого отношения, такое, что: a) существует отношение R1 (причем отношения R1 и R2 необязательно различны) с потенциальным ключом CK; b) каждое значение внешнего ключа FK в текущем значении отношения R2 обязательно совпадает со значением ключа CK некоторого кортежа в текущем значении отношения R1.

Требование ссылочной целостности состоит в следующем: Для каждого значения внешнего ключа, появляющегося в дочернем отношении, в родительском отношении должен найтись кортеж с таким же значением первичного ключа.

Как правило, поддержание ссылочной целостности также возлагается на СУБД. Например, она может не позволить пользователю добавить запись, содержащую внешний ключ с несуществующим (неопределенным) значением.

Манипуляционная часть реляционной модели

Манипуляционная часть реляционной модели описывает два эквивалентных способа манипулирования реляционными данными – реляционную алгебру и реляционное исчисление. Принципиальное различие между реляционной алгеброй и реляционным исчислением заключается в следующем: реляционная алгебра в явном виде предоставляет набор операций, а реляционное исчисление представляет систему обозначений для определения требуемого отношения в терминах данных отношений. Формулировка запроса в терминах реляционной алгебры носит предписывающий характер, а в терминах реляционного исчисления – описательный характер. Говоря неформально, реляционная алгебра носит процедурный характер (пусть на очень высоком уровне), а реляционное исчисление – непроцедурный характер.

Реляционная алгебра

Реляционная алгебра является основным компонентом реляционной модели, опубликованной Коддом, и состоит из восьми операторов, составляющих две группы по четыре оператора:

- Традиционные операции над множествами: объединение (UNION), пересечение (INTERSECT), разность (MINUS) и декартово произведение (TIMES). Все операции модифицированы, с учетом того, что их операндами являются отношения, а не произвольные множества.
- Специальные реляционные операции: ограничение (WHERE) , проекция (PROJECT), соединение (JOIN) и деление (DIVIDE BY).

Результат выполнения любой операции реляционной алгебры над отношениями также является отношением. Эта особенность называется свойством реляционной замкнутости. Утверждается, что поскольку реляционная алгебра является замкнутой, то в реляционных выражениях можно использовать вложенные выражения сколь угодно сложной структуры. Если рассматривать свойство реляционной замкнутости строго, то каждая реляционная операция должна быть определена таким образом, чтобы выдавать результат с надлежащим типом отношения (в частности, с соответствующим набором атрибутов или заголовком). Для достижения этой цели вводится новый оператор переименование (RENAME), предназначенный для переименования атрибутов в определенном отношении.

В качестве основы для последующих обсуждений рассмотрим упрощенный синтаксис выражений реляционной алгебры в форме БНФ.

```
реляционное_выражение ::= унарное_выражение | бинарное_выражение
унарное_выражение ::= переименование | ограничение | проекция
переименование ::= терм RENAME имя_атрибута AS имя_атрибута
терм ::= имя_отношения | ( реляционное_выражение )
ограничение ::= терм WHERE логическое_выражение
проекция ::= терм | терм [ список_имен_атрибутов ]
бинарное_выражение ::= проекция бинарная_операция реляционное_выражение
бинарная_операция :: = UNION | INTERSECT | MINUS | TIMES | JOIN | DIVIDEBY
```

По приведенной грамматике можно сделать следующие замечания.

- Реляционные операторы UNION, INTERSECT и MINUS требуют, чтобы отношения были совместимыми по типу, т. е. имели идентичные заголовки.
- Реляционные операторы UNION, INTERSECT, TIMES и JOIN ассоциативны и коммутативны.
- Если отношения A и B не имеют общих атрибутов, то операция соединения A JOIN B эквивалентна операции A
  TIMES B, т. е. в таком случае соединение вырождается в декартово произведение. Такое соединение называют естественным.
- Другой допустимый синтаксис для синтаксической категории переименования таков: (терм RENAME список- переименований). Здесь каждый из элементов списка переименований представляет собой выражение имя_атрибута AS имя_атрибута.
- Несмотря на большие возможности, предоставляемые операторами реляционной алгебры, существует несколько типов запросов, которые нельзя выразить этими средствами. Для таких случаев необходимо использовать процедурные расширения реляционных языков.

В алгебре Кодда не все операторы являются независимыми, т. е. некоторые из реляционных операторов могут быть выражены через другие реляционные операторы.

Оператор естественного соединения по атрибуту Y определяется через оператор декартового произведения и оператор ограничения:

A JOIN B = ((A TIMES (B RENAME Y AS Y1)) WHERE Y=Y1)[X, Y, Z]

Оператор пересечения выражается через вычитание следующим образом:

A INTERSECT B = A MINUS (A MINUS B)

Оператор деления выражается через операторы вычитания, декартового произведения и проекции следующим образом:

A DIVIDEBY B = A[X] MINUS ((A[X] TIMES B) MINUS A)[X]

Оставшиеся реляционные операторы (объединение, вычитание, декартово произведение, ограничение, проекция) являются примитивными операторами – их нельзя выразить друг через друга.

В качестве примера рассмотрим запросы на языке реляционной алгебры для схемы базы данных «Поставщики и детали», представленной следующими схемами отношений:

S(Sno: integer, Sname: string, Status: integer, City: string)
P(Pno: integer, Pname: string, Color: string, Weight: real, City: string)
SP(Sno: integer, Pno: integer, Qty: integer)

В данном примере имена доменов представлены именами типов, имена типов отделяются от имен атрибутов двоеточием, первичные ключи выделены подчеркиванием, а имена внешних ключей схемы отношения SP (ПОСТАВКА) совпадают с именами первичных ключей схем отношений S (ПОСТАВЩИК) и P (ДЕТАЛЬ).

Получить имена поставщиков, которые поставляют деталь под номером 2	((SP JOIN S) WHEPE Pno = 2) [Sname]
Получить имена поставщиков, которые поставляют по крайней мере одну красную деталь	(((P WHERE Color = 'Красный') JOIN SP) [Sno] JOIN S) [Sname] Другая формулировка того же запроса: (((P WHERE Color = 'Красный') [Pno] JOIN SP) JOIN S) [Sname] Этот пример подчеркивает одно важное обстоятельство: возможность сформулировать один и тот же запрос несколькими способами.
Получить имена поставщиков, которые поставляют все детали	((SP [Sno, Pno] DIVIDE BY P [Pno] JOIN S) [Sname]

Получить номера поставщиков, поставляющих по крайней мере все те детали, которые поставляет поставщик под номером 2	SP [Sno, Pno] DIVIDE BY (SP WHEPE Sno = 2) [Pno]
Получить все пары номеров поставщиков, размещенных в одном городе	(((S RENAME Sno AS FirstSno) [FirstSno, City] JOIN (S RENAME Sno AS SecondSno) [SecondSno , City]) WHEPE FirstSno < SecondSno) [FirstSno, SecondSno]
Получить имена поставщиков, которые не поставляют деталь под номером 2	((S[Sno] MINUS (SP WHEPE Pno = 2) [Sno]) JOIN S) [Sname]

Вычислительные возможности реляционной алгебры можно увеличить путем введения дополнительных операторов. Дополнительные операторы реляционной алгебры

Многие авторы предлагали новые алгебраические операторы после определения Коддом первоначальных восьми. Рассмотрим несколько таких операторов:

SEMIJOIN (полусоединение), SEMIMINUS (полувычитание), EXTEND (расширение), SUMMARIZE (обобщение) и TCLOSE (транзитивное замыкание).

Синтаксис этих операторов выглядит следующим образом:

<полусоединение> ::= <реляционное выражение> SEMIJOIN <реляционное выражение>
< полувычитание > ::= <реляционное выражение> SEMIMINUS <реляционное выражение>
< расширение> ::= EXTEND <реляционное выражение> ADD
<список добавляемых расширений>
<добавляемое расширение> ::= <выражение> AS <имя атрибута>
<обобщение> ::= SUMMARIZE <реляционное выражение> PER <реляционное выражение> ADD <список добавляемых обобщений>
<добавляемое обобщение> ::= <тип обобщения> [(<скалярное выражение>)] AS <имя атрибута> <тип обобщения> ::= COUNT | SUM | AVG | MAX | MIN | ALL | ANY | COUNTD | SUMD | AVGD <транзитивное замыкание> ::= TCLOSE <реляционное выражение>

Операция расширения

С помощью операции расширения из определенного отношения (по крайней мере, концептуально) создается новое отношение, которое содержит дополнительный атрибут, значения которого получены посредством некоторых скалярных вычислений.

EXTEND S ADD 'Supplier' AS TAG
EXTEND P ADD (Weight * 454) AS GMWT
(EXTEND P ADD (Weight * 454) AS GMWT) WHERE GMWT > Weight (10000.0)) { ALL BUT GMWT } EXTEND (P JOIN SP) ADD (Weight * Qty) AS SHIPWT
(EXTEND S ADD City AS SCity) { ALL BUT City }
EXTEND P ADD Weight * 454 AS GMW, Weight * 16 AS OZWT)
EXTEND S ADD COUNT ((SP RENAME SNo AS X) WHERE X = SNo) AS NP

Рассмотрим вкратце обобщающие функции. Общее назначение этих функций состоит в том, чтобы на основе значений некоторого атрибута определенного отношения получить скалярное значение. В языке Tutorial D параметр <вызов обобщающей функции> является особым случаем параметра <скалярное выражение> и в общем случае имеет следующий вид.

<имя функции> (<реляционное выражение> [, <имя атрибута>])
Если параметр <имя функции> имеет значение COUNT, то параметр <имя атрибута> недопустим и должен быть опущен. В остальных случаях параметр <имя атрибута> может быть опущен тогда и только тогда, когда параметр <реляционное выражение> задает отношение со степенью, равной единице.

Операция обобщения

В реляционной алгебре операция расширения позволяет выполнять "горизонтальные" вычисления в отношении отдельных строк. Оператор обобщения выполняет аналогичную функцию для "вертикальных" вычислений в отношении отдельного столбца. Примеры.

SUMMARIZE SP PER SP { PNo } ADD SUM (Qty) AS TOTQTY

В результате его вычисления создается отношение с заголовком {P#, TOTQT Y}, содержащее один кортеж для каждого значения атрибута P# в проекции SP{P#}. Каждый из этих кортежей содержит значение атрибута P# и соответствующее общее количество деталей. Другими словами, концептуально исходное отношение P «перегруппировано» в множество групп кортежей (по одной группе для каждого уникального значения атрибута P#), после чего для каждой полученной группы сгенерирован один кортеж, помещаемый в окончательный результат.

В общем случае выражение выглядит: SUMMARIZE A PER B ADD <обобщение> AS

Определяется следующим образом:

- Отношение B должно иметь такой же тип, как и некоторая проекция отношения A, T.e. каждый атрибут отношения B должен одновременно присутствовать в отношении A. Примем, что атрибутами этой проекции (или, что эквивалентно, атрибутами отношения B) являются атрибуты A1, A2, ... , An.
- Результатом вычисления данного выражения будет отношение с заголовком {Al, A2, ... , An, Z}, где Z является новым добавленным атрибутом.
- Тело результата содержит все кортежи t, где t является кортежем отношения B, расширенным значением нового атрибута Z. Это значение нового атрибута Z подсчитывается посредством вычисления обобщающего выражения по всем кортежам отношения A, которое имеет те же значения для атрибутов A1, A2, ... , An, что и кортеж t. (Разумеется, если в отношении A нет кортежей, принимающих те же значения для атрибутов A1, A2, .. . , An, что и кортеж t, то обобщающее выражение будет вычислено для пустого множества.) Отношение B не должно со- держать атрибут с именем Z, а обобщающее выражение не должно ссылаться на атрибут Z. Заметьте, что кардинальность результата равна кардинальности отношения B, а степень результата равна степени отношения B плюс единица. Типом переменной Z в этом случае будет тип обобщающего выражения.

Вот еще один пример:
SUMMARIZE (P JOIN SP) PER P { City } ADD COUNT AS NSP

Легко заметить, что оператор SUMMARIZE не примитивен – его можно моделировать с помощью оператора EXTEND. Рассмотрим следующее выражение

SUMMARIZE SP PER S { SNo } ADD COUNT AS NP
По сути, это сокращенная запись представленного ниже более сложного выражения
{ EXTEND S { Sno } ADD ((SP RENAME Sno AS X) WHERE X=Sno) AS Y, COUNT (Y) AS NP) { Sno, NP }

Группирование и разгруппирование

Поскольку значениями атрибутов отношений могут быть другие отношения, было бы желательным наличие дополнительных реляционных операторов, называемых операторами группирования и разгруппирования. Рассмотрим пример SP GROUP (PNo, Qty) AS PQ который можно прочесть как «сгруппировать отношение SP по атрибуту SNo», поскольку атрибут SNo является единственным атрибутом отношения SP, не упомянутым в предложении GROUP. В результате получится отношение, заголовок которого выглядит так

{ SNo SNo, PQ RELATION { PNo PNo, Qty Qty } }

Другими словами, он состоит из атрибута PQ, принимающего в качестве значений отношения (PQ, в свою очередь, имеет атрибуты PNo и Qty), а также из всех остальных атрибутов отношения SP (в нашем случае "все остальные атрибуты" - это атрибут SNo). Тело этого отношения содержит ровно по одному кортежу для всех различных значений атрибута SNo исходного отношения SP.

Перейдем теперь к операций разгруппирования. Пусть SPQ - это отношение, полученное в результате группирования. Тогда выражение SPQ UNGROUP PQ возвращает нас к отношению SP (как и следовало ожидать). Точнее, оно выдает в качестве результата отношение, заголовок которого выглядит так (SNo SNo, PNo PNo, Qty Qty)

Реляционные сравнения

Реляционная алгебра в том виде, в котором она была изначально определена, не поддерживает прямого сравнения двух отношений (например, проверки их равенства или того, является ли одно из них подмножеством другого). Это упущение легко исправляется следующим образом. Сначала определяется новый вид условия - реляционное сравнение - со следующим синтаксисом.

<реляционное выражение> <отношение> <реляционное выражение> <отношение> ::=
> -- Собственное супермножество | >= -- Супермножество
| < -- Собственное подмножество | <= -- Подмножество
| = -- Равно
| <> -- Не равно
Здесь параметр <реляционное выражение> в обоих случаях выражения реляционной алгебры, представляющие совместимые по типу отношения.

S (City) = P (City)

Смысл выражения: совпадает ли проекция отношения поставщиков S по атрибуту City с проекцией отношения деталей P по атрибуту City? S (SNo) > SPJ (SNo)

Смысл выражения: есть ли поставщики, вообще не поставляющие деталей?
На практике часто требуется определить, является ли данное отношение пустым. Соответствующий оператор, возвращающий логическое значение имеет вид

IS_EMPTY (<реляционное выражение>)
Не менее часто требуется проверить, присутствует ли данный кортеж t в данном отношении R. Для этой цели подойдет следующее реляционное сравнение.

RELATION { t } <= R
Однако, с точки зрения пользователя, удобнее применять следующее сокращение: t IN R

Реляционное исчисление

Реляционное исчисление основано на разделе математической логики, который называется исчислением предикатов. Реляционное исчисление существует в двух формах: исчисление кортежей и исчисление доменов. Основное различие между ними состоит в том, что переменные исчисления кортежей являются переменными кортежей (они изменяются на отношении, а их значения являются кортежами), в то время как переменные исчисления доменов являются переменными доменов (они изменяются на доменах, а их значения являются скалярами).

Исчисление кортежей

Реляционное исчисление является альтернативой реляционной алгебре. Внешне два подхода очень отличаются – исчисление описательное, а алгебра предписывающая, но на более низком уровне они представляют собой одно и то же, поскольку любые выражения исчисления могут быть преобразованы в семантически эквивалентные выражения в алгебре и наоборот.

Исчисление существует в двух формах: исчисление кортежей и исчисление доменов. Основное различие между ними состоит в том, что переменные исчисления кортежей являются переменными кортежей (они изменяются на отношении, а их значения являются кортежами), в то время как переменные исчисления доменов являются переменными доменов (они изменяются на доменах, а их значения являются скалярами; в этом смысле, действительно, "переменная домена" - не очень точный термин).

Выражение исчисления кортежей содержит заключенный в скобки список целевых элементов и выражение WHERE, содержащее формулу WFF ("правильно построенную формулу"). Такая формула WFF составляется из кванторов (EXISTS и FORALL), свободных и связанных переменных, литералов, операторов сравнения, логических (булевых) опе- раторов и скобок. Каждая свободная переменная, которая встречается в формуле WFF, должна быть также перечислена в списке целевых элементов.

Упрощенный синтаксис выражений исчисления кортежей в форме БНФ имеет вид:

```
объявление-кортежной-переменной ::= RANGE OF переменная IS список-областей
область ::= отношение | реляционное-выражение
реляционное-выражение ::= (список-целевых-элементов)[WHERE wff]
целевой-элемент ::= переменная | переменная.атрибут [AS атрибут]
wff ::= условие | NOT wff | условие AND wff | условие OR wff | IF условие THEN wff |
        EXISTS переменная (wff) | FORALL переменная (wff) | (wff)
условие ::= (wff) | компаранд операция-отношения компаранд
```

По приведенной грамматике можно сделать следующие замечания.

- Квадратные скобки здесь указывают на компоненты, которые по умолчанию могут быть опущены.
- Категории отношение, атрибут и переменная – это идентификаторы (т. е. имена).
- Реляционное выражение содержит заключенный в скобки список целевых элементов и выражение WHERE, содержащее формулу wff («правильно построенную формулу»). Такая формула wff составляется из кванторов (EXISTS и FORALL), свободных и связанных переменных, констант, операторов сравнения, логических (булевых)

операторов и скобок. Каждая свободная переменная, которая встречается в формуле wff, должна быть также перечислена в списке целевых элементов.
- Категория условие представляет или формулу wff, заключенную в скобки, или простое скалярное сравнение, где каждый компаранд оператора сравнения – это либо скалярная константа, либо значение атрибута в форме переменная.атрибут.

Пусть кортежная переменная T определяются следующим образом:

RANGE OF T IS R1, R2, ..., Rn

Тогда отношения R1, R2, ..., Rn должны быть совместимы по типу т. е. они должны иметь идентичные заголовки, и кортежная переменная T изменяется на объединении этих отношений, т. е. её значение в любое заданное время будет некоторым текущим кортежем, по крайней мере одного из этих отношений.

Примеры объявлений кортежных переменных.

RANGE OF SX IS S RANGE OF SPX IS SP RANGE OF SY IS

(SX) WHERE SX.City = 'Смоленск',
(SX) WHERE EXISTS SPX (SPX.Sno = SX.Sno AND SPX.Pno = 1)

Здесь переменная кортежа SY может принимать значения из множества кортежей S для поставщиков, которые или размещены в Смоленске, или поставляют деталь под номером 1, или и то и другое.

Для сравнения с реляционной алгеброй рассмотрим некоторые запросы на языке исчисления кортежей, которые соответствуют рассмотренным ранее.

Для сравнения с реляционной алгеброй некоторые примеры соответствуют рассмотренным ранее. Любые примеры можно расширить, включив в них завершающий шаг присвоения, присвоив значение выражения некоторому именованному отношению; этот шаг для краткости опускается.

Получить номера поставщиков из Смоленска со статусом больше 20	(SX.Sno) WHERE SX.City = 'Смоленск' AND SX.Status > 20
Получить все такие пары номеров поставщиков, что два поставщика размещаются в одном городе	(SX.Sno AS FirstSno, SY.Sno AS SecondSno) WHERE SX.City = SY.City AND SX.Sno < SY.Sno Замечание. Спецификации «AS FirstSno» и «AS SecondSno» дают имена атрибутам результата; следовательно, такие имена недоступны для использования во фразе WHERE и потому второе сравнение во фразе WHERE «SX.Sno < SY.Sno», а не «FirstSno < SecondSno».
Получить имена поставщиков, которые поставляют деталь с номером 2	SX.Sname WHERE EXISTS SPX (SPX.Sno = SX.Sno AND SPX.Pno = 2)

Получить имена поставщиков, которые поставляют по крайней мере одну красную деталь	SX.Sname WHERE EXISTS SPX (SX.Sno = SPX.Sno AND EXISTS PX (PX.Pno = SPX.Pno AND PX.Color = 'Красный')) Или эквивалентная формула (но в предваренной нормальной форме, в которой все кванторы записываются в начале формулы WFF): SX.Sname WHERE EXISTS SPX (EXISTS PX (SX.Sno = SPX.Sno AND SPX. Pno = PX.Pno AND PX.Color = 'Красный'))
Получить имена поставщиков, которые поставляют по крайней мере одну деталь, поставляемую поставщиком под номером 2	SX.Sname WHERE EXISTS SPX (EXISTS SPY (SX.Sno = SPX.Sno AND SPX.Pno = SPX.Pno AND SPY.sno = 2))
Получить имена поставщиков, которые поставляют все детали	SX.SNAME WHERE FORALL PX (EXISTS SPX (SPX.Sno = SX.Sno AND SPX.Pno = PX.Pno)) Или равносильное выражение без использования квантора FORALL: SX. SNAME WHERE NOT EXISTS PX (NOT EXISTS SPX (SPX.Sno = SX.Sno AND SPX.Pno = PX.Pno)) Отметим, что содержание этого запроса – «имена поставщиков, которые поставляют все детали» - является точным на 100% (в отличие от случая с алгебраическим аналогом при использовании оператора DIVIDEBY).
Получить имена поставщиков, которые не поставляют деталь под номером 2	SX.Sname WHERE NOT EXISTS SPX (SPX.Sno = SX.Sno AND SPX.Pno = 2)
Получить номера поставщиков, которые поставляют по крайней мере все детали, поставляемые поставщиком с номером 2	SX.Sno WHERE FORALL SPY (SPY.Sno <> 2 OR EXISTS SPZ (SPZ.Sno = SX.Sno AND SPZ.Pno = SPY.Pno))

Предваренная нормальная форма не является более или менее правильной по сравнению с другими формами, но немного попрактиковавшись можно убедиться, что в большинстве случаев это наиболее естественная формулировка запросов. Кроме того, эта форма позволяет уменьшить количество скобок, как показано ниже.
Формулу WFF ::= квантор_1 переменная_1 (квантор_2 переменная_2 (wff)), где каждый из кванторов квантор_1 и квантор_2 представляет или квантор EXISTS, или квантор FORALL, переменная_1 и переменная_2 - имена переменных) по умолчанию можно однозначно сократить к виду: квантор_1 переменная_1 квантор_2 переменная_2 (wff)

Таким образом, приводимое в четвертом примере выражение исчисления можно переписать (при желании) следующим образом:

SX.Sname WHERE EXISTS SPX EXISTS PX (SX.Sno = SPX.Sno AND SPX.Pno = PX.Pno AND PX.Color = 'Красный')

Вычислительные возможности исчисления кортежей

Добавить вычислительные возможности в исчисление довольно просто: необходимо расширить определение компарандов и целевых элементов так, чтобы они включали новую категорию - скалярные выражения, в которых операнды, в свою очередь, могут включать литералы, ссылки на атрибуты и (или) ссылки на итоговые функции. Для целевых элементов также требуется использовать спецификацию вида "AS attribute" для того, чтобы дать подходящее имя результирующему атрибуту, если нет очевидного наследуемого имени.

Т. к. что смысл скалярного выражения легко воспринимается, подробности опускаются. Однако синтаксис для ссылок на итоговые функции будет показан:

aggregate_function (expression [, attribute]), где
- aggregate_function - это COUNT, SUM, AVG, MAX или MIN (возможны, конечно, и некоторые другие функции),
- expression - это выражение исчисления кортежей (вычисляющее отношение),
- attribute - это такой атрибут результирующего отношения, по которому подсчитывается итог.

Для функции COUNT аргумент attribute не нужен и опускается; для других итоговых функций его можно опустить по умолчанию, если и только если вычисление аргумента expression дает отношение степени один и в таком случае единственный атрибут результата вычисления выражения expression подразумевается по умолчанию. Обратите внимание, что ссылка на итоговую функцию возвращает скалярное значение и поэтому допустима в качестве операнда скалярного выражения.

Из сказанного можно сделать следующие выводы:

- Итоговая функция действует в некоторых отношениях как новый тип квантора. В частности, если аргумент expression в данной ссылке на итоговую функцию представляется в виде "(tic) WHERE f", где tic- целевой элемент списка (target_item_commalist), а f - это формула WFF, и если экземпляр переменной кортежа T свободен в формуле f, то такой экземпляр переменной T связан в ссылке на итоговую функцию aggregate_function ((tic) WHERE f [, attribute])

- Пользователи, владеющие языком SQL, могут заметить, что с помощью двух следующих аргументов в ссылке на итоговую функцию, expression и attribute, можно избежать необходимых для SQL специальных приемов использования оператора DISTINCT для исключения дублирующих кортежей, если это требуется, перед выполнением итоговой операции. Вычисление аргумента expression дает отношение, из которого повторяющиеся кортежи всегда исключаются по определению. Аргумент attribute обозначает атрибут такого отношения, по которому выполняются итоговые вычисления, а дублирующие значения перед подсчетом итога из такого атрибута не удаляются. Конечно, атрибут может не содержать никаких дублирующих значений в любом случае, в частности, если такой атрибут является первичным ключом.

Получить номера деталей и их вес всех типов деталей, вес которых превышает 10 ед.	(PX.Pno, PX.Weight AS GMWT) WHERE PX.Weight > 10
Получить всех поставщиков, добавив для каждого литеральное значение "Поставщик"	(SX, 'Поставщик' AS TAG)
Получить каждую поставку с полными данными о входящих в нее деталях и общим весом поставки	(SPX.Sno, SPX.Qty, PX, PX.Weight * SPX.Qty AS SHIPWT) WHERE PX.Pno = SPX.Pno
Для каждой детали получить ее номер и общее поставляемое количество	(PX.Pno, SUM (SPX WHERE SPX.Pno = PX.Pno, Qty) AS TOTQTY)
Получить общее количество поставляемых деталей	(SUM (SPX, Qty) AS GRANDTOTAL))
Для каждого поставщика получить его номер и общее количество поставляемых им деталей	(SX.Sno, COUNT (SPX WHERE SPX.Sno = SX.Sno) AS NUMBER_OF_PARTS)
Получить города, в которых хранится а) более пяти красных деталей; б) не более пяти красных деталей	PX.City WHERE COUNT (PY WHERE PY.City = PX.City AND PY.Color = 'Красный') > 5

Исчисление доменов

Реляционное исчисление, ориентированное на домены (или исчисление доменов), отличается от исчисления кортежей тем, что в нем используются переменные доменов вместо переменных кортежей, т. е. переменные, принимающие свои значения в пределах домена, а не отношения. (Замечание. "Переменную домена" было бы лучше назвать "скалярной переменной", так как ее значения - это элементы домена, т.е. скаляры, а не сами домены.)

С практической точки зрения большинство очевидных различий между версиями исчисления для доменов и кортежей основано на том, что версия для доменов поддерживает дополнительную форму условия, которое мы будем называть условием принадлежности. В общем виде условие принадлежности можно записать так:

R (pair, pair, ...),

где R - это имя отношения, а каждая пара pair имеет вид A : v (где A - атрибут отношения R, а v - или переменная домена, или литерал). Проверка условия дает значение истина, если и только если существует кортеж в отношении R, имеющий определенные значения для определенных атрибутов. Например, вычисление выражения

SP (Sno : 1, Pno : 1) - дает значение истина, если и только если в отношении SP существует кортеж со значением Sno, равным 1, и значением Pno, равным 1. Аналогично, условие принадлежности

SP (Sno : SX, Pno : PX) - принимает значение истина, если и только если в отношении SP существует кортеж со значением Sno, эквивалентным текущему значению переменной домена SX (какому бы то ни было), и значением Pno, эквивалентным текущему значению переменной домена PX (опять же какому бы то ни было).

Далее будем подразумевать существование переменных доменов с именами: образуемыми добавлением букв X, Y, Z, ... к соответствующим именам доменов. Напомним, что в базе данных поставщиков и деталей каждый атрибут имеет такое же имя, как и соответствующий ему домен, за исключением атрибутов Sname и Pname, для которых соответствующий домен называется просто Name.

Множество всех номеров поставщиков.	(SX)
Множество всех номеров поставщиков отношения S.	(SX) WHERE S (Sno : SX)
Подмножество номеров поставщиков из города Смоленск.	(SX) WHERE S (Sno : SX, City : 'Смоленск')
Получить номера и города поставщиков, поставляющих деталь под номером 2. Обратите внимание, что для этого запроса, выраженного в терминах исчисления кортежей, требовался квантор существования; заметьте также, что это первый из приведенных здесь примеров, где действительно необходимы скобки для списка целевых элементов.	(SX, CityX) WHERE S (Sno : SX, City : CityX) AND SP (Sno : SX, Pno : 2)
Получить такие пары ‘номер поставщика’ – ‘номер детали’, что поставщики и детали размещены в одном городе".	(SX, PX) WHERE S (Sno : SX, City : CityX) AND P (Pno : PX, City : CityY) AND CityX <> CityY

Для сравнения с реляционной алгеброй некоторые примеры соответствуют рассмотренным ранее.

Получить номера поставщиков из Смоленска со статусом, большим 20	SX WHERE EXISTS StatusX (StatusX > 20 AND S (Sno : SX, Status : StatusX, City : 'Смоленск')) Обратите внимание, что кванторы все еще требуются. Этот пример несколько неуклюжий по сравнению с его аналогом, выраженным в терминах исчисления кортежей. С другой стороны, конечно, есть случаи, когда верно обратное; смотрите, в частности, более сложные примеры, приведенные ниже.
Получить все такие пары номеров поставщиков, что два поставщика размещаются в одном городе	(SX AS FirstSno, SY AS SecondSno) WHERE EXSIST CityZ (S (Sno : SX, City : CityZ) AND S (Sno : SY, City : CityZ) AND SX < SY)

Получить имена поставщиков, которые поставляют по крайней мере одну красную деталь	NAMEX WHERE EXISTS SX EXISTS PX (S (Sno : SX, Sname : NameX) AND SP (Sno : SX, Pno : PX) AND P (Pno : PX, Color : 'Красный'))
Получить имена поставщиков, которые поставляют все типы деталей	NameX WHERE EXISTS SX (S (Sno : SX, Sname : NameX) AND FORALL PX (IF P (Pno : PX) THEN SP (Sno : SX, Pno : PX)))
Получить имена поставщиков, которые не поставляют деталь с номером 2	NameX WHERE EXISTS SX (S (Sno : SX, Sname : NameX) AND NOT SP (Sno : SX, Pno : 2))
Получить номера поставщиков, которые поставляют по крайней мере все типы деталей, поставляемых поставщиком с номером 2	SX WHERE FORALL PX (IF SP (Sno : 2, Pno : PX) THEN SP (Sno : SX, Pno : PX))
Получить номера деталей, которые или весят более 16 фунтов, или поставляются поставщиком с номером 2, или и то, и другое	PX WHERE EXISTS WeightX (P (Pno : PX, Weight : WeightX) AND WeightX > 16) OR SP (Sno : 2, Pno : PX)

Исчисление доменов, как и исчисление кортежей, формально эквивалентно реляционной алгебре (т.е. оно реляционно полно). Для доказательства можно сослаться, например, на работы Ульмана (Ullman).

Реляционное исчисление и реляционная алгебра

Ранее утверждалось, что реляционная алгебра и реляционное исчисление в своей основе эквивалентны. Обсудим это утверждение более подробно. Вначале Кодд показал в своей статье, что алгебра, по крайней мере, мощнее исчисления. (Термин "исчисление" будет использоваться для обозначения исчисления кортежей.) Он сделал это, придумав алгоритм, называемый "алгоритмом редукции Кодда", с помощью которого любое выражение исчисления можно преобразовать в семантически эквивалентное выражение алгебры. Мы не станем приводить здесь полностью этот алгоритм, а ограничимся примером, иллюстрирующим в общих чертах, как этот алгоритм функционирует.

В качестве основы для нашего примера используется база данных поставщиков, деталей и проектов. Для удобства приводится набор примерных значений для этой базы данных.

Таблица Поставщики (S)

Sno	Same	Status	City
1	Алмаз	20	Смоленск
2	Циклон	10	Владимир
3	Дельта	30	Владимир
4	Орион	20	Смоленск
5	Аргон	30	Ярославль

Таблица Детали (P) Pno Pname

Pno	Pname	Color	Weight	City
1	Гайка	Красный	12	Смоленск
2	Болт	Зеленый	17	Владимир
3	Винт	Синий	17	Рязань
4	Винт	Красный	14	Смоленск
5	Шайба	Синий	12	Владимир
6	Шпунт	Зеленый	19	Смоленск

Таблица Проекты (J)

Jno	Jname	City
1	Ангара	Владимир
2	Алтай	Рязань
3	Енисей	Ярославль
4	Амур	Ярославль

Jno	Jname	City
5	Памир	Смоленск
6	Чегет	Тверь
7	Эльбрус	Смоленск

Таблица Поставки (SPJ)

Sno	Pno	Jno	Qty
1	1	1	200
1	1	4	700
2	3	1	400
2	3	2	200
2	3	3	200
2	3	4	500
2	3	5	600
2	3	6	400
2	3	7	800
2	5	2	100
3	3	1	200
3	4	2	500
4	6	3	300
4	6	7	300
5	2	2	200
5	2	4	100
5	5	5	500
5	5	7	100
5	6	2	200
5	1	4	100
5	3	4	200
5	4	4	800
5	5	4	400
5	6	4	500

Рассмотрим запрос: "Получить имена и города поставщиков, обеспечивающих по крайней мере один проект в городе Ярославль с поставкой по крайней мере 50 штук каждой детали". Выражение исчисления для этого запроса следующее:

(SX.Name, SX.City) WHERE EXISTS JX FORALL PX EXISTS SPJX (JX.City = 'Ярославль' AND JX.Jno = SPJX.Jno AND PX.Pno = SPJX.Pno AND SX.Sno = SPJX.Sno AND SPJX.Qty >= 50)

Здесь SX, PX, JX и SPJX - переменные кортежей, берущие свои значения из отношений S, P, J и SPJ соответственно. Теперь покажем, как вычислить это выражение, чтобы добиться необходимого результата.

Шаг 1. Для каждой переменной кортежа выбираем ее область значений (т.е. набор всех значений для этой переменной) если это возможно. Выражение «выбираем, если возможно» подразумевает, что существует условие выборки, встроенное в фразу WHEPE, которую можно использовать, чтобы сразу исключить из рассмотрения некоторые кортежи. В нашем случае выбираются следующие наборы кортежей:

SX : Все кортежи отношения S 5 кортежей
PX : Все кортежи отношения P 6 кортежей
JX : Кортежи отношения J, в которых City = 'Ярославль' 2 кортежа SPJX : Кортежи отношения SPJ, в которых Qty >= 50 24 кортежа

Шаг 2. Строим декартово произведение диапазонов, выбранных на первом шаге. Получим ...
Для экономии места таблица не приводится. Полное произведение содержит 5*6*2*24 = 1440 кортежей.

Шаг 3. Осуществляем выборку из произведения, построенного на втором шаге в соответствии с частью «условие соединения» фразы WHERE. В нашем примере эта часть следующая:

JX.Jno = SPJX.Jno AND PX.Pno = SPJX.Pno AND SX.Sno = SPJX.Sno AND

Поэтому из произведения исключаются кортежи, для которых значение Sno поставщика не равно значению Sno поставки, значение Pno детали не равно значению Pno поставки, значение Jno проекта не равно значению Jno поставки, после чего получаем подмножество декартова произведения, состоящее только из 10 кортежей.

Шаг 4. Применяем кванторы справа налево следующим образом:
- Для квантора «EXISTS RX» (где RX - переменная кортежа, принимающая значение на не котором отношении R) проецируем текущий промежуточный результат, чтобы исключить все атрибуты отношения R.
- Для квантора «FORALL RX» делим текущий промежуточный результат на отношение «выбранной области значений», соответствующее RX, которое было получено выше. При выполнении этой операции также будут исключены все атрибуты отношения R.

В нашем примере имеем следующие кванторы: EXISTS JX FORALL PX EXISTS SPJX Выполняем соответствующие операции.

- EXISTS SPJX. Проецируем, исключая атрибуты отношения SPJ (SPJ.Sno, SPJ.Pno, SPJ.Jno и SPJ.Qty). В результате получаем:

Sno	Same	Status	City	Pno	Pname	Color	Weight	City	Jno	Jname	City
1	Алмаз	20	Смоленск	1	Гайка	Красный	12	Смоленск	4	Амур	Ярославль
2	Циклон	10	Владимир	3	Винт	Синий	17	Рязань	3	Енисей	Ярославль
2	Циклон	10	Владимир	3	Винт	Синий	17	Рязань	4	Амур	Ярославль
4	Орион	20	Смоленск	6	Шпон	Красный	19	Смоленск	3	Енисей	Ярославль
5	Аргон	30	Ярославль	2	Болт	Зеленый	17	Владимир			Ярославль
5	Аргон	30	Ярославль	1	Гайка	Красный	12	Смоленск	4	Амур	Ярославль
5	Аргон	30	Ярославль	3	Винт	Синий	17	Рязань	4	Амур	Ярославль
5	Аргон	30	Ярославль	4	Винт	Красный	14	Смоленск	4	Амур	Ярославль
5	Аргон	30	Ярославль	5	Шайба	Синий	12	Владимир	4	Амур	Ярославль
5	Аргон	30	Ярославль	6	Шпунт	Красный	19	Смоленск	4	Амур	Ярославль

- FORALL PX. Делим на отношение P. В результате получаем: 17

Sno	Same	Status	City	Jno	Jname	City
5	Аргон	30	Ярославль	4	Амур	Ярославль

- EXISTS JX. Проецируем, исключая атрибуты отношения J (J.Jno, J.name и J.City). В результате получаем: Sname Status City

Sno	Same	Status	City
5	Аргон	30	Ярославль

Шаг 5. Проецируем результат шага 4 в соответствии со спецификациями в целевом списке элементов. В нашем примере целевым элементом списка будет: SX.Sname, SX.City. Следовательно, конечный результат таков:

Same	City
Аргон	Ярославль

Из сказанного выше следует, что начальное выражение исчисления семантически эквивалентно определенному вложенному алгебраическому выражению, а если быть более точным, то проекции от проекции деления проекции выборки из произведения четырех выборок (!). Этим завершаем пример. Конечно, можно намного улучшить алгоритм, хотя многие подробности скрыты в наших пояснениях; но вместе с тем, необходим адекватный пример, предлагающий общую идею.

Теперь можно объяснить одну из причин (и не только одну) определения Коддом ровно восьми алгебраических операторов. Эти восемь операторов обеспечивают соглашение целевого языка, как средства возможной реализации исчисления. Другими словами, для

данного языка, такого как QUEL, который основывается на исчислении, одно из возможных применений заключается в том, чтобы можно было брать запрос в том виде, в каком он предоставляется пользователем (являющийся, по существу, просто выражением исчисления), и применять к нему алгоритм получения эквивалентного алгебраического выражения. Это алгебраическое выражение, конечно, содержит множество алгебраических операций, которые согласно определению по своей природе выполнимы. (Следующий шаг состоит в продолжении оптимизации этого алгебраического выражения.)

Также необходимо отметить, что восемь алгебраических операторов Кодда являются мерой оценки выразительной силы любого языка баз данных (существующего или предлагаемого). Обсудим этот вопрос подробнее.

Во-первых, язык называется реляционно полным, если он по своим возможностям, по крайней мере, не уступает реляционному исчислению, т.е. любое отношение, которое можно определить с помощью реляционного исчисления, также можно определить и с помощью некоторого выражения рассматриваемого языка. «Реляционно полный» - значит, не уступающий по возможностям алгебре, а не исчислению, но это то же самое. По сути, из существования алгоритма преобразования Кодда немедленно следует, что реляционная алгебра обладает реляционной полнотой.)

Реляционную полноту можно рассматривать как основную меру возможностей выборки и выразительной силы языков баз данных вообще. В частности, так как исчисление и алгебра - реляционно полные, они могут служить базисом для проектирования языков, не уступающих им по выразительности, без необходимости пересортировки для использования циклов - особенно важное замечание, если язык предназначается конечным пользователям, хотя оно также применимо, если язык предназначается прикладным программистам.

Далее, поскольку алгебра реляционно полная, то, чтобы доказать, что некоторый язык L обладает реляционной полнотой, достаточно показать, что в языке L есть аналоги всех восьми алгебраических операций (на самом деле достаточно по казать, что в нем есть аналоги пяти примитивных операций) и что операндами любой операции языка L могут быть любые выражения этого языка. Язык SQL - это пример языка, реляционную замкнутость которого можно показать таким способом. Язык QUEL – еще один такой пример. В действительности на практике часто проще показать, что в данном языке есть эквиваленты алгебраических операций, чем найти в нем эквиваленты выражений исчисления. Именно поэтому реляционная полнота обычно определяется в терминах алгебраических выражений, а не выражений исчисления.

Функциональные зависимости

Для демонстрации основных идей данной темы, будет использоваться несколько измененная версия отношения поставок SP, содержащая атрибут City, представляющий город соответствующего поставщика. Это измененное отношение будет называться SCP.

Sno	City	Pno	Qty
1	Смоленск	1	100
1	Смоленск	2	100
2	Владимир	1	200
2	Владимир	2	200
3	Владимир	2	300
4	Смоленск	2	400
4	Смоленск	4	400
4	Смоленск	5	400

Следует четко различать:
1. значение этого отношения в определенный момент времени;
2. и набор всех возможных значений, которые данное отношение может принимать в различные моменты времени.

Примеры ФЗ, которым удовлетворяет отношение SCP в данном состоянии:
{ Sno } -> {City}
{Sno,Pno} -> {Qty}
{Sno,Pno} -> {City}
{Sno,Pno} -> { City, Qty }
{Sno,Pno} -> {Sno}
{Sno,Pno} -> { Sno, Pno, City, Qty }
{ Sno } -> {Qty}
{ Qty } -> {Sno}

Определение 1. Пусть R - это отношение, а X и Y - произвольные подмножества множества атрибутов отношения R. Тогда Y функционально зависимо от X, что в символическом виде записывается как X -> Y тогда и только тогда? Когда любое значение множества X связано в точности с одним значением множества Y.

Левая и правая стороны ФЗ будут называются детерминантом и зависимой частью соответственно.

Определение 2. Пусть R является переменной-отношением, а X и Y - произвольными подмножествами множества атрибутов переменной-отношения R. Тогда X -> Y тогда и только тогда, когда для любого допустимого значения отношения R любое значение X связано в точности с одним значением Y.

Определение 2а. Пусть R(A1, A2, ..., An) - схема отношения. Функциональная зависимость, обозначаемая X -> Y между двумя наборами атрибутов X и Y, которые являются подмножествами R определяет ограничение на возможность существования кортежа в некотором отношении r. Ограничение означает, что для любых двух кортежей t1 и t2 в r, для которых имеет место t1[X] = t2[X], также имеет место t1[Y] = t2[Y].

1. Если ограничение на схеме отношения R утверждает, что не может быть более одного кортежа со значением атрибутов X в любом отношении экземпляре отношения r, то X является потенциальным ключом R. Это означает, что X -> Y для любого подмножества атрибутов Y из R. Если X является потенциальным ключ R, то X -> R.
2. Если X -> Y в R, это не означает, что Y -> X в R.

Функциональная зависимость является семантическим свойством, т. е. свойством значения атрибутов.
Примеры безотносительных ко времени ФЗ для переменной-отношения SCP:

{Sno,Pno} -> {Sno,Pno} -> {Sno,Pno} -> {Sno,Pno} -> {Sno,Pno} -> { Sno } ->
{Qty}
{City}
{ City, Qty }
{Sno}
{ Sno, Pno, City, Qty } {City}

Следует обратить внимание на ФЗ, которые выполняются для отношения SCP, но не выполняются «всегда» для переменной-отношения SCP:

{Sno} -> {Qty} {Qty} -> {Sno}
Если X является потенциальным ключом переменной-отношения R, то все атрибуты Y переменной-отношения R должны быть обязательно ФЗ от X (это следует из определения потенциального ключа). Если переменная-отношение R удовлетворяет ФЗ X -> Y и X не является потенциальным ключом, то R будет характеризоваться некоторой избыточностью. Например, в случае отношения SCP сведения о том, что каждый данный поставщик находится в данном городе, будут повторяться много раз (это хорошо видно из таблицы).
Эта избыточность приводит к разным аномалиям обновления, получившим такое название по историческим причинам. Под этим понимаются определенные трудности, появляющиеся при выполнении операций обновления INSERT, DELETE и UPDATE. Рассмотрим избыточность, соответствующую ФЗ (Sno -> City). Ниже поясняются проблемы, которые возникнут при выполнении каждой из указанных операций обновления.

1. Операция INSERT. Нельзя поместить в переменную-отношение SCP информацию о том, что некоторый поставщик находится в определенном городе, не указав сведения хотя бы об одной детали, поставляемой этим поставщиком.
2. Операция DELETE. Если из переменной-отношения SCP удалить кортеж, который является единственным для некоторого поставщика, будет удалена не только информация о поставке поставщиком некоторой детали, но также информация о том, что этот поставщик находится в определенном городе. В действительности проблема заключается в том, что в переменной-отношении SCP содержится слишком много собранной в одном месте информации, поэтому при удалении некоторого кортежа теряется слишком много информации.
3. Операция UPDATE. Название города для каждого поставщика повторяется в переменной-отношении SCP несколько раз, и эта избыточность приводит к возникновению проблем при обновлении.

Для решения всех этих проблем, как предлагалось выше, необходимо выполнить декомпозицию переменной-отношения SCP на две следующие переменные-отношения.
S' { Sno, City }
SP { Sno, Pno, Qty }

Даже если ограничиться рассмотрением ФЗ, которые выполняются «всегда», множество ФЗ, выполняющихся для всех допустимых значений данного отношения, может быть все еще очень большим. Поэтому встает задача сокращения множества ФЗ до компактных размеров. Важность этой задачи вытекает из того, что ФЗ являются ограничениями целостности. Поэтому при каждом обновлении данных в СУБД все они должны быть проверены. Следовательно, для заданного множества ФЗ S желательно найти такое множество T, которое (в идеальной ситуации) было бы гораздо меньшего размера, чем множество S, причем каждая ФЗ множества S могла бы быть заменена ФЗ множества T. Если бы такое множество T было найдено, то в СУБД достаточно было бы использовать ФЗ из множества T, а ФЗ из множества S подразумевались бы автоматически.

Очевидным способом сокращения размера множества ФЗ было бы исключение тривиальных зависимостей, т. е. таких, которые не могут не выполняться. Примером тривиальной ФЗ для отношения SCP может быть { Sno, Pno } -> { Sno }.

Определение 3. ФЗ (X -> Y) тривиальная тогда и только тогда, когда Y включается в X.
Одни ФЗ могут подразумевать другие ФЗ. Например, зависимость
{ Sno, Pno } -> { City, Qty }

подразумевает следующие ФЗ

{ Sno, Pno } -> City { Sno, Pno } -> Qty

В качестве более сложного примера можно привести переменную-отношение R с атрибутами A, B и C, для которых выполняются ФЗ (A -> B) и (B -> C). В этом случае также выполняется ФЗ (A -> C), которая называется транзитивной ФЗ.

Определение 4. Множество всех ФЗ, которые задаются данным множеством ФЗ S, называется замыканием S и обозначается символом S+.

Из сказанного выше становится ясно, что для выполнения сформулированной задачи следует найти способ вычисления S+ на, основе S.

Первая попытка решить эту проблему принадлежит Армстронгу (Armstrong), который предложил набор правил вывода новых ФЗ на основе заданных (эти правила также называются аксиомами Армстронга).

Пусть в перечисленных ниже правилах A, B и C – произвольные подмножества множества атрибутов заданной переменной-отношения R, а символическая запись AB означает { A, B }. Тогда правила вывода определяются следующим образом.

1. Правило **рефлексивности**: (B принадлежит A) следовательно (A -> B)
2. Правило **дополнения**: (A -> B) следовательно AC -> BC
3. Правило **транзитивности**: (A -> B) и (B -> C) следовательно (A -> C)

Каждое и этих правил может быть непосредственно доказано на основе определения ФЗ. Более того, эти правила являются полными в том смысле, что для заданного множества ФЗ S минимальный набор ФЗ, которые подразумевают все зависимости из множества S, может быть выведен из S на основе этих правил. Они также являются исчерпывающими, поскольку никакие дополнительные ФЗ (т.е. ФЗ, которые не подразумеваются ФЗ множества S) с их помощью не могут быть выведены. Иначе говоря, эти правила могут быть использованы для получения замыкания S+.

Из трех описанных выше правил для упрощения задачи практического вычисления замыкания S+ можно вывести несколько дополнительных правил. (Примем, что D - это другое произвольное подмножество множества атрибутов R.)

4. Правило **самоопределения**: A -> A
5. Правило **декомпозиции**: (A -> BC) следовательно (A -> B) и (A -> C)
6. Правило **объединения**: (A -> B) и (A -> C) следовательно (A -> BC)
7. Правило **композиции**: (A -> B) и (C -> D) следовательно (AC -> BD)

Кроме того, Дарвен (Darwen) доказал следующее правило, которое назвал общей теоремой объединения:

8. (A->B) и (C->D) следовательно (A(C–B)->BD).

Упражнение. Пусть дана некоторая переменная-отношение R с атрибутами A, B, C, D, E, F и следующими ФЗ:
A -> BC
B -> E
CD -> EF
Показать, что для переменной-отношения R также выполняется ФЗ (AD -> F), которая вследствие этого принадлежит замыканию заданного множества ФЗ.

1. A -> BC (дано)
2. A -> C (следует из п. 1 согласно правилу декомпозиции)
3. AD -> CD (следует из п. 2 согласно правилу дополнения)
4. CD -> EF (дано)
5. AD -> EF (следует из п. 3 и 4 согласно правилу транзитивности)
6. AD -> F (следует из п. 5 согласно правилу декомпозиции)

Хотя эффективный алгоритм для вычисления замыкания S+ на основе множества ФЗ S так и не сформулирован, можно описать алгоритм, который определяет, будет ли данная ФЗ находиться в данном замыкании. Для этого, прежде всего, следует дать определение понятию **суперключ**.

Определение 5. Суперключ переменной-отношения R - это множество атрибутов переменной-отношения R, которое содержит в виде подмножества (но не обязательно собственного подмножества), по крайней мере, один потенциальный ключ.

Из этого определения следует, что суперключи для данной переменной-отношения R - это такие подмножества K множества атрибутов переменной-отношения R, что ФЗ (K -> A) истинна для каждого атрибута A переменной- отношения R.

Предположим, что известны некоторые ФЗ, выполняющиеся для данной переменной-отношения, и требуется определить потенциальные ключи этой переменной-отношения. По определению потенциальными ключами называются неприводимые суперключи. Таким образом, выясняя, является ли данное множество атрибутов K суперключом, можно в значительной степени продвинуться к выяснению вопроса, является ли K потенциальным ключом. Для этого нужно определить, будет ли набор атрибутов переменной-отношения R множеством всех атрибутов, функционально зависящих от K. Таким образом, для заданного множества зависимостей S, которые выполняются для переменной-отношения R, необходимо найти способ определения множества всех атрибутов переменной-отношения R, которые функционально зависимы от K, т.е. так называемое замыкание K^+ множества K для S.

Алгоритм вычисления этого замыкания имеет вид.

```
function Closure(K: SetOfAttr; S: SetOfFD): SetOfAttr;
var
        Jold, Jnew: SetOfAttr;
begin
        Jnew := K;
        repeat
                Jold := Jnew;
                foreach ((X -> Y in S) do
                        if (X принадлежит Jnew) then Jnew=Jnew + Y;
        until (Jold = Jnew);
        return Jold;
end; // of Closure
```

Определение. Суперключ в схеме отношения R(A1, A2, ... , An) - это набор атрибутов S из R со свойством, что ни для каких двух кортежей t1 и t2 в любом отношении над схемой R не будет выполняться равенство t1[S] = t2[S].

Определение. Потенциальный ключ K - это суперключ с дополнительным свойством: удаление любого атрибута из K приведет к тому, что K перестанет быть суперключом.

Определение. Атрибут в схеме отношения R называется первичным атрибутом R, если он является членом некоторого потенциального ключа R. Атрибут называется непервичным, если он не является первичным атрибутом, то есть, если он не является членом какого-либо потенциального ключа.

Алгоритм нахождения ключа K схемы отношения R для заданного множество функциональных зависимостей F

Вход: Схема отношения R и множество функциональных зависимостей F на атрибутах R.

1. Пусть K := R.

2. Для каждого атрибута из K
   {
   вычислить Closure({K - A), F);
   если замыкание содержит все атрибуты из R, то установите K: = K - {A}
   };

Заметьте, что алгоритм определяет только один ключ из множества возможных ключей на R; возвращаемый ключ зависит от порядка, в котором атрибуты удаляются из R на шаге 2.

Упражнение. Пусть дана переменная-отношение R с атрибутами A, B, C, D, E и F и следующими ФЗ:
A -> BC
E -> CF
B -> E
CD -> EF
Вычислить замыкание $\{ A, B \}^+$ множества атрибутов { A, B }, исходя из заданного множества ФЗ.

1. Присвоим Jold и Jnew начальное значение - множество {A,B}.
2. Выполним внутренний цикл четыре раза - по одному разу для каждой заданной ФЗ. На первой итерации (для зависимости A -> BC) будет обнаружено, что левая часть действительно является подмножеством замыкания Jnew. Таким образом, к Jnew можно добавить атрибуты B и C. Jnew теперь представляет собой множество {A,B,C}.
3. На второй итерации (для зависимости E -> CF) обнаруживается, что левая часть не является подмножеством полученного до этого момента результата, который, таким образом, остается неизменным.
4. На третьей итерации (для зависимости B -> E) к Jnew будет добавлено множество E, которое теперь будет иметь вид {A, B, C, E}.
5. На четвертой итерации (для зависимости CD -> EF) Jnew останется неизменным.
6. Далее внутренний цикл выполняется еще четыре раза. На первой итерации результат останется прежним, на
   второй он будет расширен до {A, B, C, E, F}, а на третьей и четвертой - снова не изменится.
7. Наконец после еще одного четырехкратного прохождения цикла замыкание Jnew останется неизменным и весь
   процесс завершится с результатом $\{A, B\}^+ = \{A, B, C, E, F\}$. // Конец упр.

Выводы.

1. Для заданного множества ФЗ S легко можно указать, будет ли заданная ФЗ (X -> Y) следовать из S, поскольку это возможно тогда и только тогда, когда множество Y является подмножеством замыкания X^+ множества X для заданного множества S. Таким образом, представлен простой способ определения, будет ли данная ФЗ (X->Y) включена в замыкание S^+ множества S.
2. Напомним определение понятия суперключа. Суперключ переменной-отношения R - это множество атрибутов переменной-отношения R, которое в виде подмножества (но необязательно собственного подмножества) содержит по крайней мере один потенциальный ключ. Из этого определения прямо следует, что суперключи для данной пеменной-отношения R - это такие подмножества K множества атрибутов переменной-отношения R, что ФЗ (K -> A) будет истинна для каждого атрибута A переменной-отношения R. Другими словами, множество K является суперключ тогда и только тогда, когда замыкание K^+ для множества K в пределах заданного мноства ФЗ является множеством абсолютно всех атрибутов переменной-отношения R. (Кроме того, множество K является потенциальным ключом тогда и только тогда, когда оно является неприводимым суперключом).

Определение 6. Два множества ФЗ S1 и S2 эквивалентны тогда и только тогда, когда они являются покрытиями друг для друга, т. е. S1+ = S2+.

Упражнение.
Эквивалентны ли следующие множества ФЗ: F= {A ->C, AC ->D,E ->AD, E ->H} and G = {A ->CD, E ->AH}.
Проверим, что G покрывается множеством F и F покрывается множеством G

G покрывается множеством F:
{A}+ = {A, C, D} (относительно F), покрывается A ->CD из G
{E}+ = {E, A, D, H, C} (относительно F), покрывается E ->AH в G

F покрывается множеством G:
{A}+ = {A, C, D} (относительно G), покрывается A ->C в F

{A, C}+= {A, C, D} (относительно G), покрывается AC ->D в F
{E}+ = {E, A, H, C, D} (относительно G), покрывается E ->AD и E ->H в F

Каждое множество ФЗ эквивалентно, по крайней мере, одному неприводимому множеству.

Определение 7. Множество ФЗ является неприводимым тогда и только тогда, когда оно обладает всеми перечисленными ниже свойствами.
1. Каждая ФЗ этого множества имеет одноэлементную правую часть.
2. Ни одна ФЗ множества не может быть устранена без изменения замыкания этого множества.
3. Ни один атрибут не может быть устранен из левой части любой ФЗ данного множества без изменения замыкания множества.
   Если I является неприводимым множеством, которое эквивалентно множеству S, то проверка выполнения ФЗ из множества I автоматически обеспечит выполнение ФЗ из множества S.

Пример. Рассмотрим переменную-отношение деталей P с функциональными зависимостями, перечисленными ниже.

Pno -> Pname
Pno -> Color
Pno -> Weight
Pno -> City

Нетрудно заметить, что это множество функциональных зависимостей является неприводимым:
1. правая часть каждой зависимости содержит только один атрибут,
2. левая часть, очевидно, является неприводимой,
3. ни одна из функциональных зависимостей не может быть опущена без изменения замыкания множества (т. е. без
   утраты некоторой информации).

В противоположность этому приведенные ниже множества функциональных зависимостей не являются неприводимыми.

1. Pno -> { Pname, Color }
   Pno -> Weight
   Pno -> City
(Правая часть первой ФЗ не является одноэлементным множеством.)

2. { Pno, Pname } -> Color
   Pno -> Pname
   Pno -> Weight
   Pno -> City
(Первую ФЗ можно упростить, опустив атрибут Pname в левой части без изменения замыкания, т. е. она не является неприводимой слева.)

3. Pno -> Pno
   Pno -> Pname
   Pno -> Color
   Pno -> Weight
   Pno -> City
(Здесь первую ФЗ можно опустить без изменения замыкания.)

Можно сделать утверждение, что для любого множества ФЗ существует, по крайней мере, одно эквивалентное множество, которое является неприводимым. Это достаточно легко продемонстрировать на следующем примере. Пусть дано исходное множество ФЗ S. Тогда благодаря правилу декомпозиции можно без утраты общности предположить, что каждая ФЗ в этом множестве S имеет одноэлементную правую часть. Далее для каждой ФЗ f из этого множества S следует проверить каждый атрибут A в левой части зависимости f. Если

множество S и множество зависимостей, полученное в результате устранения атрибута A в левой части зависимости f, эквивалентны, значит этот атрибут следует удалить. Затем для каждой оставшейся во множестве S зависимости f, если множества S и S \ {f} эквивалентны, следует удалить зависимость f из множества S. Получившееся в результате таких действий множество S является неприводимым и эквивалентно исходному множеству S.

Алгоритм поиска минимального покрытия F для множества функциональных зависимостей E

Вход: Множество функциональных зависимостей E.

1. Пусть F: = E.
2. Заменить каждую функциональную зависимость X -> {A1, A2, ..., An} из F на n функциональных зависимостей X -> A1, X -> A2, ..., X -> An.
3. Для каждой функциональной зависимости X -> A из F
    1. Для каждого атрибута B, который является элементом X
        1. если { {F – {X -> A} } объединить { (X – {B}) -> A} } эквивалентно F заменить X -> A на (X – {B}) -> A в F.
4. Для каждой оставшейся функциональной зависимости X -> A из F
    1. если {F - {X -> A}} эквивалентно F, удалить X -> A из F.

Упражнение. Пусть дана переменная-отношение R с атрибутами A, B, C, D и следующими функциональными зависимостями.

A -> BC
B -> C
A -> B
AB -> C
AC -> D

Найти неприводимое множество функциональных зависимостей эквивалентное данному множеству.

1. Прежде всего, следует переписать заданные ФЗ таким образом, чтобы каждая из них имела одноэлементную правую часть.

A -> B
A -> C
B -> C
A -> B
AB -> C
AC -> D

Нетрудно заметить, что зависимость A -> B записана дважды, так что одну из них можно удалить.

2. Затем в левой части зависимости AC -> D может быть опущен атрибут C, поскольку дана зависимость A -> C, из которой по правилу дополнения можно получить зависимость A -> AC. Кроме того, дана зависимость AC -> D, из которой по правилу транзитивности можно получить зависимость A -> D. Таким образом, атрибут C в левой части исходной зависимости AC -> D является избыточным.
3. Далее можно заметить, что зависимость AB -> C может быть исключена, поскольку дана зависимость A -> C, из которой по правилу дополнения можно получить зависимость AB -> CB, а затем по правилу декомпозиции - зависимость AB -> C.
4. Наконец зависимость A -> C подразумевается зависимостями A -> B и B -> C, так что она также может быть отброшена. В результате получается неприводимое множество зависимостей.

A -> B
B -> C
A -> D

Множество ФЗ I, которое неприводимо и эквивалентно другому множеству ФЗ S, называется неприводимым покрытием множества S. Таким образом, с тем же успехом в системе вместо

исходного множества ФЗ S может использоваться его неприводимое покрытие I (здесь следует повторить, что для вычисления неприводимого эквивалентного покрытия I необязательно вычислять замыкание S+). Однако необходимо отметить, что для заданного множества ФЗ не всегда существует уникальное неприводимое покрытие.

Проектирование базы данных может быть выполнено с использованием двух подходов: снизу вверх (bottom-up) или сверху вниз (top-down).

1. Методология проектирования bottom-up (также называемая методологией синтеза) рассматривает основные связи между отдельными атрибутами в качестве отправной точки и использует их, чтобы построить схемы отношений схем. Этот подход не пользуется популярностью на практике, потому что она страдает от проблемы того, чтобы собрать большое число бинарных связей между атрибутами в качестве отправной точки. На пактике это сделать почти невозможно.
2. Методология проектирования top-down (также называемая методологией анализа) начинается с некоторого набора отношений, состоящих из атрибутов. Затем отношения анализируются отдельно или совместно, в результате чего происходит их декомпозиция до тех пор, пока не будут достигнуты все желаемые свойства.
   Неявными целями обеих методологий являются сохранение информации и минимизация избыточности.

Процесс нормализации схем отношений, основанный на операции декомпозии, должен обладать следующими свойства:

1. неаддитивностью JOIN или JOIN без потерь информации (NJP) , которая гарантирует, что в результате декомпозиции не появятся лишние кортежи.
2. свойством сохранения зависимостей (DPP), которое гарантирует, что каждая функциональная зависимость будет представлена в каком-либо отдельном отношении после декомпозиции.

Свойство NJP является очень критическим и должно быть достигнуто любой ценой. Свойство DPP желательно, но не всегда достижимо.

Теория проектирования реляционных баз данных

При проектировании базы данных решаются две основные проблемы:
1. Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области, и было, по возможности, лучшим (эффективным, удобным и т. д.)? Часто эту проблему называют проблемой логического проектирования баз данных.
2. Как обеспечить эффективность выполнения запросов к базе данных? Эту проблему обычно называют проблемой физического проектирования баз данных.

В случае реляционных баз данных нет общих рецептов по части физического проектирования. Здесь слишком много зависит от используемой СУБД. Поэтому ограничимся только существенными вопросами логического проектирования реляционных баз данных. Более того, не будем касаться определения ограничений целостности общего вида, а ограничимся ограничениями первичного и внешнего ключей. Будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том, из каких отношений должна состоять базы данных, и какие атрибуты должны быть у этих отношений.
Классический подход к проектированию реляционных баз данных заключается в том, что сначала предметная область представляется в виде одного или нескольких отношений, а далее осуществляется процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая. Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером набора ограничений является ограничение первой нормальной формы – значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, будем считать, что исходный набор отношений уже соответствует этому требованию.
В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

1. первая нормальная форма (1НФ или 1NF);
2. вторая нормальная форма (2НФ или 2NF);
3. третья нормальная форма (3НФ или 3NF);
4. нормальная форма Бойса-Кодда (НФБК или BCNF);
5. четвертая нормальная форма (4НФ или 4NF);
6. пятая нормальная форма, или нормальная форма проекции-соединения (5НФ или 5NF или PJ/NF).

Основные свойства нормальных форм такие:

1. каждая следующая нормальная форма в некотором смысле лучше предыдущей;
2. при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

Процесс проектирования реляционной базы данных на основе метода нормализации преследует две основные цели:
1. избежать избыточности хранения данных;
2. устранить аномалии обновления отношений.

Эти цели являются актуальными для информационных систем оперативной обработки транзакций (On-Line Transaction Processing – OLTP), которым свойственны частые обновления базы данных, и потому аномалии обновления могут сильно вредить

эффективности приложения. В информационных системах оперативной аналитической обработки (On-Line Analytical Processing – OLAP), в частности, в системах поддержки принятия решений, базы данных в основном используются для выборки данных. Поэтому аномалиями обновления можно пренебречь. Из этого не следует, что принципы нормализации непригодны при проектировании баз данных OLAP-приложений. Даже если схема такой базы данных должна быть денормализована по соображениям эффективности, то чтобы получить правильную денормализованную схему, нужно сначала понять, как выглядит нормализованная схема.

В основе метода нормализации лежит декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы. Считаются правильными такие декомпозиции отношения, которые обратимы, т. е. имеется возможность собрать исходное отношение из декомпозированных отношений без потери информации.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости.

Декомпозиция без потерь

Декомпозицией отношения R называется замена R на совокупность отношений $\{R_1, R_2, \ldots, R_n\}$ такую, что каждое из них есть проекция R, и каждый атрибут R входит хотя бы в одну из проекций декомпозиции.

Например, для отношения R с атрибутами {a, b, c} существуют следующие основные варианты декомпозиции:

- {a}, {b}, {c}
- {a}, {b, c}
- {a, b}, {c}
- {b}, {a, c}
- {a, b}, {b, c}
- {a, b}, {a, c}
- {b, c}, {a, c}
- {a, b}, {b, c}, {a, c}

Рассмотрим теперь отношение R', которое получается в результате операции естественного соединения (NATURAL JOIN), применённой к отношениям, полученным в результате декомпозиции R.

Декомпозиция называется декомпозицией без потерь, если R' в точности совпадает с R.

Неформально говоря, при декомпозиции без потерь отношение «разделяется» на отношения-проекции таким образом, что из полученных проекций возможна «сборка» исходного отношения с помощью операции естественного соединения.

Далеко не всякая декомпозиция является декомпозицией без потерь. Проиллюстрируем это на примере отношения R с атрибутами {a, b, c}, приведённом выше. Пусть отношение R имеет вид:

a	b	c
Москва	Россия	столица
Томск	Россия	не столица
Берлин	Германия	столица

Декомпозиция $R_1 = \{a\}, R_2 = \{b, c\}$ имеет вид:

a
Москва
Томск
Берлин

b	c
Россия	столица
Россия	не столица
Германия	столица

Результат операции соединения этих отношений: $R' = R_1$ NATURAL JOIN R_2

a	b	c
Москва	Россия	столица
Москва	Россия	не столица
Москва	Германия	столица
Томск	Россия	столица
Томск	Россия	не столица
Томск	Германия	столица
Берлин	Россия	столица
Берлин	Россия	не столица
Берлин	Германия	столица

Очевидно, что R' не совпадает с R, а значит такая декомпозиция не является декомпозицией без потерь. Рассмотрим теперь декомпозицию $R_1 = \{a, b\}, R_2 = \{a, c\}$:

a	b
Москва	Россия
Томск	Россия
Берлин	Германия

a	c
Москва	столица
Томск	не столица
Берлин	столица

Такая декомпозиция является декомпозицией без потерь, в чём читатель может убедиться самостоятельно.

В некоторых случаях отношение вовсе невозможно декомпозировать без потерь. Существуют также примеры отношений, для которых нельзя выполнить декомпозицию без потерь на две проекции, но которые можно подвергнуть декомпозиции без потерь на три или большее количество проекций.

Первая нормальная форма

Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице. Например, есть таблица «Автомобили»:

Фирма	Модели
BMW	M5, X5M, M1
Nissan	GT-R

Нарушение нормализации 1НФ происходит в моделях BMW, т.к. в одной ячейке содержится список из 3 элементов: M5, X5M, M1, т.е. он не является атомарным. Преобразуем таблицу к 1НФ:

Фирма	Модели
BMW	M5
BMW	X5M
BMW	M1
Nissan	GT-R

Вторая нормальная форма

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК).

Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

Например, дана таблица:

Модель	Фирма	Цена	Скидка
M5	BMW	5500000	5 %
X5M	BMW	6000000	5 %
M1	BMW	2500000	5 %
GT-R	Nissan	5000000	10 %

Таблица находится в первой нормальной форме, но не во второй. Цена машины зависит от модели и фирмы. Скидка зависят от фирмы, то есть зависимость от первичного ключа неполная. Исправляется это путем декомпозиции на два отношения, в которых не ключевые атрибуты зависят от ПК.

Модель	Фирма	Цена
M5	BMW	5500000
X5M	BMW	6000000
M1	BMW	2500000
GT-R	Nissan	5000000

Фирма	Скидка
BMW	5 %
Nissan	10 %

Третья нормальная форма
Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

Рассмотрим таблицу:

Модель	Магазин	Телефон
BMW	Риал-авто	87-33-98
Audi	Риал-авто	87-33-98
Nissan	Некст-Авто	94-54-12

Таблица находится во 2НФ, но не в 3НФ.
В отношении атрибут «Модель» является первичным ключом. Личных телефонов у автомобилей нет, и телефон зависит исключительно от магазина.
Таким образом, в отношении существуют следующие функциональные зависимости: Модель → Магазин, Магазин → Телефон, Модель → Телефон.
Зависимость Модель → Телефон является транзитивной, следовательно, отношение не находится в 3НФ.
В результате разделения исходного отношения получаются два отношения, находящиеся в 3НФ:

Магазин	Телефон
Риал-авто	87-33-98
Некст-Авто	94-54-12

Модель	Магазин
BMW	Риал-авто
Audi	Риал-авто
Nissan	Некст-Авто

Нормальная форма Бойса-Кодда (НФБК) (частная форма третьей нормальной формы)
Определение 3НФ не совсем подходит для следующих отношений:
1) отношение имеет два или более потенциальных ключа;
2) два и более потенциальных ключа являются составными;
3) они пересекаются, т.е. имеют хотя бы один общий атрибут.

Для отношений, имеющих один потенциальный ключ (первичный), НФБК является 3НФ.
Отношение находится в НФБК, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта.
Предположим, рассматривается отношение, представляющее данные о бронировании стоянки на день:

Номер стоянки	Время начала	Время окончания	Тариф
1	09:30	10:30	Бережливый
1	11:00	12:00	Бережливый
1	14:00	15:30	Стандарт
2	10:00	12:00	Премиум-В
2	12:00	14:00	Премиум-В
2	15:00	18:00	Премиум-А

Тариф имеет уникальное название и зависит от выбранной стоянки и наличии льгот, в частности:

- «Бережливый»: стоянка 1 для льготников
- «Стандарт»: стоянка 1 для не льготников
- «Премиум-А»: стоянка 2 для льготников
- «Премиум-В»: стоянка 2 для не льготников.

Таким образом, возможны следующие составные первичные ключи: {Номер стоянки, Время начала}, {Номер стоянки, Время окончания}, {Тариф, Время начала}, {Тариф, Время окончания}.

Отношение находится в 3НФ. Требования второй нормальной формы выполняются, так как все атрибуты входят в какой-то из потенциальных ключей, а неключевых атрибутов в отношении нет. Также нет и транзитивных зависимостей, что соответствует требованиям третьей нормальной формы. Тем не менее, существует функциональная зависимость Тариф → Номер стоянки, в которой левая часть (детерминант) не является потенциальным ключом отношения, то есть отношение не находится в нормальной форме Бойса — Кодда.

Недостатком данной структуры является то, что, например, по ошибке можно приписать тариф «Бережливый» к бронированию второй стоянки, хотя он может относиться только к первой стоянки.

Можно улучшить структуру с помощью декомпозиции отношения на два и добавления атрибута Имеет льготы, получив отношения, удовлетворяющие НФБК (подчёркнуты атрибуты, входящие в первичный ключ.):

Тариф	Время начала	Время окончания
Бережливый	09:30	10:30
Бережливый	11:00	12:00
Стандарт	14:00	15:30
Премиум-В	10:00	12:00
Премиум-В	12:00	14:00
Премиум-А	15:00	18:00

Тариф	Номер	Имеет
Бережливый	1	Да
Стандарт	1	Нет
Премиум-А	2	Да
Премиум-В	2	Нет

Четвертая нормальная форма
Отношение находится в 4НФ, если оно находится в НФБК и все нетривиальные многозначные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей.

В отношении R (A, B, C) существует многозначная зависимость R.A -> -> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

Предположим, что рестораны производят разные виды пиццы, а службы доставки ресторанов работают только в определенных районах города. Составной первичный ключ соответствующей переменной отношения включает три атрибута: {Ресторан, Вид пиццы, Район доставки}.

Ресторан	Вид пиццы	Район доставки
A1 Пицца	Толстая корка	Springfield
A1 Пицца	Толстая корка	Shelbyville
A1 Пицца	Толстая корка	Столица
A1 Пицца	Фаршированная корочка	Springfield
A1 Пицца	Фаршированная корочка	Shelbyville
A1 Пицца	Фаршированная корочка	Столица
Элитная пицца	Толстая корка	Столица
Элитная пицца	Фаршированная корочка	Столица
Пицца Винченцо	Толстая корка	Springfield
Пицца Винченцо	Толстая корка	Shelbyville
Пицца Винченцо	Толстая корка	Springfield
Пицца Винченцо	Толстая корка	Shelbyville

Такая переменная отношения не соответствует 4НФ, так как существует следующая многозначная зависимость:
{Ресторан} → {Вид пиццы}
{Ресторан} → {Район доставки}

То есть, например, при добавлении нового вида пиццы придется внести по одному новому кортежу для каждого района доставки. Возможна логическая аномалия, при которой определенному виду пиццы будут соответствовать лишь некоторые районы доставки из обслуживаемых рестораном районов.

Для предотвращения аномалии нужно декомпозировать отношение, разместив независимые факты в разных отношениях. В данном примере следует выполнить декомпозицию на {Ресторан, Вид пиццы} и {Ресторан, Район доставки}.

Однако, если к исходной переменной отношения добавить атрибут, функционально зависящий от потенциального ключа, например цену с учётом стоимости доставки ({Ресторан, Вид

пиццы, Район доставки} → Цена), то полученное отношение будет находиться в 4НФ и его уже нельзя подвергнуть декомпозиции без потерь.

Ресторан	Район доставки
A1 Пицца	Springfield
A1 Пицца	Shelbyville
A1 Пицца	Столица
Элитная пицца	Столица
Пицца Винченцо	Springfield
Пицца Винченцо	Shelbyville

Ресторан	Вид пиццы
A1 Пицца	Толстая корка
A1 Пицца	Фаршированная корочка
Элитная пицца	Толстая корка
Элитная пицца	Фаршированная корочка
Пицца Винченцо	Толстая корка

Пятая нормальная форма

Отношение находится в пятой нормальной форме (иначе — в проекционно-соединительной нормальной форме) тогда и только тогда, когда каждая нетривиальная зависимость соединения в нём определяется потенциальным ключом (ключами) этого отношения.

Это очень жесткое требование, которое можно выполнить лишь при дополнительных условиях. На практике трудно найти пример реализации этого требования в чистом виде.
Предположим, что нужно хранить данные об ассортименте нескольких продавцов, торгующих продукцией нескольких фирм (номенклатура товаров фирм может пересекаться):

Продавец	Фирма	Товар
Иванов	Рога и Копыта	Пылесос
Иванов	Рога и Копыта	Хлебница
Петров	Безенчук&Ко	Сучкорез
Петров	Безенчук&Ко	Пылесос
Петров	Безенчук&Ко	Хлебница
Петров	Безенчук&Ко	Зонт
Сидоров	Безенчук&Ко	Пылесос
Сидоров	Безенчук&Ко	Телескоп
Сидоров	Рога и Копыта	Пылесос
Сидоров	Рога и Копыта	Лампа
Сидоров	Геркулес	Вешалка

Если дополнительных условий нет, то данное отношение, которое находится в 4-й нормальной форме, является корректным и отражает все необходимые ограничения.

Теперь предположим, что нужно учесть следующее ограничение: каждый продавец имеет в своём ассортименте ограниченный список фирм и ограниченный список типов товаров и предлагает товары из списка товаров, производимые фирмами из списка фирм.

То есть продавец не имеет право торговать какими угодно товарами каких угодно фирм. Если продавец П имеет право торговать товарами фирмы Ф, и если продавец П имеет право

торговать товарами типа Т, то в этом случае в ассортимент продавца П входят товары типа Т фирмы Ф при условии, что фирма Ф производит товары типа Т.

Такое ограничение может быть вызвано, например, тем, что список типов товаров продавца ограничен имеющимися у него лицензиями, либо знаниями и квалификацией, необходимыми для их продажи, а список фирм каждого продавца определён партнёрскими соглашениями.

В рассматриваемом примере, в частности, предполагается, что продавец Иванов имеет право торговать товарами только фирмы «Рога и Копыта», продавец Петров — товарами только фирмы «Безенчук&Ко», зато продавец Сидоров не имеет права торговать хлебницами и сучкорезами и т. д.

Предложенное выше отношение не может исключить ситуации, при которых данное ограничение будет нарушено. Ничто не препятствует занести данные о торговле товаром, который данная фирма вообще не выпускает, либо данные о торговле товарами той фирмы, которую данный продавец не обслуживает, либо данные о торговле таким типом товара, который данный продавец не имеет право продавать.

Отношение не находится в 5NF, поскольку в нём есть нетривиальная зависимость соединения *{{Продавец, Фирма}, {Фирма, Товар}, {Продавец, Товар}}, однако подмножества {Продавец, Фирма}, {Фирма, Товар}, {Продавец, Товар} не являются суперключами исходного отношения.

В данном случае для приведения к 5NF отношение должно быть разбито на три: {Продавец, Фирма}, {Фирма, Товар}, {Продавец, Товар}.

Продавец	Фирма
Иванов	Рога и Копыта
Петров	Безенчук&Ко
Сидоров	Безенчук&Ко
Сидоров	Рога и Копыта
Сидоров	Геркулес

Фирма	Товар
Рога и Копыта	Пылесос
Рога и Копыта	Хлебница
Рога и Копыта	Лампа
Безенчук&Ко	Сучкорез
Безенчук&Ко	Пылесос
Безенчук&Ко	Хлебница
Безенчук&Ко	Зонт
Безенчук&Ко	Телескоп
Геркулес	Вешалка

Продавец	Товар
Иванов	Пылесос
Иванов	Хлебница
Петров	Сучкорез
Петров	Пылесос
Петров	Хлебница
Петров	Зонт
Сидоров	Телескоп
Сидоров	Пылесос
Сидоров	Лампа
Сидоров	Вешалка

Шестая нормальная форма

Переменная отношения находится в шестой нормальной форме тогда и только тогда, когда она удовлетворяет всем нетривиальным зависимостям соединения. Из определения следует, что переменная находится в 6НФ тогда и только тогда, когда она неприводима, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Каждая переменная отношения, которая находится в 6НФ, также находится и в 5НФ.

Идея «декомпозиции до конца» выдвигалась до начала исследований в области хронологических данных, но не нашла поддержки. Однако для хронологических баз данных максимально возможная декомпозиция позволяет бороться с избыточностью и упрощает поддержание целостности базы данных.

Для хронологических баз данных определены U_операторы, которые распаковывают отношения по указанным атрибутам, выполняют соответствующую операцию и упаковывают полученный результат. В данном примере соединение проекций отношения должно производится при помощи оператора U_JOIN.

Таб.№	Время	Должность	Домашний адрес
6575	01-01-2000:10-02-2003	слесарь	ул.Ленина,10
6575	11-02-2003:15-06-2006	слесарь	ул.Советская,22
6575	16-06-2006:05-03-2009	бригадир	ул.Советская,22

Переменная отношения «Работники» не находится в 6НФ и может быть подвергнута декомпозиции на переменные отношения «Должности работников» и «Домашние адреса работников».

Таб.№	Время	Должность
6575	01-01-2000: 10-02-2003	слесарь
6575	16-06-2006: 05-03-2009	бригадир

Таб.№	Время	Домашний адрес
6575	01-01-2000: 10-02-2003	ул.Ленина,10
6575	11-02-2003: 15-06-2006	ул.Советская, 22

Транзакции

Транзакция — это последовательность операций, выполняемая как единое целое. В составе транзакций можно исполнять почти все операторы языка Transact-SQL. Если при выполнении транзакции не возникает никаких ошибок, то все модификации базы данных, сделанные во время выполнения транзакции, становятся постоянными. Транзакция выполняется по принципу «все или ничего». Транзакция не оставляет данные в промежуточном состоянии, в котором база данных не согласована. Транзакция переводит базу данных из одного целостного состояния в другое.

В качестве примера транзакции рассмотрим последовательность операций по приему заказа в коммерческой компании. Для приема заказа от клиента приложение ввода заказов должно:

- выполнить запрос к таблице товаров и проверить наличие товара на складе;
- добавить заказ к таблице счетов;
- обновить таблицу товаров, вычтя заказанное количество товаров из количества товара, имеющегося в наличии;
- обновить таблицу продаж, добавив стоимость заказа к объему продаж служащего, принявшего заказ;
- обновить таблицу офисов, добавив стоимость заказа к объему продаж офиса, в котором работает данный служащий.

Для поддержания целостности транзакция должна обладать четырьмя свойствами АСИД: атомарность, согласованность, изоляция и долговечность. Эти свойства называются также ACID-свойствами (от англ., atomicity, consistency, isolation, durability).

- Атомарность (Atomicity). Транзакция должна представлять собой атомарную (неделимую) единицу работы (исполняются либо все модификации, из которых состоит транзакция, либо ни одна).
- Согласованность (или Непротиворечивость) (Consistency). По завершении транзакции все данные должны
  остаться в согласованном состоянии. Чтобы сохранить целостность всех данных, необходимо выполнение модификации транзакций по всем правилам, определенным в реляционных СУБД.
- Изоляция (Isolation). Модификации, выполняемые одними транзакциями, следует изолировать от модификаций, выполняемых другими транзакциями параллельно. Уровни изоляции транзакции могут изменяться в широких пределах. На каждом уровне изоляции достигается определенный компромисс между степенью распараллеливания и степенью непротиворечивости. Чем выше уровень изоляции, тем выше степень непротиворечивости данных. Но чем выше степень непротиворечивости, тем ниже степень распараллеливания и тем ниже степень доступности данных.
- Долговечность (или Устойчивость) (Durability). По завершении транзакции ее результат должен сохраниться в системе, несмотря на сбой системы, либо (что касается незафиксированных транзакций) может быть полностью отменен вслед за сбоем системы.

Способы определения границ транзакций

Замечания:

- В данной теме не будут учитываться особенности режима MARS (Multiple Active Result Sets). Режим MARS — это функция, которая появилась в SQL Server 2005 и ADO.NET 2.0 для выполнения нескольких пакетов по одному соединению. По умолчанию режим MARS отключен. Включить его можно, добавив в строку соединения параметр «MultipleActiveResultSets=True».

- В данной теме не будут рассматриваться распределенные транзакции.

По признаку определения границ различают автоматические, неявные и явные транзакции.

Автоматические транзакции. Режим автоматической фиксации транзакций является режимом управления транзакциями SQL Server по умолчанию. В этом режиме каждая инструкция T-SQL выполняется как отдельная транзакция. Если выполнение инструкции завершается успешно, происходит фиксация; в противном случае происходит откат. Если возникает ошибка компиляции, то план выполнения пакета не строится и пакет не выполняется.

В следующем примере ни одна из инструкций INSERT во втором пакете не выполнится из-за ошибки компиляции. При этом произойдет откат первых двух инструкций INSERT, и они не будут выполняться:

```
CREATE TABLE Tab1 (
    Col1 int NOT NULL PRIMARY KEY,
    Col2 char(3)
);

GO
INSERT INTO Tab1 VALUES (1, 'aaa');
INSERT INTO Tab1 VALUES (2, 'bbb');
INSERT INTO Tab1 VALUSE (3, 'ccc'); -- Синтаксическая ошибка.

GO
SELECT * FROM Tab1; -- Результат пустой. GO
```

В следующем примере третья инструкция INSERT вызывает ошибку дублирования первичного ключа во время выполнения. Поэтому первые две инструкции INSERT выполняются успешно и фиксируются, а третья инструкция INSERT вызывает ошибку времени выполнения и не выполняется.

```
CREATE TABLE Tab1 (
    Col1 int NOT NULL PRIMARY KEY,
    Col2 char(3)
);

GO
INSERT INTO Tab1 VALUES (1, 'aaa');
INSERT INTO Tab1 VALUES (2, 'bbb');
INSERT INTO Tab1 VALUES (1, 'ccc'); -- Ошибка времени исполнения. GO
SELECT * FROM Tab1; -- Возвращаются две строки.
GO
```

Неявные транзакции. Если соединение работает в режиме неявных транзакций, то после фиксации или отката текущей транзакции SQL Server автоматически начинает новую транзакцию. В этом режиме явно указывается только граница окончания транзакции с помощью инструкций COMMIT TRANSACTION и ROLLBACK TRANSACTION. Для ввода в действие поддержки неявных транзакций применяется инструкция SET IMPLICIT_TRANSACTION ON. В конце каждого пакета необходимо отключать этот режим. По умолчанию режим неявных транзакций в SQL Server отключен.

В следующем примере сначала создается таблица Tab1, затем включается режим неявных транзакций, после чего начинаются две транзакции. После их исполнения режим неявных транзакций отключается:

```
CREATE TABLE Tab1 (
    Col1 int NOT NULL PRIMARY KEY,
    Col2 char(3) NOT NULL
)

GO
SET IMPLICIT_TRANSACTIONS ON
-- Первая неявная транзакция, начатая оператором INSERT INSERT INTO Tab1 VALUES (1,
'aaa')
INSERT INTO Tab1 VALUES (2, 'bbb')
COMMIT TRANSACTION -- Фиксация первой транзакции
-- Вторая неявная транзакция, начатая оператором INSERT INSERT INTO Tab1 VALUES (3,
'ccc')
SELECT * FROM Tab1
COMMIT TRANSACTION -- Фиксация второй транзакции
SET IMPLICIT_TRANSACTIONS OFF
GO
```

Явные транзакции. Для определения явных транзакций используются следующие инструкции:

- BEGIN TRANSACTION – задает начальную точку явной транзакции для соединения;
- COMMIT TRANSACTION или COMMIT WORK – используется для успешного завершения транзакции, если не возникла ошибка;
- ROLLBACK TRANSACTION или ROLLBACK WORK – используется для отмены транзакции, во время которой возникла ошибка.
- SAVE TRANSACTION – используется для установки точки сохранения или маркера внутри транзакции. Точка сохранения определяет место, к которому может возвратиться транзакция, если часть транзакции условно отменена. Если транзакция откатывается к точке сохранения, то ее выполнение должно быть продолжено до завершения с обработкой дополнительных инструкций языка T-SQL, если необходимо, и инструкции COMMIT TRANSACTION, либо транзакция должна быть полностью отменена откатом к началу. Для отмены всей транзакции следует использовать инструкцию ROLLBACK TRANSACTION; в этом случае отменяются все инструкции транзакции.

В следующем примере демонстрируется использование точки сохранения транзакции для отката изменений:

```
USE pubs
GO

BEGIN TRANSACTION royaltychange
        UPDATE titleauthor
        SET royaltyper = 65
        FROM titleauthor, titles
        WHERE royaltyper = 75 AND titleauthor.title_id = titles.title_id
                AND title = 'The Gourmet Microwave'
```

```
        UPDATE titleauthor
        SET royaltyper = 35
        FROM titleauthor, titles
        WHERE royaltyper = 25 AND titleauthor.title_id = titles.title_id
                AND title = 'The Gourmet Microwave'
SAVE TRANSACTION percentchanged

/* После того, как обновлено royaltyper для двух авторов, вставляется точка сохранения percentchanged, а затем определяется, насколько изменится заработок авторов после увеличения на 10 процентов цены книги */
UPDATE titles
SET price = price * 1.1
WHERE title = 'The Gourmet Microwave'

SELECT (price * royalty * ytd_sales) * royaltyper
FROM titles, titleauthor
WHERE title = 'The Gourmet Microwave' AND titles.title_id = titleauthor.title_id

/* Откат транзакции до точки сохранения и фиксация транзакции в целом */
ROLLBACK TRANSACTION percentchanged
COMMIT TRANSACTION
```

Режим явных транзакций действует только на протяжении данной транзакции. После завершения явной транзакции соединение возвращается в режим, заданный до запуска этого режима, то есть в неявный или автоматический.

Функции для обработки транзакций

- @@TRANCOUNT возвращает число активных транзакций для текущего соединения. Инструкция BEGIN TRANSACTION увеличивает значение @@TRANCOUNT на 1, а инструкция ROLLBACK TRANSACTION уменьшает его до 0 (исключение — инструкция ROLLBACK TRANSACTION имя_точки_сохранения, которая не влияет на значение @@TRANCOUNT). Инструкции COMMIT TRANSACTION уменьшают значение @@TRANCOUNT на 1.
- XACT_STATE () сообщает о состоянии пользовательской транзакции текущего выполняемого запроса в соответствии с данными, представленными в таблице.

Возвращаемое значение	Пояснение
1	Текущий запрос содержит активную пользовательскую транзакцию и может выполнять любые действия, включая запись данных и фиксацию транзакции.
0	У текущего запроса нет активной пользовательской транзакции.
-1	В текущем запросе есть активная транзакция, однако произошла ошибка, из-за которой транзакция классифицируется как не фиксируемая. Запросу не удается зафиксировать транзакцию или выполнить откат до точки сохранения; можно только запросить полный откат транзакции.

Ограничения.

- Функция @@TRANCOUNT не может использоваться для определения фиксируемости транзакции.
- Функция XACT_STATE не может использоваться для определения наличия вложенных транзакций.

Ошибки, возникающие в процессе обработки транзакций

Если ошибка делает невозможным успешное выполнение транзакции, SQL Server автоматически выполняет ее откат и освобождает ресурсы, удерживаемые транзакцией. Если сетевое соединение клиента с SQL Server разорвано, то после того, как SQL Server получит уведомление от сети о разрыве соединения, выполняется откат всех необработанных транзакций для этого соединения. В случае сбоя клиентского приложения, выключения либо перезапуска клиентского компьютера соединение также будет разорвано, а SQL Server выполнит откат всех необработанных транзакций после получения уведомления о разрыве от сети. Если клиент выйдет из приложения, выполняется откат всех необработанных транзакций.

В случае ошибки (нарушения ограничения целостности) во время выполнения инструкции в пакете по умолчанию SQL Server выполнит откат только той инструкции, которая привела к ошибке. Это поведение можно изменить с помощью инструкции SET XACT_ABORT. После выполнения инструкции SET XACT_ABORT ON любая ошибка во время выполнения инструкции приведет к автоматическому откату текущей транзакции.

На случай возникновения ошибок код приложения должен содержать исправляющее действие: COMMIT или ROLLBACK. Эффективным средством для обработки ошибок, включая ошибки транзакций, является конструкция языка Transact-SQL TRY...CATCH.

Пример

```
USE MyDB;
GO
IF OBJECT_ID ( N'dbo.SaveTranExample', N'P' ) IS NOT NULL DROP PROCEDURE dbo.SaveTranExample;
GO

CREATE PROCEDURE dbo.SaveTranExample ... -- Список параметров AS
--  Надо проверить, была ли процедура вызвана из активной транзакции, и если да,
--  то устанавить точку сохранения для последующего использования.
--  @TranCounter = 0 означает, что активной транзакции нет и процедура явно начинает
--  локальную транзакцию.
--   @TranCounter > 0 означает, что активная транзакция началась еще до вызова процедуры.
DECLARE @TranCounter INT;
SET @TranCounter = @@TRANCOUNT;
IF @TranCounter > 0
        SAVE TRANSACTION ProcedureSave;
ELSE
        BEGIN TRANSACTION;
-- Пытаемся модифицировать базу данных.
BEGIN TRY
-- INSERT, UPDATE, DELETE
```

```
...
-- Здесь мы окажемся, если не произойдет никакой ошибки.
-- Если транзакция началась внутри процедуры, то выполнить COMMIT TRANSACTION.
-- Если транзакция началась до вызова процедуры, то выполнять COMMIT TRANSACTION нельзя.
IF @TranCounter = 0
-- @TranCounter = 0 означает, что никакая транзакция до вызова процедуры не начиналась. -- Процедура должна завершить начатую в ней транзакцию.
        COMMIT TRANSACTION;
END TRY
BEGIN CATCH
-- Здесь мы окажемся, если произошла ошибка.
-- Надо определить, какой тип отката транзакции применять.
IF @TranCounter = 0
-- Транзакция началась в теле процедуры. -- Полный откат транзакции.
        ROLLBACK TRANSACTION;
            ELSE

-- Транзакция началась еще до вызова процедуры.
-- Нельзя отменять изменения, сделанные до вызова процедуры.
IF XACT_STATE() <> -1
-- Если транзакция активна, то выполнить откат до точки сохранения.
ROLLBACK TRANSACTION ProcedureSave;
-- Если транзакция актива, однако произошла ошибка, из-за которой транзакция
-- классифицируется как нефиксируемая,
-- то вернуться в точку вызова процедуры, где должен произойти откат внешней транзакции.
-- В точку вызова процедуры передается информация об ошибках.
DECLARE @ErrorMessage NVARCHAR(4000); DECLARE @ErrorSeverity INT;
DECLARE @ErrorState INT;
SELECT @ErrorMessage = ERROR_MESSAGE(); SELECT @ErrorSeverity = ERROR_SEVERITY(); SELECT @ErrorState = ERROR_STATE();
RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState); END CATCH
GO
```

Пояснения к примеру

Для получения сведений об ошибках в области блока CATCH конструкции TRY...CATCH можно использовать следующие системные функции:

- ERROR_LINE() - возвращает номер строки, в которой произошла ошибка.
- ERROR_MESSAGE() - возвращает текст сообщения, которое будет возвращено приложению. Текст содержит значения таких подставляемых параметров, как длина, имена объектов или время.
- ERROR_NUMBER() - возвращает номер ошибки.
- ERROR_PROCEDURE() - возвращает имя хранимой процедуры или триггера, в котором произошла ошибка. Эта функция возвращает значение NULL, если данная ошибка не была совершена внутри хранимой процедуры или триггера.
- ERROR_SEVERITY() - возвращает уровень серьезности ошибки.
- ERROR_STATE() - возвращает состояние.

Инструкция RAISERROR позволяет вернуть приложению сообщение в формате системных ошибок или предупреждений.

Управлением параллельным выполнением транзакций

Когда множество пользователей одновременно пытаются модифицировать данные в базе данных, необходимо создать систему управления, которая защитила бы модификации, выполненные одним пользователем, от негативного воздействия модификаций, сделанных другими. Выделяют два типа управления параллельным выполнением:

- **Пессимистическое** управление параллельным выполнением.
- **Оптимистическое** управление параллельным выполнением.

Пессимистическое управление реализуется с помощью технологии блокировок, оптимистическое управления реализуется с помощью технологии версии строк.

Система блокировок не разрешает пользователям выполнить модификации, влияющие на других пользователей. Если пользователь выполнил какое-либо действие, в результате которого установлена блокировка, то другим пользователям не удастся выполнять действия, конфликтующие с установленной блокировкой, пока владелец не освободит ее. Пессимистическое управление используется главным образом в средах, где высока конкуренция за данные.

В случае управления на основе версий строк пользователи не блокируют данные при чтении. Во время обновления система следит, не изменил ли другой пользователь данные после их прочтения. Если другой пользователь модифицировал данные, генерируется ошибка. Как правило, получивший ошибку пользователь откатывает транзакцию и повторяет операцию снова. Этот способ управления в основном используется в средах с низкой конкуренцией за данные.

SQL Server поддерживает различные механизмы оптимистического и пессимистического управления параллельным выполнением. Пользователю предоставляется право определить тип управления параллельным выполнением, установив уровень изоляции транзакции для соединения и параметры параллельного выполнения для курсоров.

Если в случае пессимистического управления в СУБД не реализованы механизмы блокирования, то при одновременном чтении и изменении одних и тех же данных несколькими пользователями могут возникнуть следующие проблемы одновременного доступа:

проблема последнего изменения возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна, т.к. каждая последующая транзакция перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;

проблема "грязного" чтения (Dirty Read) возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;

проблема неповторяемого чтения (Non-repeatable or Fuzzy Read) является следствием неоднократного считывания транзакцией одних и тех же данных. Во время выполнения первой транзакции другая может внести в данные изменения, поэтому при повторном чтении первая транзакция получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности;

проблема чтения фантомов (Phantom) появляется после того, как одна транзакция выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

Для решения перечисленных проблем в стандарте ANSI SQL определены четыре уровня блокирования. Уровень изоляции транзакции определяет, могут ли другие (конкурирующие) транзакции вносить изменения в данные, измененные текущей транзакцией, а также может ли текущая транзакция видеть изменения, произведенные конкурирующими транзакциями, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

уровень 0 – запрещение «загрязнения» данных. Этот уровень требует, чтобы изменять данные могла только одна транзакция; если другой транзакции необходимо изменить те же данные, она должна ожидать завершения первой транзакции;

уровень 1 – запрещение «грязного» чтения. Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать их до завершения первой;

уровень 2 – запрещение неповторяемого чтения. Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии;

уровень 3 – запрещение фантомов. Если транзакция обращается к данным, то никакая другая транзакция не сможет добавить новые или удалить имеющие строки, которые могут быть считаны при выполнении транзакции. Реализация этого уровня блокирования выполняется путем использования блокировок диапазона ключей. Подобная блокировка накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому условию.

Проблемы, связанные с параллельным выполнением транзакций, разрешают, используя уровни изоляции транзакции. От уровня изоляции зависит то, в какой степени транзакция влияет на другие транзакции и испытывает влияние других транзакций. Более низкий уровень изоляции увеличивает возможность параллельного выполнения, но за это приходится расплачиваться согласованностью данных. Напротив, более высокий уровень изоляции гарантирует согласованность данных, но при этом страдает параллельное выполнение.

Стандарт ISO определяет следующие уровни изоляции:

- **read uncommitted** (самый низкий уровень, при котором транзакции изолируются до такой степени, чтобы только уберечь от считывания физически поврежденных данных);
- **read committed** (уровень по умолчанию);
- изоляция повторяющегося чтения **repeatable read**;
- изоляция упорядочиваемых транзакций **serializable** (самый высокий уровень, при котором транзакции полностью изолированы друг от друга).

SQL Server также поддерживает еще два уровня изоляции транзакций, использующих управление версиями строк.

- **read committed** с использованием управления версиями строк;
- уровень изоляции моментальных снимков **snapshot**.

Следующая таблица показывает побочные эффекты параллелизма, допускаемые различными уровнями изоляции.

Уровень изоляции	**«Грязное» чтение**	**Неповторяющееся чтение**	**Фантомное чтение**
read uncommitted	Да	Да	Да
read committed	Нет	Да	Да
repeatable read	Нет	Нет	Да
snapshot	Нет	Нет	Нет
serializable	Нет	Нет	Нет

Для установки уровня изоляции используется следующая инструкция SET TRANSACTION ISOLATION LEVEL, имеющая следующий синтаксис:

```
SET TRANSACTION ISOLATION LEVEL
        { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SNAPSHOT |
          SERIALIZABLE }
[ ;]
```

Описание аргументов

READ UNCOMMITTED - указывает, что инструкции могут считывать строки, которые были изменены другими транзакциями, но еще не были зафиксированы.

READ COMMITTED - указывает, что инструкции не могут считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы. Это предотвращает чтение «грязных» данных. Данные могут быть изменены другими транзакциями между отдельными инструкциями в текущей транзакции, результатом чего будет неповторяемое чтение или недействительные данные.

Напоминание. Поведение READ COMMITTED зависит от настройки аргумента базы данных READ_COMMITTED_SNAPSHOT (находится в состоянии OFF по умолчанию). REPEATABLE READ - указывает на то, что инструкции не могут считывать данные, которые были изменены, но еще не зафиксированы другими транзакциями, а также на то, что другие транзакции не могут изменять данные, читаемые текущей транзакцией, до ее завершения.

SNAPSHOT - указывает на то, что данные, считанные любой инструкцией транзакции, будут согласованы на уровне транзакции с версией данных, существовавших в ее начале. Транзакция распознает только те изменения, которые были зафиксированы до ее начала. Инструкции, выполняемые текущей транзакцией, не видят изменений данных, произведенных другими транзакциями после запуска текущей транзакции. Таким образом достигается эффект получения инструкциями в транзакции моментального снимка зафиксированных данных на момент запуска транзакции. Перед запуском транзакции, использующей уровень изоляции моментальных снимков, необходимо установить параметр базы данных ALLOW_SNAPSHOT_ISOLATION в ON. Если транзакция с уровнем изоляции моментального снимка обращается к данным из нескольких баз данных, аргумент ALLOW_SNAPSHOT_ISOLATION должен быть включен в каждой базе данных.

SERIALIZABLE - указывает следующее:

- Инструкции не могут считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы.
- Другие транзакции не могут изменять данные, считываемые текущей транзакцией, до ее завершения.
- Другие транзакции не могут вставлять новые строки со значениями ключа, которые входят в диапазон ключей, считываемых инструкциями текущей транзакции, до ее завершения.

Замечания.

- Одновременно может быть установлен только один параметр уровня изоляции, который продолжает действовать для текущего соединения до тех пор, пока не будет явно изменен.
- Уровни изоляции транзакции определяют тип блокировки, применяемый к операциям считывания.
- В любой момент транзакции можно переключиться с одного уровня изоляции на другой, однако есть одно исключение. Это смена уровня изоляции на уровень изоляции SNAPSHOT. Такая смена приводит к ошибке и откату транзакции. Однако для транзакции, которая была начата с уровнем изоляции SNAPSHOT, можно установить любой другой уровень изоляции.
- Когда для транзакции изменяется уровень изоляции, ресурсы, которые считываются после изменения, защищаются в соответствии с правилами нового уровня. Ресурсы, которые считываются до изменения, остаются защищенными в соответствии с правилами предыдущего уровня. Например, если для транзакции уровень изоляции изменяется с READ COMMITTED на SERIALIZABLE, то совмещаемые блокировки, полученные после изменения, будут удерживаться до завершения транзакции.
- Если инструкция SET TRANSACTION ISOLATION LEVEL использовалась в хранимой процедуре или триггере, то при возврате управления из них уровень изоляции будет изменен на тот, который действовал на момент их вызова. Например, если уровень изоляции REPEATABLE READ устанавливается в пакете, а пакет затем вызывает хранимую процедуру, которая меняет уровень изоляции на SERIALIZABLE, при возвращении хранимой процедурой управления пакету, настройки уровня изоляции меняются назад на REPEATABLE READ.
- Определяемые пользователем функции и типы данных среды CLR не могут выполнять инструкцию SET TRANSACTION ISOLATION LEVEL. Однако уровень изоляции можно переопределить с помощью табличной подсказки.

Блокировки

Блокировка — это механизм, с помощью которого компонент Database Engine синхронизирует одновременный доступ нескольких пользователей к одному фрагменту данных. Прежде чем транзакция сможет распоряжаться текущим состоянием фрагмента данных, например, для чтения или изменения данных, она должна защититься от изменений этих данных другой транзакцией. Для этого транзакция запрашивает блокировку фрагмента данных. Существует несколько режимов блокировки, например общая или монопольная. Режим блокировки определяет уровень подчинения данных транзакции. Ни одна транзакция не может получить блокировку, которая противоречит другой блокировке этих данных, предоставленной другой транзакции. Если транзакция запрашивает режим блокировки, противоречащий предоставленной ранее блокировке тех же данных, экземпляр компонента Database Engine приостанавливает ее работу до тех пор, пока первая блокировка не освободится.

При изменении фрагмента данных транзакция удерживает блокировку, защищая изменения до конца транзакции. Продолжительность блокировки, полученной для защиты операций чтения, зависит от уровня изоляции транзакции. Все блокировки, удерживаемые транзакцией, освобождаются после ее завершения (при фиксации или откате).
Приложения обычно не запрашивают блокировку напрямую. За управление блокировками отвечает внутренний компонент Database Engine, называемый диспетчером блокировок. Когда экземпляр компонента Database Engine обрабатывает инструкцию Transact-SQL, обработчик запросов компонента Database Engine определяет, к каким ресурсам требуется доступ. Обработчик запросов определяет, какие типы блокировок требуются для защиты каждого ресурса, в зависимости от типа доступа и уровня изоляции транзакции. Затем обработчик запросов запрашивает соответствующую блокировку у диспетчера блокировок. Диспетчер блокировок предоставляет блокировку, если она не противоречит блокировкам, удерживаемым другими транзакциями. Использование блокировки как механизма управления транзакциями может разрешить проблемы параллелизма. Блокировка позволяет запускать все транзакции в полной изоляции друг от друга, что позволяет одновременно выполнять несколько транзакций. Уровень, на котором транзакция готова к принятию противоречивых данных, называется уровнем изоляции. Чем выше уровень изоляции, тем ниже вероятность непоследовательности данных, однако при этом появляется такой недостаток, как сокращение параллелизма.

Каждая блокировка обладает тремя свойствами: гранулярностью (или размером блокировки), режимом (или типом блокировки) и продолжительностью.

Гранулярность блокировок и иерархии блокировок

SQL Server поддерживает мультигранулярную блокировку, позволяющую транзакции блокировать различные типы ресурсов. Чтобы уменьшить издержки применения блокировок, компонент Database Engine автоматически блокирует ресурсы на соответствующем задаче уровне. Блокировка при меньшей гранулярности, например на уровне строк, увеличивает параллелизм, но в то же время увеличивает и накладные расходы на обработку, поскольку при большом количестве блокируемых строк требуется больше блокировок. Блокировки на большем уровне гранулярности, например на уровне таблиц, обходится дорого в отношении параллелизма, поскольку блокировка целой таблицы ограничивает доступ ко всем частям таблицы других транзакций. Однако накладные расходы в этом случае ниже, поскольку меньше количество поддерживаемых блокировок.

Компонент Database Engine часто получает блокировки на нескольких уровнях гранулярности одновременно, чтобы полностью защитить ресурс. Такая группа блокировок на нескольких уровнях гранулярности называется иерархией блокировки. Например, чтобы полностью защитить операцию чтения индекса, экземпляру компоненту Database Engine может потребоваться получить разделяемые блокировки на строки и намеренные разделяемые блокировки на страницы и таблицу.

Следующая таблица содержит перечень ресурсов, которые могут блокироваться компонентом Database Engine.

Ресурс	Описание
RID	Идентификатор строки, используемый для блокировки одной строки в куче
KEY	Блокировка строки в индексе, используемая для защиты диапазонов значений ключа в сериализуемых транзакциях.
PAGE	8-килобайтовая (КБ) страница в базе данных, например страница данных или индекса.
EXTENT	Упорядоченная группа из восьми страниц, например страниц данных или индекса.
HOBT	Куча или сбалансированное дерево. Блокировка, защищающая индекс или кучу страниц данных в таблице, не имеющей кластеризованного индекса.
TABLE	Таблица полностью, включая все данные и индексы.
FILE	Файл базы данных.
APPLICATION	Определяемый приложением ресурс.
METADATA	Блокировки метаданных.
ALLOCATION_UNIT	Единица размещения.
DATABASE	База данных, полностью.

По умолчанию блокировки укрупняются с уровня отдельных строк до целых страниц даже до уровня таблицы из соображений повышения производительности. Хотя в общем случае укрупнение считается хорошей штукой, оно может создавать проблемы, например, когда один SPID блокирует всю таблицу, препятствуя другому SPID работать ней. Можно настроить параметры блокировки на уровне строк и страниц. Такие блокировки по умолчанию разрешены для индексов.

Режимы блокировки

SQL Server блокирует ресурсы с помощью различных режимов блокировки, которые определяют доступ одновременных транзакций к ресурсам. В следующей таблице показаны режимы блокировки ресурсов, применяемые компонентом Database Engine.

Режим блокировки	Описание
Совмещаемая блокировка (S)	Совмещаемая Используется для операций считывания, которые не меняют и не обновляют данные, такие как инструкция SELECT.
Блокировка обновления (U)	Применяется к тем ресурсам, которые могут быть обновлены. Предотвращает возникновение распространенной формы взаимоблокировки, возникающей тогда, когда несколько сеансов считывают, блокируют и затем, возможно, обновляют ресурс.
Монопольная блокировка (X)	Используется для операций модификации данных, таких как инструкции INSERT, UPDATE или DELETE. Гарантирует, что несколько обновлений не будет выполнено одновременно для одного ресурса.
Блокировка с намерением	Используется для создания иерархии блокировок. Типы намеренной блокировки: с намерением совмещаемого доступа (IS), с намерением монопольного доступа (IX), а также совмещаемая с намерением монопольного доступа (SIX).
Блокировка схемы	Используется во время выполнения операции, зависящей от схемы таблицы. Типы блокировки схем: блокировка изменения схемы (Sch-S) и блокировка стабильности схемы (Sch-M).
Блокировка массового обновления (BU)	Используется, если выполняется массовое копирование данных в таблицу и указана подсказка TABLOCK.
Диапазон ключей	Защищает диапазон строк, считываемый запросом при использовании уровня изоляции сериализуемой транзакции. Запрещает другим транзакциям вставлять строки, что помогает запросам сериализуемой транзакции уточнять, были ли запросы запущены повторно.

Совмещаемые блокировки. Совмещаемые (S) блокировки позволяют одновременным транзакциям считывать (SELECT) ресурс под контролем пессимистичного параллелизма. Пока для ресурса существуют совмещаемые (S) блокировки, другие транзакции не могут изменять данные. Совмещаемые блокировки (S) ресурса снимаются по завершении операции считывания, если только уровень изоляции транзакции не установлен на повторяющееся чтение или более высокий уровень, а также если совмещаемые блокировки (S) не продлены на все время транзакции с помощью указания блокировки.

Блокировки обновления. Блокировки обновления (U) предотвращают возникновение распространенной формы взаимоблокировки. В сериализуемой транзакции или транзакции операцией чтения с возможностью повторения транзакция считывает данные, запрашивает совмещаемую (S) блокировку на ресурс (страницу или строку), затем выполняет изменение данных, что требует преобразование блокировки в монопольную (X). Если две транзакции запрашивают совмещаемую блокировку на ресурс и затем пытаются одновременно обновить

данные, то одна из транзакций пытается преобразовать блокировку в монопольную (X). Преобразование совмещаемой блокировки в монопольную потребует некоторого времени, поскольку монопольная блокировка для одной транзакции несовместима с совмещаемой блокировкой для другой транзакции. Начнется ожидание блокировки. Вторая транзакция попытается получить монопольную (X) блокировку для обновления. Поскольку обе транзакции выполняют преобразование в монопольную (X) блокировку и при этом каждая из транзакций ожидает, пока вторая снимет совмещаемую блокировку, то в результате возникает взаимоблокировка.
Чтобы избежать этой потенциальной взаимоблокировки, применяются блокировки обновления (U). Блокировку обновления (U) может устанавливать для ресурса одновременно только одна транзакция. Если транзакция изменяет ресурс, то блокировка обновления (U) преобразуется в монопольную (X) блокировку.

Монопольные блокировки. Монопольная (X) блокировка запрещает транзакциям одновременный доступ к ресурсу. Если ресурс удерживается монопольной (X) блокировкой, то другие транзакции не могут изменять данные. Операции считывания будут допускаться только при наличии указания NOLOCK или уровня изоляции незафиксированной операции чтения. Изменяющие данные инструкции, такие как INSERT, UPDATE или DELETE, соединяют как операции изменения, так и операции считывания. Чтобы выполнить необходимые операции изменения данных, инструкция сначала получает данные с помощью операций считывания. Поэтому, как правило, инструкции изменения данных запрашивают как совмещаемые, так и монопольные блокировки. Например, инструкция UPDATE может изменять строки в одной таблице, основанной на соединении данных из другой таблицы. В этом случае инструкция UPDATE кроме монопольной блокировки обновляемых строк запрашивает также совмещаемые блокировки для строк, считываемых в соединенной таблице.

Блокировки с намерением. В компоненте Компонент Database Engine блокировки с намерением применяются для защиты размещения совмещаемой (S) или монопольной (X) блокировки ресурса на более низком уровне иерархии. Блокировки с намерением называются так потому, что их получают до блокировок более низкого уровня, то есть они обозначают намерение поместить блокировку на более низком уровне.

Блокировка с намерением выполняет две функции:

- предотвращает изменение ресурса более высокого уровня другими транзакциям таким образом, что это сделает недействительной блокировку более низкого уровня;
- повышает эффективность компонента Компонент Database Engine при распознавании конфликтов блокировок на более высоком уровне гранулярности.

Например, в таблице требуется блокировка с намерением совмещаемого доступа до того, как для страниц или строк этой таблицы будет запрошена совмещаемая (S) блокировка. Если задать блокировку с намерением на уровне таблицы, то другим транзакциям будет запрещено получать монопольную (X) блокировку для таблицы, содержащей эту страницу.

Блокировка с намерением повышает производительность, поскольку компонент Компонент Database Engine проверяет наличие таких блокировок только на уровне таблицы, чтобы определить, может ли транзакция безопасно получить для этой таблицы совмещаемую блокировку. Благодаря этому нет необходимости проверять блокировки в каждой строке и на каждой странице, чтобы убедиться, что транзакция может заблокировать всю таблицу.

Блокировки схем. В компоненте Компонент Database Engine блокировка изменения схемы (Sch-M) применяется с операциями языка DDL для таблиц, например при добавлении столбца или очистке таблицы. Пока удерживается блокировка изменения схемы (Sch-M),

одновременный доступ к таблице запрещен. Это означает, что любые операции вне блокировки изменения схемы (Sch-M) будут запрещены до снятия блокировки.

- Блокировка изменения схемы (Sch-M) применяется с некоторыми операциями языка обработки данных, например усечением таблиц, чтобы предотвратить одновременный доступ к таблице.
- Блокировка стабильности схемы (Sch-S) применяется компонентом Компонент Database Engine при компиляции и выполнении запросов. Блокировка стабильности схемы (Sch-S) не влияет на блокировки транзакций, включая монопольные (X) блокировки. Поэтому другие транзакции (даже транзакции с монопольной блокировкой (X) для таблицы) могут продолжать работу во время компиляции запроса. Однако одновременные операции DDL и DML, которые запрашивают блокировки изменения схемы (Sch-M), не могут выполняться над таблицей.

Блокировки массового обновления. Блокировка массового обновления (BU) позволяет поддерживать несколько одновременных потоков массовой загрузки данных в одну и ту же таблицу и при этом запрещать доступ к таблице любым другим процессам, отличным от массовой загрузки данных. Компонент Database Engine использует блокировки массового обновления (BU), если выполняются два следующих условия. Используется инструкция Transact-SQL BULK INSERT, функция OPENROWSET(BULK) или одна из таких команд массовой вставки API, как .NET SqlBulkCopy, OLEDB Fast Load APIs или ODBC Bulk Copy APIs, для массового копирования данных в таблицу. Выделено указание TABLOCK или установлен параметр таблицы table lock on bulk load с помощью хранимой процедуры sp_tableoption.

Блокировки диапазона ключа. Блокировки диапазона ключей защищают диапазон строк, неявно включенный в набор записей, считываемый инструкцией Transact-SQL при использовании уровня изоляции сериализуемых транзакций. Блокировка диапазона ключей предотвращает фантомные чтения. Кроме того, защита диапазона ключей между строк предотвращает фантомную вставку или удаление из набора записи, к которому получает доступ транзакция.

Совместимость блокировок

Совместимость блокировок определяет, могут ли несколько транзакций одновременно получить блокировку одного и того же ресурса. Если ресурс уже блокирован другой транзакцией, новая блокировка может быть предоставлена только в том случае, если режим запрошенной блокировки совместим с режимом существующей. В противном случае транзакция, запросившая новую блокировку, ожидает освобождения ресурса, пока не истечет время ожидания существующей блокировки.
Например, с монопольными блокировками не совместим ни один из режимов блокировки. Пока удерживается монопольная (X) блокировка, больше ни одна из транзакций не может получить блокировку ни одного из типов (разделяемую, обновления или монопольную) на этот ресурс, пока не будет освобождена монопольная (X) блокировка. И наоборот, если к ресурсу применяется разделяемая (S) блокировка, другие транзакции могут получать разделяемую блокировку или блокировку обновления (U) на этот элемент, даже если не завершилась первая транзакция. Тем не менее, другие транзакции не могут получить монопольную блокировку до освобождения разделяемой. Полная матрица совместимости блокировок приводится в справочниках.

Продолжительность блокировки

Продолжительность блокировок также определяется типами запросов. Когда в рамках транзакции запрос не выполняется, и не используются подсказки блокировки, блокировки для выполнения инструкций SELECT выполняются только на время чтения ресурса, но не во время запроса. Блокировки для инструкций INSERT, UPDATE и DELETE сохраняются на все время выполнения запроса. Это помогает гарантировать согласованность данных и позволяет SQL Server откатывать запросы в случае необходимости.
Когда запрос выполняется в рамках транзакции, продолжительность блокировки определяется тремя факторами:

- типом запроса;
- уровнем изоляции транзакции;
- наличием или отсутствием подсказок блокировки.

Кратковременные (locking) и обычные (blocking) блокировки — нормальное явление в реляционных базах данных, но они могут ухудшать производительность, если блокировки ресурсов сохраняются на протяжении длительного времени. Производительность также страдает, когда, заблокировав ресурс, SPID не в состоянии освободить его.
В первом случае проблема обычно разрешается через какое-то время, так как SPID, в конечном счете, освобождает блокировку, но угроза деградации производительности остается вполне реальной. Проблемы с блокировкой второго типа могут вызвать серьезное падение производительности, но, к счастью, они легко обнаруживаются при мониторинге SQL Server на предмет кратковременных и обычных блокировок.

Эскалация блокировок и их влияние на работу системы

Эскалация (lock escalation) связана с тем, что по мере увеличения, количества отдельных малых заблокированных объектов накладные расходы, связанные с их поддержкой, начинают значительно сказываться на производительности. Блокировки длятся дольше, что приводит к спорным ситуациям – чем дольше существует блокировка, тем выше вероятность обращения к заблокированному объекту со стороны другой транзакции. Очевидно, что на некотором этапе потребуется выполнить объединение (увеличение масштаба) блокировок, чем собственно и занимается диспетчер блокировок. В инструкции ALTER TABLE предусмотрена опция вида SET (LOCK_ESCALATION = { AUTO | TABLE | DISABLE }), которая указывает разрешенные методы укрупнения блокировки для таблицы.

Задание определенного типа блокировки в запросе

Для повышения эффективности исполнения запросов и получения дополнительного контроля над блокировками в запросе можно давать подсказки (hints или хинты), указывая их непосредственно за именем таблицы, которая нуждается в том или ином типе блокировки. Существуют следующие параметры оптимизатора запросов:
SERIALIZABLE/HOLDLOCK
READUNCOMMITTED/NOLOCK
READCOMMITTED
REPEATABLEREAD
READPAST
ROWLOCK
PAGLOCK
TABLOCK
TABLOCKX
UPDLOCK

XLOCK

Пример задания эксклюзивной табличной блокировки для таблицы Orders (вместо блокировки на уровне ключей или строк, которую может предложить оптимизатор) может быть таким:

```
SELECT *
FROM Orders AS o WITH (TABLOCKX) JOIN [Order Details] AS od
        ON o.OrderID = od.OrderID
```

Поскольку оптимизатор запросов обычно выбирает лучший план выполнения запроса, использовать подсказки рекомендуется только опытным разработчикам и администраторам баз данных в самом крайнем случае. Сведения об активных блокировках на текущий момент времени можно получить с помощью системной хранимой процедуры sp_lock.

Взаимоблокировки транзакций

Взаимоблокировки или тупиковые ситуации (deadlocks) возникают тогда, когда одна из транзакций не может завершить свои действия, поскольку вторая транзакция заблокировала нужные ей ресурсы, а вторая в то же время ожидает освобождения ресурсов первой транзакцией. На рисунке транзакция T1 зависит от транзакции T2 для ресурса блокировки таблицы Детали. Аналогично транзакция T2 зависит от транзакции T1 для ресурса блокировки таблицы Поставщик. Так как эти зависимости из одного цикла, возникает взаимоблокировка транзакций T1 и T2.

Транзакция T1 не может завершиться до того, как завершится транзакция T2, а транзакция T2 заблокирована транзакцией T1. Такое условие также называется цикличной зависимостью: транзакция T1 зависит от транзакции T2, а транзакция T2 зависит от транзакции T1 и этим замыкает цикл. Обе транзакции находятся в состоянии взаимоблокировки и будут всегда находиться в состоянии ожидания, если взаимоблокировка не будет разрушена внешним процессом.

Взаимоблокировки часто путают с обычными блокировками. Если транзакция запрашивает блокировку на ресурс, заблокированный дугой транзакцией, то запрашивающая транзакция ожидает до тех пор, пока блокировка не освобождается. По умолчанию время ожидания транзакций SQL Server не ограничено, если только не установлен параметр LOCK_TIMEOUT.

Значение -1 (по умолчанию) указывает на отсутствие времени ожидания (то есть инструкция будет ждать всегда). Когда ожидание блокировки превышает значение времени ожидания, возвращается ошибка. Значение «0» означает, что ожидание отсутствует, а сообщение

возвращается, как только встречается блокировка. Функция @@LOCK_TIMEOUT возвращает значение времени ожидания блокировки в миллисекундах для текущего сеанса.

Запрашивающая транзакция блокируется, но не устанавливается в состояние взаимоблокировки, потому что запрашивающая транзакция ничего не сделала, чтобы заблокировать транзакцию, владеющую блокировкой. Наконец, владеющая транзакция завершится и освободит блокировку, и затем запрашивающая транзакция получит блокировку и продолжится.

Как SQL Server обнаруживает взаимоблокировки?

Обнаружение взаимоблокировки выполняется потоком диспетчера блокировок, который периодически производит поиск по всем задачам в экземпляре компонента Database Engine. Следующие пункты описывают процесс поиска:

- Значение интервала поиска по умолчанию составляет 5 секунд.
- Если диспетчер блокировок находит взаимоблокировки, интервал обнаружения взаимоблокировок снижается с 5 секунд до 100 миллисекунд в зависимости от частоты взаимоблокировок.
- Если поток диспетчера блокировки прекращает поиск взаимоблокировок, компонент Database Engine увеличивает интервал до 5 секунд.
- Если взаимоблокировка была только что найдена, предполагается, что следующие потоки, которые должны ожидать блокировки, входят в цикл взаимоблокировки. Первая пара элементов, ожидающих блокировки, после того как взаимоблокировка была обнаружена, запускает поиск взаимоблокировок вместо того, чтобы ожидать следующий интервал обнаружения взаимоблокировки. Например, если текущее значение интервала равно 5 секунд и была обнаружена взаимоблокировка, следующий ожидающий блокировки элемент немедленно приводит в действие детектор взаимоблокировок. Если этот ожидающий блокировки элемент является частью взаимоблокировки, она будет обнаружена немедленно, а не во время следующего поиска взаимоблокировок.
- Компонент Database Engine обычно выполняет только периодическое обнаружение взаимоблокировок. Так как число взаимоблокировок, произошедших в системе, обычно мало, периодическое обнаружение взаимоблокировок помогает сократить издержки от взаимоблокировок в системе.
- Если монитор блокировок запускает поиск взаимоблокировок для определенного потока, он идентифицирует ресурс, ожидаемый потоком. После этого монитор блокировок находит владельцев определенного ресурса и рекурсивно продолжает поиск взаимоблокировок для этих потоков до тех пор, пока не найдет цикл. Цикл, определенный таким способом, формирует взаимоблокировку.
- После обнаружения взаимоблокировки компонент Database Engine завершает взаимоблокировку, выбрав один из потоков в качестве жертвы взаимоблокировки. Компонент Database Engine прерывает выполняемый в данный момент пакет потока, производит откат транзакции жертвы взаимоблокировки и возвращает приложению ошибку 1205. Откат транзакции жертвы взаимоблокировки снимает все блокировки, удерживаемые транзакцией. Это позволяет транзакциям потоков разблокироваться, и продолжить выполнение. Ошибка 1205 жертвы взаимоблокировки записывает в журнал ошибок сведения обо всех потоках и ресурсах, затронутых взаимоблокировкой.

Как выбирается жертва взаимоблокировки?

По умолчанию в качестве жертвы взаимоблокировки выбирается сеанс, выполняющий ту транзакцию, откат которой потребует меньше всего затрат. В качестве альтернативы

пользователь может указать приоритет сеансов, используя инструкцию SET DEADLOCK_PRIORITY. DEADLOCK_PRIORITY может принимать значения LOW, NORMAL или HIGH или в качестве альтернативы может принять любое целочисленное значение на отрезке [-10..10]. По умолчанию DEADLOCK_PRIORITY устанавливается на значение NORMAL. Если у двух сеансов имеются различные приоритеты, то в качестве жертвы взаимоблокировки будет выбран сеанс с более низким приоритетом. Если у обоих сеансов установлен одинаковый приоритет, то в качестве жертвы взаимоблокировки будет выбран сеанс, откат которого потребует наименьших затрат. Если сеансы, вовлеченные в цикл взаимоблокировки, имеют один и тот же приоритет и одинаковую стоимость, то жертва взаимоблокировки выбирается случайным образом.

Журнализация

Базы данных SQL Server содержат файлы трех типов:

- **Первичные файлы данных**. Первичный файл данных является отправной точкой базы данных. Он указывает на остальные файлы базы данных. В каждой базе данных имеется один первичный файл данных. Для имени первичного файла данных рекомендуется использовать расширение MDF.

- **Вторичные файлы данных.** Ко вторичным файлам данных относятся все файлы данных, за исключением первичного файла данных. Некоторые базы данных могут вообще не содержать вторичных файлов данных, тогда как другие содержат несколько вторичных файлов данных. Для имени вторичного файла данных рекомендуется использовать расширение NDF.

- **Файлы журналов.** Файлы журналов содержат все сведения журналов, используемые для восстановления базы данных. В каждой базе данных должен быть, по меньшей мере, один файл журнала, но их может быть и больше. Для имен файлов журналов рекомендуется использовать расширение LDF. Журнал транзакций нельзя ни удалять, ни изменять, если только не известны возможные последствия. Кэш журнала управляется отдельно от буферного кэша для страниц данных.

Журнал транзакций поддерживает следующие операции:

- **Восстановление отдельных транзакций.** Если приложение выполняет инструкцию ROLLBACK или если компонент Database Engine обнаруживает ошибку, такую как потеря связи с клиентом, записи журнала используются для отката изменений, сделанных незавершенной транзакцией.

- **Восстановление всех незавершенных транзакций при запуске SQL Server**. Если на сервере, где работает SQL Server, происходит сбой, базы данных могут остаться в таком состоянии, в котором часть изменений не была переписана из буферного кэша в файлы данных, и могут быть изменения в файлах данных, совершенные незаконченными транзакциями. Когда экземпляр SQL Server будет запущен, он выполнит восстановление каждой базы данных. Будет выполнен накат каждого записанного в журнал изменения, которое, возможно, не было переписано в файл данных. Чтобы сохранить целостность базы данных, будет также произведен откат каждой незавершенной транзакции, найденной в журнале транзакций.

- **Накат восстановленной базы данных, файла**, файловой группы или страницы до момента сбоя. После потери оборудования или сбоя диска, затрагивающего файлы базы данных, можно восстановить базу данных на момент, предшествующий сбою. Сначала восстановите последнюю полную резервную копию и последнюю дифференциальную резервную копию базы данных, затем восстановите последующую серию резервных копий журнала транзакций до момента возникновения сбоя. Поскольку восстанавливается каждая резервная копия журнала, компонент Database Engine повторно применяет все модификации, записанные в журнале, для наката всех транзакций. Когда последняя резервная копия журнала будет восстановлена, тогда компонент Database Engine начнет

использовать данные журнала для отката всех транзакций, которые не были завершены на момент сбоя.

- **Поддержка репликации транзакций.** Агент чтения журнала следит за журналами транзакций всех баз данных, которые настроены на репликацию транзакций, и копирует отмеченные для репликации транзакции из журнала транзакций в базу данных распространителя. Дополнительные сведения см. в разделе Как работает репликация транзакций.

- **Поддержка решений с резервными серверами**. Решения резервного сервера, зеркальное отображение базы данных и доставка журналов в значительной степени полагаются на журнал транзакций. В сценарии доставки журналов основной сервер отправляет активный журнал транзакций основной базы данных одному или более адресатам. Каждый сервер-получатель восстанавливает журнал в свою локальную базу данных-получатель.

Операции резервного копирования и восстановления выполняются в контексте модели восстановления. Модель восстановления — это свойство базы данных, которое управляет процессом регистрации транзакций, определяет, требуется ли для журнала транзакций резервное копирование, а также определяет, какие типы операций восстановления доступны. Есть три модели восстановления:

- **Модель полного восстановления (FULL)**. Обеспечивает модель обслуживания для баз данных, в которых необходима поддержка длительных транзакций. Требуются резервные копии журналов. При использовании этой модели выполняется полное протоколирование всех транзакций, и сохраняются записи журнала транзакций до момента их резервного копирования. Модель полного восстановления позволяет восстановить базу данных до точки сбоя при условии, что после сбоя возможно создание резервной копии заключительного фрагмента журнала. Кроме того, модель полного восстановления поддерживает восстановление отдельных страниц данных.

- **Модель восстановления с неполным протоколированием (BULK_LOGGED).** Эта модель восстановления обеспечивает неполное протоколирование большинства массовых операций. Она предназначена для работы только в качестве дополнения к полной модели полного восстановления. Для ряда масштабных массовых операций (массовый импорт, создание индекса и т.п.) временное переключение на модель восстановления с неполным протоколированием повышает производительность и уменьшает место, необходимое для журналов. Тем не менее, для работы этой модели требуются резервные копии журналов. Как и в модели полного восстановления, в модели восстановления с неполным протоколированием сохраняются записи журнала транзакций после его резервного копирования. Это увеличивает объем резервных копий журналов и повышает риск потери результатов работы, поскольку модель восстановления с неполным протоколированием не поддерживает восстановление до заданного момента времени.

- **Простая модель восстановления (SIMPLE).** Сводит к минимуму административные затраты, связанные с журналом транзакций, поскольку его резервная копия не создается. При использовании простой модели восстановления в случае повреждения базы данных возникает риск потери значительной части результатов работы. Данные могут быть восстановлены только до момента последнего резервного копирования. Поэтому при использовании простой модели восстановления интервалы между резервным

копированием должны быть достаточно короткими, чтобы предотвратить потерю значительного объема данных. В то же время они должны быть велики настолько, чтобы затраты на резервное копирование не влияли на производительность. Снизить затраты поможет использование разностного резервного копирования.

Обычно в базе данных используется модель полного восстановления или простая модель восстановления. Модели восстановления оказывают влияние на поведение журнала транзакций и способ регистрации операций, либо на и на то, и на другое.

Модель восстановления FULL подразумевает, что регистрируется каждая часть каждой операции, и это называется полной регистрацией. После выполнения полного резервного копирования базы данных в модели восстановления FULL в журнале транзакций не будет проводиться автоматическое усечение до тех пор, пока не будет выполнено резервное копирование журнала. Если вы не намерены использовать резервные копии журнала и возможность восстановления состояния базы данных на конкретный момент времени, не следует использовать модель восстановления FULL. Однако, если вы предполагаете использовать зеркальное отображение базы данных, тогда у вас нет выбора, поскольку оно поддерживает только модель восстановления FULL.

Модель восстановления BULK_LOGGED обладает такой же семантикой усечения журнала транзакций, как и модель восстановления FULL, но допускает частичную регистрацию некоторых операций, что называется минимальной регистрацией. Примерами являются повторное создание индекса и некоторые операции массовой загрузки — в модели восстановления FULL регистрируется вся операция.

Но в модели восстановления BULK_LOGGED регистрируются только изменения распределения, что радикально сокращает число создаваемых записей журнала и, в свою очередь, сокращает потенциал разрастания журнала транзакций.

Модель восстановления SIMPLE, фактически ведет себя с точки зрения ведения журнала так же, как и модель восстановления BULK_LOGGED, но имеет совершенно другую семантику усечения журнала транзакций. В модели восстановления SIMPLE невозможны резервные копии журнала, что означает, что журнал может быть усечен (если ничто не удерживает записи журнала в активном состоянии) при возникновении контрольной точки.

SQL Server поддерживает восстановление данных на следующих уровнях:

- **База данных** (полное восстановление базы данных). Вся база данных возвращается в прежнее состояние и восстанавливается, при этом база данных находится в автономном режиме во время операций возврата и восстановления.

- **Файл данных** (восстановление файла). Файл данных или набор файлов данных возвращается в исходное состояние и восстанавливается. Во время восстановления файлов файловые группы, содержащие обрабатываемые файлы, автоматически переводятся в автономный режим на время восстановления. Любые попытки подключения и работы с недоступной файловой группой приведут к ошибке.

- **Страница данных** (восстановление страницы). При использовании модели полного восстановления или модели восстановления с неполным протоколированием можно

восстановить отдельные базы данных. Восстановление страниц может применяться для любой базы данных вне зависимости от числа файловых групп.

Логическая архитектура журнала транзакций

На логическом уровне журнал транзакций состоит из последовательности записей. Каждая запись содержит:

- Log Sequence Number: регистрационный номер транзакции (LSN). Каждая новая запись добавляется в логический конец журнала с номером LSN, который больше номера LSN предыдущей записи.
- Prev LSN: обратный указатель, который предназначен для ускорения отката транзакции.
- Transaction ID number: идентификатор транзакции.
- Type: тип записи Log-файла.
- Other information: Прочая информация.

Двумя основными типами записи Log-файла являются:

- Transaction Log Operation Code: код выполненной логической операции, либо
- Update Log Record: исходный и результирующий образ измененных данных. Исходный образ записи – это копия данных до выполнения операции, а результирующий образ – копия данных после ее выполнения.

Действия, которые необходимо выполнить для восстановления операции, зависят от типа Log-записи:

- Зарегистрирована логическая операция.
  - Для наката логической операции выполняется эта операция.
  - Для отката логической операции выполняется логическая операция, обратная зарегистрированной.
- Зарегистрированы исходный и результирующий образы записи.
  - Для наката операции применяется результирующий образ.
  - Для отката операции применяется исходный образ.

В журнал транзакций записываются различные типы операций:
0 BEGINXACT: Begin Transaction
4 Insert
5 Delete
6 Indirect Insert
7 Index Insert
8 Index Delete
9 MODIFY: Modify the record on the page
11 Deferred Insert (NO-OP)
12 Deferred Delete (NO-OP)
13 Page Allocation
15 Extent Allocation
16 Page Split
17 Checkpoint
20 DEXTENT
30 End Transaction (Either a commit or rollback)
38 CHGSYSINDSTAT — A change to the statistics page in sysindexes
39 CHGSYSINDPG — Change to a page in the sysindexes table

Каждая транзакция резервирует в журнале транзакций место, чтобы при выполнении инструкции отката или возникновения ошибки в журнале было достаточно места для регистрации отката. Объем резервируемого пространства зависит от выполняемых в транзакции операций, но обычно он равен объему, необходимому для регистрации каждой из операций. Все это пространство после завершения транзакции освобождается.

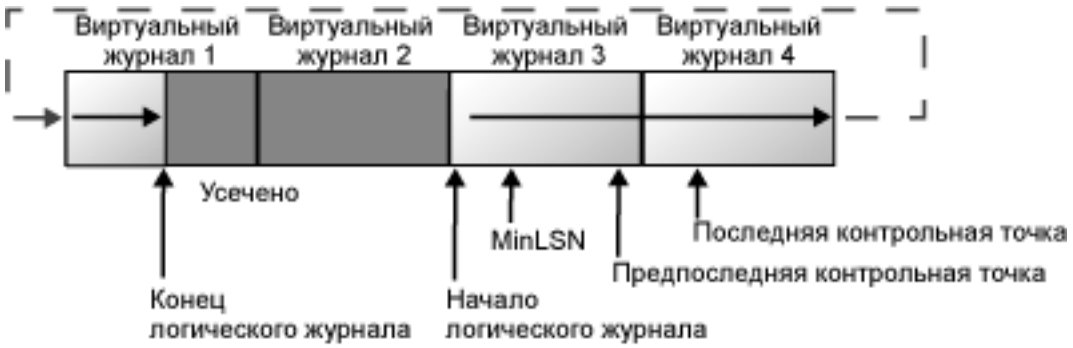
Раздел журнального файла, который начинается от первой записи, необходимой для успешного отката на уровне базы данных, до последней зарегистрированной записи называется активной частью журнала, или активным журналом. Именно этот раздел необходим для выполнения полного восстановления базы данных. Активный журнал не может быть усечен.

Физическая архитектура журнала транзакций

На физическом уровне журнал транзакций состоит из одного или нескольких физических файлов. Каждый физический файл журнала разбивается на несколько виртуальных файлов журнала (VLF). VLF не имеет фиксированного размера. Не существует также и определенного числа VLF, приходящихся на один физический файл журнала. Компонент Database Engine динамически определяет размер VLF при создании или расширении файлов журнала. Компонент Database Engine стремится обслуживать небольшое число VLF. Администраторы не могут настраивать или устанавливать размеры и число VLF.

Журнал транзакций является оборачиваемым файлом. Рассмотрим пример. Пусть база данных имеет один физический файл журнала, разделенный на четыре виртуальных файла журнала. При создании базы данных логический файл журнала начинается в начале физического файла журнала. Новые записи журнала добавляются в конце логического журнала и приближаются к концу физического файла журнала. Усечение журнала освобождает любые виртуальные журналы, все записи которых находятся перед минимальным регистрационным номером восстановления в журнале транзакций (MinLSN). MinLSN является регистрационным номером самой старой записи, которая необходима для успешного отката на уровне всей базы данных. Журнал транзакций рассматриваемой в данном примере базы данных будет выглядеть примерно так же, как на следующей иллюстрации.

Когда конец логического журнала достигнет конца физического файла журнала, новые записи журнала будут размещаться в начале физического файла журнала.



Этот цикл повторяется бесконечно, пока конец логического журнала не совмещается с началом этого логического журнала. Если старые записи журнала усекаются достаточно часто, так что при этом всегда остается место для новых записей журнала, созданных с новой контрольной точки, журнал постоянно остается незаполненным. Однако, если конец логического журнала совмещается с началом этого логического журнала, происходит одно из двух событий, перечисленных ниже:

- Если для данного журнала применена установка FILEGROWTH и на диске имеется свободное место, файл расширяется на величину, указанную в growth_increment, и новые записи журнала добавляются к этому расширению. Дополнительные сведения о настройке FILEGROWTH см. в разделе ALTER DATABASE (Transact- SQL).

- Если установка FILEGROWTH не применяется или диск, на котором размещается файл журнала, имеет меньше свободного места, чем это указано в growth_increment, формируется ошибка 9002.

- Если в журнале содержится несколько физических файлов журнала, логический журнал будет продвигаться по всем физическим файлам журнала до тех пор, пока он не вернется на начало первого физического файла журнала.

Просмотр журнала транзакций
Вариант А. Недокументированная команда DBCC LOG(имя_базы_данных, тип_вывода)
где тип_вывода принимает значение из диапазона 0..4. По умолчанию принят 0 (Current LSN, Operation, Context, Transaction ID). 3 выдает полный информационный дамп каждой операции.
Вариант В. Недокументированная функция fn_dblog().
Пример.

```
create table t1 (id int, name varchar(20));

insert into t1 values (1, 'test1');
checkpoint;
insert into t1 values (2, 'test2');
checkpoint;

select * from t1;
drop table t1;

-- DBCC LOG(N'dbtest', 3)
-- Выводится таблица 45*102
-- select top 1 * from fn_dblog(null, null)
-- Выводится таблица 1*97
```

Так как все изменения страниц данных происходят в страничных буферах, то изменения данных в памяти не обязательно отражаются в этих страницах на диске. Процесс кэширования происходит по алгоритму последней использованной страницы, поэтому страница, подверженная постоянным изменениям, помечается как последняя использованная, и она не записывается его на диск. Чтобы эти страницы были записаны на диск применяется контрольная точка. Все грязные страницы должны быть сохранены на диске в обязательном порядке.

Контрольная точка выполняет в базе данных следующее:

- Записывает в файл журнала запись, отмечающую начало контрольной точки.

- Сохраняет данные, записанные для контрольной точки в цепи записей журнала контрольной точки. Одним из элементов данных, регистрируемых в записях контрольной точки, является номер LSN первой записи журнала, при отсутствии которой успешный откат в масштабе всей базы данных невозможен. Такой номер LSN называется минимальным номером LSN восстановления (MinLSN). Номер MinLSN является наименьшим значением из:
    - номера LSN начала контрольной точки;
    - номера LSN начала старейшей активной транзакции;
    - номера LSN начала старейшей транзакции репликации, которая еще не была доставлена базе данных распространителя.
    - Записи контрольной точки содержат также список активных транзакций, изменивших базу данных.
- Если база данных использует простую модель восстановления, помечает для повторного использования пространство, предшествующее номеру MinLSN.
- Записывает все измененные страницы журналов и данных на диск.
- Записывает в файл журнала запись, отмечающую конец контрольной точки.
- Записывает в страницу загрузки базы данных номер LSN начала соответствующей цепи.

Действия, приводящие к срабатыванию контрольных точек

Контрольные точки срабатывают в следующих ситуациях:
- При явном выполнении инструкции CHECKPOINT. Контрольная точка срабатывает в текущей базе данных соединения.
- При выполнении в базе данных операции с минимальной регистрацией, например при выполнении операции массового копирования для базы данных, на которую распространяется модель восстановления с неполным протоколированием.
- При добавлении или удалении файлов баз данных с использованием инструкции ALTER DATABASE.
- При остановке экземпляра SQL Server с помощью инструкции SHUTDOWN или при остановке службы SQL Server (MSSQLSERVER). И в том, и в другом случае будет создана контрольная точка каждой базы данных в экземпляре SQL Server.
- Если экземпляр SQL Server периодически создает в каждой базе данных автоматические контрольные точки для сокращения времени восстановления базы данных.
- При создании резервной копии базы данных.

- При выполнении действия, требующего отключения базы данных. Примерами могут служить присвоение параметру AUTO_CLOSE значения ON и закрытие последнего соединения пользователя с базой данных или изменение параметра базы данных, требующее перезапуска базы данных.

Автоматические контрольные точки

Компонент Database Engine создает контрольные точки автоматически. Интервал между автоматическими контрольными точками определяется на основе использованного места в журнале и времени, прошедшего с момента создания последней контрольной точки. Интервал между автоматическими контрольными точками колеблется в широких пределах и может быть довольно длительным, если база данных изменяется редко. При крупномасштабных изменениях данных частота автоматических контрольных точек может быть гораздо выше.
Можно использовать параметр конфигурации сервера recovery interval для вычисления интервала между автоматическими контрольными точками для всех баз данных на экземпляре сервера. Значение этого параметра определяет максимальное время, отводимое компоненту Database Engine на восстановление базы данных при перезапуске системы. Компонент Database Engine оценивает количество записей журнала, которые он может обработать за время recovery interval при выполнении операции восстановления.
Если используется простая модель восстановления базы данных, автоматическая контрольная точка создается каждый раз, когда число записей в журнале достигает меньшего из двух предельных условий:
- журнал заполняется на 70 процентов;
- число записей в журнале достигает значения, определенного компонентом Database Engine в качестве количества записей, которое он может обработать за время, заданное параметром recovery interval.

Активный журнал

Часть журнала, начинающаяся с номера MinLSN и заканчивающаяся последней записью, называется активной частью журнала, или активным журналом. Этот раздел журнала необходим для выполнения полного восстановления базы данных. Ни одна часть активного журнала не может быть усечена. Все записи журнала до номера MinLSN должны быть удалены из частей журнала.

На следующем рисунке изображена упрощенная схема журнала завершения транзакций, содержащего две активные транзакции. Записи контрольных точек были сжаты в одну запись.

Последней записью в журнале транзакций является запись с номером LSN, равным 148. На момент обработки записанной контрольной точки с номером LSN 147 транзакция 1 уже зафиксирована и единственной активной транзакцией является транзакция 2. В результате первая запись журнала, созданная для транзакции 2, становится старейшей записью активной транзакции на момент последней контрольной точки. Таким образом, номером MinLSN становится номер LSN, равный 142 и соответствующий записи начала транзакции 2.

Длительные транзакции

Активный журнал должен включать в себя все элементы всех незафиксированных транзакций. Приложение, инициирующее транзакцию и не выполняющее ее фиксацию или откат, не позволяет компоненту Database Engine повышать MinLSN.

Это может привести к проблемам двух типов.

- Если система будет выключена после того, как транзакцией было выполнено много незафиксированных изменений, этап восстановления при последующем перезапуске может занять гораздо больше времени, чем указано параметром recovery interval.
- Журнал может достичь очень большого объема, потому что после номера MinLSN усечь его нельзя. Это справедливо даже в том случае, если используется простая модель восстановления, когда журнал транзакций обычно усекается при каждой автоматической контрольной точке.

Журнал транзакций с упреждающей записью

SQL Server использует журнал с упреждающей записью, который гарантирует, что до занесения на диск записи, связанной с журналом, никакие изменения данных записаны не будут. Таким образом обеспечиваются свойства ACID для транзакции. Для понимания принципов работы журнала с упреждающей записью важно знать принципы записи измененных данных на диск. SQL Server поддерживает буферный кэш, из которого система считывает страницы данных при извлечении необходимых данных. Изменения данных не заносятся непосредственно на диск, а записываются на копии страницы в буферном кэше. Изменение не записывается на диск, пока в базе данных не возникает контрольная точка, или же изменение должно быть записано на диск таким образом, чтобы для хранения новой страницы мог использоваться буфер. Запись измененной страницы данных из буферного кэша на диск называется сбросом страницы на диск. Страница, измененная в кэше, но еще не записанная на диск, называется грязной страницей.

Во время изменения страницы в буфере запись журнала строится в кэше журнала, который записывает изменение. Данная запись журнала должна быть перенесена на диск до того, как соответствующая «грязная» страница будет записана из буферного кэша на диск. Если «грязная» страница переносится на диск до записи журнала, эта страница создает изменение на диске, которое не может быть откачено, если сервер выйдет из строя до переноса записи журнала на диск. SQL Server обладает алгоритмом, который защищает «грязную» страницу от записи на диск до переноса на него соответствующей записи журнала. Содержимое журнала запишется на диск после того, как будут зафиксированы транзакции.

Управление журналом транзакций

Чтобы логический журнал не увеличивался до размера физических файлов журнала, следует периодически выполнять его усечение. Процесс усечения журнала уменьшает размер файла

логического журнала, помечая виртуальные файлы журнала, которые не содержат частей логического журнала, как неактивные. В некоторых случаях может оказаться полезным физическое сжатие или расширение размера реального файла журнала.

Время усечения журнала зависит от модели восстановления базы данных. Есть три модели восстановления: простая модель восстановления, модель полного восстановления и модель восстановления с неполным протоколированием. Обычно в базе данных используется полная модель восстановления или простая модель восстановления. В качестве примера рассмотрим усечение журнала в простой модели восстановления.

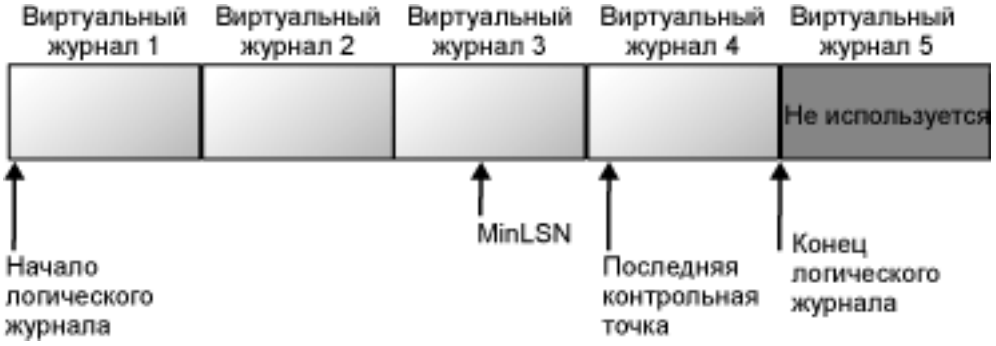
При использовании простой модели восстановления усечение журналов выполняется автоматически. Если все записи в виртуальном файле журнала неактивны, то этот логический журнал усекается обычно после контрольной точки. При этом освобождается место для повторного использования. Это относится и к контрольным точкам инструкции CHECKPOINT, и к неявным контрольным точкам, сформированным системой. Однако усечение журнала может быть отложено, если виртуальные файлы журнала остаются активными вследствие выполнения долгой транзакции или резервного копирования.

Эта модель регистрирует только минимальные сведения, необходимые для обеспечения согласованности базы данных после сбоя системы или для восстановления данных из резервной копии. Это сводит к минимуму расход места на диске под журнал транзакций по сравнению с другими моделями восстановления. Чтобы предотвратить переполнение журнала, базе данных требуется достаточно места для записи в случае задержки его усечения.

Как работает усечение журнала

Кроме прочих данных, в контрольной точке записывается номер LSN первой записи журнала, которую необходимо сохранить для успешного отката на уровне базы данных. Этот номер LSN называется минимальным номером LSN восстановления (MinLSN). Начало активной части журнала занято VLF, содержащим MinLSN. При усечении журнала транзакций освобождаются только те записи, которые находятся перед этим VLF.

На следующем рисунке показан журнал транзакций до усечения и после. На первом рисунке показан журнал транзакций, который никогда не усекался. В настоящий момент логический журнал состоит из четырех виртуальных файлов. Логический журнал начинается с начала первого файла виртуального журнала и заканчивается виртуальным файлом журнала 4. Запись MinLSN находится в виртуальном журнале 3. Виртуальные журналы 1 и 2 содержат только неактивные записи журнала. Эти записи можно усечь. Виртуальный журнал 5 пока не используется и не является частью текущего логического журнала.



На втором рисунке показан журнал после усечения. Виртуальные журналы 1 и 2 усечены и могут использоваться повторно. Логический журнал теперь начинается с начала виртуального журнала 3. Виртуальный журнал 5 все еще не используется и не является частью текущего логического журнала.

Управление размером файла журнала транзакций

Контролировать используемое пространство журнала можно с помощью процедуры DBCC SQLPERF (LOGSPACE). Она возвращает сведения об объеме пространства, используемого журналом в данный момент, и указывает, если необходимо провести усечение журнала транзакций. Для получения сведений о текущем размере файла журнала, его максимальном размере и параметре автоматического увеличения файла можно использовать столбцы size, max_size и growth для данного файла журнала в представлении sys.database_files.

Сжатие файла журнала

Усечение журнала освобождает место на диске для повторного использования, но не уменьшает размер физического файла журнала. Для уменьшения физического размера файл журнала должен быть сжат с целью удаления одного или более неактивных VLF. VLF, хранящий какие-либо активные записи журнала, удалить нельзя. При сжатии файла журнала транзакций в конце файла журнала удаляется достаточное количество неактивных виртуальных файлов журнала, чтобы журнал уменьшился до приблизительного целевого размера.

Примечание. Если журнал транзакций не усекался в последнее время, его сжатие может быть невозможным до тех пор, пока не выполнится усечение.

Система безопасности SQL Server

SQL Server обеспечивает защиту данных от неавторизированного доступа и от фальсификации. Основными функциями безопасности SQL Server являются:

- **проверка подлинности (аутентификация)** – процедура проверки соответствия некоего лица и его учетной записи в компьютерной системе. Один из способов аутентификации состоит в задании пользовательского идентификатора, в просторечии называемого «логином» (login – регистрационное имя пользователя) и пароля – некой конфиденциальной информации, знание которой обеспечивает владение определенным ресурсом. Аутентификацию не следует путать с идентификацией. Идентификация – это установление личности самого физического лица (а не его виртуальной учетной записи, коих может быть много).

- **авторизация** – это предоставление лицу прав на какие-то действия в системе.

Дополнительными функциями безопасности SQL Server являются:
- шифрование,
- контекстное переключение,
- олицетворение,
- встроенные средства управления ключами.

В основе системы безопасности SQL Server лежат три понятия:
- участники системы безопасности (Principals);
- защищаемые объекты (Securables) и
- система разрешений (Permissions).

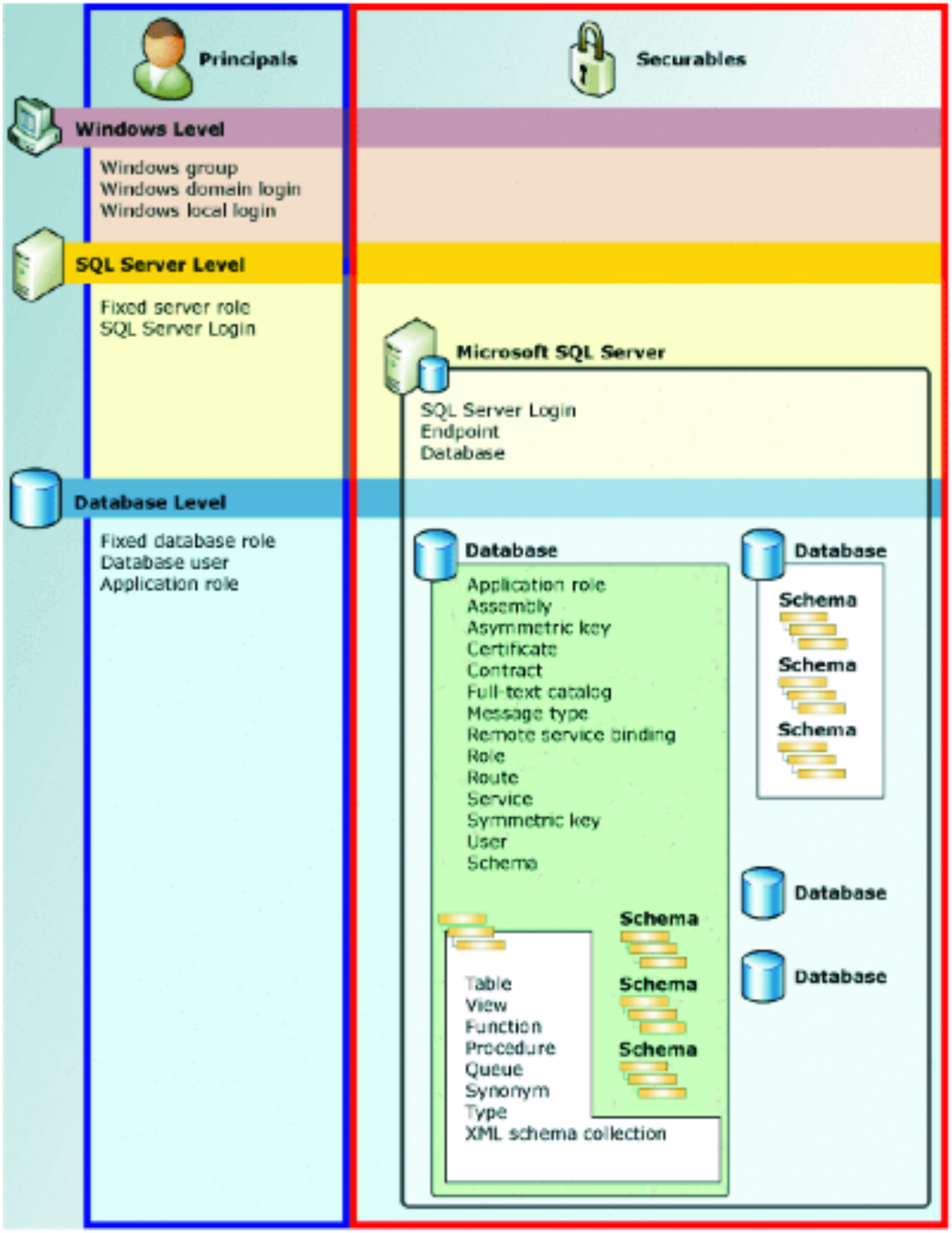
Участники системы безопасности

Участники системы безопасности или принципалы – это сущности, которые могут запрашивать ресурсы SQL Server. Принципалы могут быть иерархически упорядочены. Область влияния принципала зависит
- от области его определения: Windows, SQL Server, база данных,
- от того, коллективный это участник или индивидуальный. Имя входа Windows является примером индивидуального (неделимого) участника, а группа Windows — коллективного.

При создании учетной записи SQL Server ей назначается идентификатор и идентификатор безопасности. В представлении каталога sys.server_principals они отображаются в столбцах principal_id и SID.
Участники уровня Windows – это

A. Имя входа домена Windows,

B. Локальное имя входа Windows. Участники уровня SQL Server – это a) Имя входа SQL Server, b) Роль сервера.
   Участники уровня базы данных – это a) Пользователь базы данных, b) Роль базы данных, c) Роль приложения.


Principals
Securables
Windows Level
Windows group
Windows domain login
Windows local login
SQL Server Level
Fixed server role
SQL Server Login
Microsoft SQL Server
SQL Server Login
Endpoint
Database
Database Level
Fixed database role
Database user
Application role
Database
Application role
Assembly
Asymmetric key
Certificate
Contract
Full-text catalog
Message type
Remote service binding
Role
Route
Service
Symmetric key
User
Schema
Schema
Schema
Schema
Table
View
Function
Procedure
Queue
Synonym
Type
XML schema collection
Database
Schema
Schema
Schema
Database
Database

Замечания.

1. Имя входа «sa» SQL Server. Имя входа sa SQL Server является участником уровня сервера. Оно создается по умолчанию при установке экземпляра. В SQL Server для имени входа sa базой данных по умолчанию будет master.

2. Роль базы данных public. Каждый пользователь базы данных является членом роли базы данных public. Если пользователю не были предоставлены или запрещены особые разрешения на защищаемый объект, то он наследует на него разрешения роли public.

3. Пользователи INFORMATION_SCHEMA и sys. Каждая база данных включает в себя две сущности, которые отображены в представлениях каталога в виде пользователей: INFORMATION_SCHEMA и sys. Они необходимы для работы SQL Server; эти пользователи не являются участниками и не могут быть изменены или удалены.

Защищаемые объекты

Защищаемые объекты – это ресурсы, доступ к которым регулируется системой авторизации. Некоторые защищаемые объекты могут храниться внутри других, создавая иерархии «областей», которые сами могут защищаться. К областям защищаемых объектов относятся (a) сервер, (b) база данных и (c) схема.

Далее введем следующие ограничения:
1. для области «сервер» ограничимся рассмотрением вопросами управления учетными записями подключения (login),
2. для области «база данных» ограничимся рассмотрением вопросами управления
    1. учетными записями пользователей (user),
    2. ролями (role),
    3. ролями приложений (application role).
3. для области «схема» ограничимся рассмотрением вопросами управления разрешениями на
    1. функции (function),
    2. процедуры (stored procedure),
    3. таблицы (table),
    4. представления (view).

Разрешения

У субъекта системы есть только один путь получения доступа к объектам - иметь назначенные непосредственно или опосредовано разрешения. При непосредственном управлении разрешениями они назначаются субъекту явно, а при опосредованном разрешения назначаются через членство в группах, ролях или наследуются от объектов, лежащих выше по цепочке иерархии. Управление разрешениями производится путем выполнения инструкций языка DCL (Data Control Language): GRANT (разрешить), DENY (запретить) и REVOKE (отменить).

Управление учетными данными сервера (credential)

Поддерживаются два вида учетных записей подключения к серверу
- учетные записи сервера, создаваемые на основании учетных записей операционной системы
- учетные записи сервера, создаваемые для прямого подключения к серверу.

Проверка подлинности Windows означает, что для подключения к SQL Server проверка подлинности полностью выполняется операционной системой Microsoft Windows. В этом

случае клиент идентифицируется на основании учетной записи Windows. Проверка подлинности SQL Server означает, что для подключения к SQL Server проверка подлинности выполняется путем сравнения имени пользователя и пароля с хранящимся на сервере SQL Server списком действительных имен пользователей и паролей). Учетные данные сервера создаются при помощи инструкции CREATE CREDENTIAL. После создания учетных данных можно сопоставить их имени входа SQL Server, используя инструкцию CREATE LOGIN или ALTER LOGIN.

Управление именами входа SQL Server (login)

Сами имена входа не имеют доступа ни к одной конкретной базе данных на сервере, они позволяют только подключиться к SQL Server. Имена входа – это объекты, которым может быть дано разрешение в масштабе сервера на выполнение определенных действий. Эти действия собираются в фиксированные серверные роли: bulkadmin, dbcreator, diskadmin, processadmin, public, securityadmin, serveradmin, setupadmin, sysadmin. Все имена входа SQL Server являются обладателями роли public. Добавление имени входа в качестве члена предопределенной роли сервера выполняется при помощи хранимой процедуры sp_addsrvrolemember. Имя входа сервера может быть создано в SSMS, при помощи инструкции CREATE LOGIN или при помощи хранимой процедуры sp_addlogin.

Замечание. В SQL Server 2012 при помощи инструкции CREATE SERVER ROLE можно создать новую, определяемую пользователем роль сервера, а для изменения членства в роли сервера можно использовать инструкцию ALTER SERVER ROLE.

Примеры.

```
--Creating a new SQL login
CREATE LOGIN Carol
WITH PASSWORD = 'Th1sI$lVlyP@ssw0rd'; GO

--Creating a credential based on a Windows user
CREATE CREDENTIAL StreetCred
WITH IDENTITY = 'AughtFive\CarolStreet', SECRET = 'P@ssw0rd';
--Associating a login with a credential
ALTER LOGIN Carol WITH CREDENTIAL = StreetCred; GO

--Creating a login, and adding the user to a fixed server role
CREATE LOGIN Ted WITH PASSWORD = 'P@ssw0rd'; GO
EXEC sp_addsrvrolemember 'Ted', 'securityadmin'; GO
```

Управление учетными записями пользователей базы данных (user)

После создания имени входа в SQL Server можно предоставить этому имени доступ к конкретной базе данных. Для этого сначала надо создать пользователя базы данных для определенного ранее имени входа. Это можно сделать в SSMS, при помощи инструкции CREATE USER или при помощи хранимой процедуры sp_adduser. При входе в SQL Server под именем входа база данных запросит имя и идентификатор созданного пользователя базы данных. Если при создании пользователя базы данных не будет указано имя входа SQL Server, то новый пользователь базы данных будет сопоставлен с именем входа SQL Server, имеющим такое же имя. После создания пользователя базы данных можно (но это необязательно) предоставить ему одну из фиксированных ролей базы данных: db_accessadmin, db_backupoperator, db_datareader, db_datawriter, db_ddladmin, db_denydatareader,

db_denydatawriter, db_owner, db_securityadmin. При желании можно определить дополнительные роли базы данных при помощи инструкции CREATE ROLE. Для добавления пользователя базы данных к роли текущей базы данных используется инструкция ALTER ROLE или хранимая процедура sp_addrolemember.

Примеры

```
-- Managing Server RoleMembers
CREATE LOGIN Veronica
WITH PASSWORD = 'PalmTree1' GO
EXEC master..sp_addsrvrolemember 'Veronica', 'sysadmin' GO
USE TestDB
GO

CREATE USER Veronica
GO

-- Т. к. предложение FOR LOGIN не указано, то новый пользователь базы данных будет ----
сопоставлен с именем входа SQL Server, имеющим такое же имя.
CREATE ROLE HR_ReportSpecialist AUTHORIZATION db_owner GO
EXEC sp_addrolemember 'HR_ReportSpecialist', 'Veronica' GO
```

Замечание. В SQL Server 2012 многие системные хранимые процедуры, используемые для управления безопасностью, считаются устаревшими, но остаются доступными для обратной совместимости. Для устаревших процедур рекомендуется использовать инструкции Transact-SQL, например, ALTER LOGIN, ALTER SERVER ROLE, ALTER USER, ALTER ROLE и др.

Управление разрешениями

После добавления пользователя следует определить разрешения, управляющие действиями, которые пользователь может выполнять, с помощью инструкций GRANT, DENY и REVOKE. Необходимо, чтобы пользователь был владельцем базы данных.

А. Предоставление разрешений на объекты (инструкция GRANT)

Инструкция GRANT предоставляет разрешения на таблицу, представление, функцию, хранимую процедуру, очередь обслуживания, синоним. Синтаксис инструкции GRANT:

```
GRANT { ALL [ PRIVILEGES ] | список_разрешений } ON список_объектов
TO список_принципалов
[ WITH GRANT OPTION ]
[ AS принципал ]
```

Пояснения к инструкции GRANT

1. Ключевое слово ALL с необязательным словом RIVILEGES не включает все возможные разрешения, оно эквивалентно предоставлению всех разрешений ANSI-92, применимых к указанному объекту. Значение ALL различается для разных типов объектов. Ниже перечислены главные классы разрешений и защищаемых объектов, к которым эти разрешения могут применяться:
    1. разрешения на скалярные функции: EXECUTE, REFERENCES;

2. разрешения на функции, возвращающие табличное значение: DELETE, INSERT, REFERENCES, SELECT, UPDATE;
3. разрешения на хранимые процедуры: EXECUTE;
4. разрешения на таблицы: DELETE, INSERT, REFERENCES, SELECT, UPDATE;
5. разрешения на представления: DELETE, INSERT, REFERENCES, SELECT, UPDATE.

2. Полный список разрешений содержит 195 пунктов.
    1. Если разрешение предоставляется на таблицу, представление или функцию, возвращающую табличное значение, то справа от разрешения в круглых скобоках могут указываться имена столбцов. На столбец могут быть предоставлены только разрешения SELECT, REFERENCES и UPDATE.
    2. Объект, на который предоставляется разрешение, имеет следующее описание: [OBJECT ::] [имя_схемы]. имя_объекта. Фраза OBJECT необязательна, если указан аргумент имя_схемы. Если же она указана, указание квалификатора области (::) обязательно. Если не указан аргумент имя_схемы, подразумевается схема по умолчанию. Если указан аргумент имя_схемы, обязательно указание квалификатора области схемы (.).

3. Принципалом может быть:
    1. пользователь базы данных,
    2. роль базы данных,
    3. роль приложения.

4. Необязательная фраза WITH GRANT OPTION указывает, что принципалу также дается возможность предоставлять указанное разрешение другим принципалам.

5. Необязательная фраза AS принципал определяет принципала, у которого другой принципал, выполняющий данный запрос, наследует право предоставлять данное разрешение.

Пример. Предоставление разрешения EXECUTE на хранимую процедуру HumanResources.uspUpdateEmployeeHireInfo роли приложения Role03.

```
USE AdventureWorks;
GRANT EXECUTE ON OBJECT::HumanResources.uspUpdateEmployeeHireInfo TO Role03;
GO
```

Пример. Предоставление разрешения REFERENCES на столбец EmployeeID в представлении HumanResources.vEmployee пользователю User02 с параметром GRANT OPTION.

```
USE AdventureWorks;
GRANT REFERENCES (EmployeeID) ON OBJECT::HumanResources.vEmployee TO User02
WITH GRANT OPTION;
GO
```

B. Отмена разрешений на объекты (инструкция REVOKE)

Инструкция REVOKE отменяет разрешения, ранее предоставленные инструкцией GRANT. Синтаксис инструкции REVOKE:

```
REVOKE [ GRANT OPTION FOR ] список_разрешений ON список_объектов
{ FROM | TO } список_принципалов
[ CASCADE ]
[ AS принципал ]
```

Пояснения к инструкции REVOKE.

1. Необязательная фраза GRANT OPTION FOR показывает, что право на предоставление заданного разрешения другим принципалам будет отменено. Само разрешение отменено не будет.
2. Необязательное ключевое слово CASCADE показывает, что отменяемое разрешение также отменяется для других принципалов, для которых оно было предоставлено или запрещено этим принципалом. Каскадная отмена разрешения, предоставленного с помощью параметра WITH GRANT OPTION, приведет к отмене прав GRANT и DENY для этого разрешения.
3. Необязательная фраза AS принципал указывает принципала, от которого принципал, выполняющий данный запрос, получает право на отмену разрешения.

Пример. Отмена разрешения EXECUTE для хранимой процедуры

```
USE AdventureWorks;
REVOKE EXECUTE ON OBJECT::HumanResources.uspUpdateEmployeeHireInfo FROM Role03;
GO
```

C. Запрет разрешений на объекты (инструкция DENY)

Инструкция DENY запрещает разрешения на члены класса OBJECT защищаемых объектов. Синтаксис инструкции DENY:

```
DENY список_разрешений ON список_объектов
TO список_принципалов
[ CASCADE ]
[ AS принципал ]
```

Пример. Запрет разрешения REFERENCES на представление с CASCADE

```
USE AdventureWorks;
DENY REFERENCES (EmployeeID) ON OBJECT::HumanResources.vEmployee TO User02 CASCADE; GO
```

Цепочки владения

Если несколько объектов базы данных последовательно обращаются друг к другу, то такая последовательность известна как цепочка. Такие цепочки не могут существовать независимо, но когда SQL Server проходит по звеньям цепи, то SQL Server проверяет разрешения составляющих объектов иначе, нежели при раздельном доступе к объектам. Эти различия имеют важные последствия для обеспечения безопасности. Цепочки владения позволяют управлять доступом к нескольким объектам, таким как таблицы, назначая разрешения одному объекту, например представлению. Цепочки владения также обеспечивают небольшое

повышение производительности в случаях, когда позволено пропускать проверку наличия разрешений. Проверка разрешений в цепи выполняется так:

1. SQL Server сначала сравнивает владельца вызываемого объекта с владельцем вызывающего объекта.

2. Если оба объекта имеют одного владельца, то разрешения для ссылаемого объекта не проверяются.

Пример цепочки владения

1. Алексей выполняет инструкцию SELECT * в представлении July2003. SQL Server проверяет разрешения в представлении и подтверждает, что Алексей имеет разрешение выбирать.

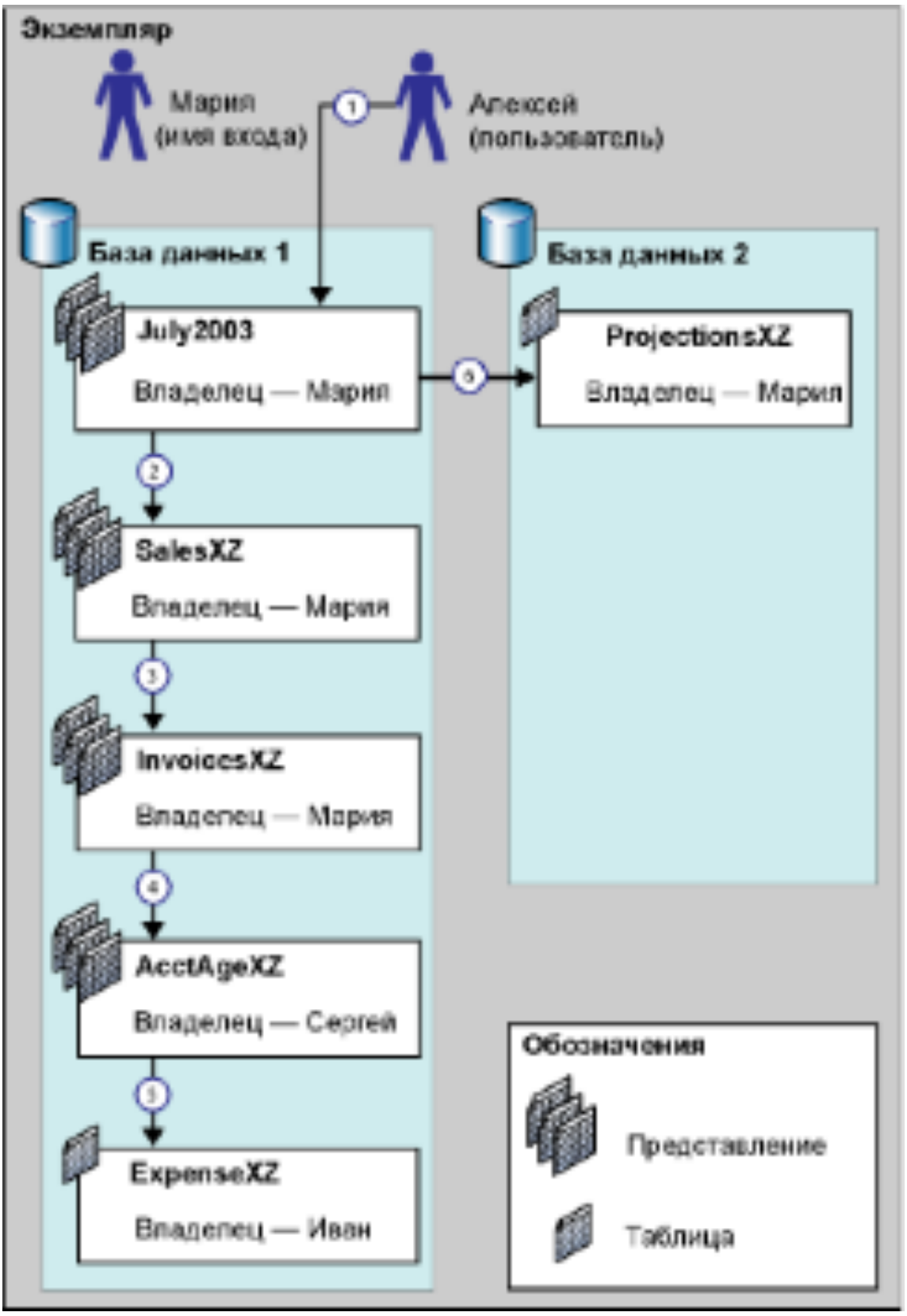
2. Представление July2003 требует данные из представления SalesXZ. SQL Server проверяет владение SalesXZ. Владелец этого представления (Мария) такой же, как у вызывающего представления, поэтому разрешения для SalesXZ не проверяются. Возвращаются необходимые данные.

3. Представление SalesXZ требует данные из представления InvoicesXZ. SQL Server проверяет владение представления SalesXZ. Владелец этого представления такой же, как у предшествующего объекта, поэтому разрешения для InvoicesXZ не проверяются. Возвращаются необходимые данные. До этого этапа все элементы последовательности имели одного владельца (Мария). Это известно как неразрывная цепочка владения.

4. Представление InvoicesXZ требует данные из представления AcctAgeXZ. SQL Server проверяет владение представления AcctAgeXZ. Владелец этого представления иной, чем у предшествующего объекта (Сергей, а не Мария), поэтому должны быть получены полные сведения о разрешениях на это представление. Если на представление AcctAgeXZ имеются разрешения, которые обеспечивают доступ со стороны пользователя Алексей, сведения будут возвращены.

5. Представление AcctAgeXZ требует данные из представления ExpenseXZ. SQL Server проверяет владение таблицы ExpenseXZ. Владелец этой таблицы иной, чем у предшествующего объекта (Иван, а не Сергей), поэтому должны быть получены полные сведения о разрешениях на эту таблицу. Если таблица ExpenseXZ имеет разрешения, которые обеспечивают доступ со стороны пользователя Алексей, сведения возвращаются.

6. Если представление July2003 пытается получить данные из таблицы ProjectionsXZ, то сервер сначала проверяет наличие цепочечных связей между базами данных Database 1 и Database 2. Если цепочечные связи между базами данных активны, то сервер проверяет владение для таблицы ProjectionsXZ. Владелец этой таблицы такой же, как у вызывающего представления (Мэрия), поэтому разрешения для этой таблицы не проверяются. Возвращаются необходимые данные.



По умолчанию межбазовые цепочки владения отключены. SQL Server можно настроить так, чтобы разрешить цепочку владения между конкретными базами данных или между всеми базами данных внутри одного экземпляра SQL Server.

Пример потенциальных опасностей при использовании цепочек владения

Включены межбазовые цепочки владения между базой данных A и базой данных B. В этом случае член предопределенной роли базы данных db_owner любой из этих баз данных может незаконно проникнуть в другую базу данных. Процедура выглядит так:

- Член предопределенной роли базы данных db_owner в базе данных A по имени X создает пользователя Y в базе данных A, который уже существует как пользователь в базе данных B.
- Затем X создает в базе данных A объект, владельцем которого является Y.
- Пользователь Y из базы данных A вызывает любой объект, принадлежащий пользователю Y в базе данных B.
- Вызывающий и вызываемый объекты имеют общего владельца, поэтому разрешения для объекта в базе данных B не будут проверяться, когда X обратится к ним через созданный ею объект. Контекст выполнения и переключение контекста

Контекст выполнения определяется подключенным к сеансу именем входа, пользователем или выполняющимся модулем. Он устанавливает идентификатор пользователя или имени входа, чьи разрешения на выполнение инструкций или совершение действий проверяются. В SQL Server контекст выполнения может переключаться на другое имя входа или пользователя при помощи выполнения инструкции EXECUTE AS или предложения EXECUTE AS в модуле. После переключения контекста SQL Server проверяет разрешения у имени входа и пользователя этой учетной записи, а не у того, кто вызвал инструкцию EXECUTE AS, или модуля. Имя входа или пользователь олицетворяется на оставшееся время выполнения сеанса или модуля либо до того момента, когда происходит явное восстановление переключения контекста (сеанс удаляется, контекст переключен на другое имя входа или на другого пользователя с помощью новой инструкции EXECUTE AS, контекст восстановлен до контекста предыдущего выполнения с помощью инструкции REVERT).

Синтаксис инструкции EXECUTE AS имеет вид:

EXECUTE AS {LOGIN | USER} = 'имя' [WITH {NO REVERT | COOKIE INTO @varbinary_переменная}] | CALLER

- LOGIN указывает, что область олицетворения — это уровень сервера.
- USER указывает, область олицетворения ограничена текущей базой данных.
- 'имя' - допустимое имя входа или имя пользователя.
- NO REVERT указывает, что переключение контекста нельзя вернуть к предыдущему контексту.
- COOKIE INTO @varbinary_переменная указывает, что контекст выполнения можно переключить к предыдущему контексту, если при вызове инструкция REVERT WITH COOKIE содержит правильное значение @varbinary_переменная. SQL Server передает куки-файл в @varbinary_переменная.
- CALLER указывает, что инструкции модуля выполняются в контексте вызывающей стороны. Вне модуля инструкция не действует.

Синтаксис предложения EXECUTE AS имеет вид:

EXECUTE AS { CALLER | SELF | OWNER | 'имя_пользователя' }

- EXECUTE AS CALLER указывает, что инструкции, содержащиеся в модуле, выполняются в контексте пользователя, вызывающего этот модуль.
- EXECUTE AS SELF указывает, что модуль выполняется от имени того пользователя, который последним модифицировал модуль.
- EXECUTE AS OWNER указывает, что инструкции, содержащиеся в модуле, выполняются в контексте текущего владельца этого модуля.
- EXECUTE AS 'имя_пользователя' указывает, что инструкции, содержащиеся в модуле, выполняются в контексте пользователя, указываемого аргументом 'имя_пользователя'.

Пример контекста выполнения

```
USE master
GO
-- Создаем имена входов
CREATE LOGIN User1 WITH PASSWORD='^*ahfn2@^(K' GO
CREATE LOGIN User2 WITH PASSWORD='*HABa7s7aas' GO
CREATE LOGIN User3 WITH PASSWORD='zxd837&^gqF' GO
CREATE DATABASE MyDB
GO
USE MyDB
GO
-- Создаем пользователей и схемы
CREATE USER User3 WITH DEFAULT_SCHEMA=User3
GO
CREATE SCHEMA User3 AUTHORIZATION User3
GO
CREATE USER User2 WITH DEFAULT_SCHEMA=User2
GO
CREATE SCHEMA User2 AUTHORIZATION User2
GO
CREATE USER User1 WITH DEFAULT_SCHEMA=User1
GO
CREATE SCHEMA User1 AUTHORIZATION User1
GO
-- Пользователь User3 имеет право создавать GRANT CREATE TABLE TO User3
GO
-- Пользователь User2 имеет право создавать GRANT CREATE PROC TO User2
GO
EXECUTE AS LOGIN='User3'
GO
CREATE TABLE User3.CustomerInformation
(

    CustomerName nvarchar(50)

)
```

```
GO
INSERT INTO CustomerInformation VALUES('Bryan''s Bowling Alley') INSERT INTO
CustomerInformation VALUES('Tammie''s Tavern') INSERT INTO CustomerInformation
VALUES('Frank''s Fresh Produce') GO
GRANT SELECT ON CustomerInformation TO User2
GO
REVERT
GO
EXECUTE AS LOGIN='User2'
GO
--create a stored proc that will return the rows in our table CREATE PROC ViewCustomerNames
AS

BEGIN
SELECT * FROM User3.CustomerInformation

END GO

GRANT EXECUTE ON ViewCustomerNames TO User1 GO
REVERT
GO

EXECUTE AS LOGIN='User1'
-- User1 cannot access table directly

SELECT * FROM User3.CustomerInformation

таблицу

процедуру

Msg 229, Level 14, State 5, Line 3
The SELECT permission was denied on the object 'CustomerInformation', database 'myDB',
schema 'User3'.

--User1 can execute the procedure but does not have permissions on the underlying table

EXEC User2.ViewCustomerNames

Msg 229, Level 14, State 5, Procedure ViewCustomerNames, Line 5
The SELECT permission was denied on the object 'CustomerInformation', database 'myDB',
schema 'User3'.

GO
REVERT
GO
EXECUTE AS LOGIN='User2'
GO
ALTER PROCEDURE ViewCustomerNames WITH EXECUTE AS OWNER AS

BEGIN
SELECT * FROM User3.CustomerInformation
```

```
END
GO
REVERT
GO
EXECUTE AS LOGIN='User1'
--User1 still cannot access table directly SELECT * from User3.CustomerInformation
```

```
Msg 229, Level 14, State 5, Line 1
The SELECT permission was denied on the object 'CustomerInformation', database 'myDB', schema 'User3'
```

```
--User1 can execute a procedure that uses the CustomerInformation table

EXEC User2.ViewCustomerNames GO
REVERT
GO
```

Аудит SQL Server

Начиная с версии SQL Server 2008 в редакции Enterprise вводится аудит SQL Server Audit – функциональность системы безопасности, которая может отслеживать практически любое действие с сервером или базой данных (выполняемое пользователями) и записывать эти действия в файловую систему или системный журнал Windows.

Система управления на основе политик

Одна из новых возможностей, появившаяся в SQL Server 2008, – система управления на основе политик (Policy-Based Management), которая позволяет создавать политики для обеспечения соответствия нормативам управления базой данных.

Система управления на основе политик позволяет администратору баз данных (АБД) создавать политики для управления объектами и экземплярами базы данных. Эти политики дают АБД возможность устанавливать правила создания и изменения объектов и их свойств. С помощью новой системы можно, например, создать политику уровня БД, запрещающую использование для БД свойства AutoShrink. Другой пример – политика, в соответствии с которой все имена табличных триггеров в таблице БД начинаются с tr_.

Система управления на основе политик предусматривает использование новых терминов и понятий. Основными из них являются:
1. Политика (Policy) – набор условий, определенных аспектами цели управления. Другими словами, политика — это набор правил для свойств БД или серверных объектов.
2. Цель управления (Target) – объект, управляемый данной системой. Сюда относятся такие объекты, как экземпляр БД, база данных, таблица, хранимая процедура, триггер, индекс.
3. Аспект (Facet)– свойство объекта (цели управления), которое используется системой управления на основе политик. Например, имя триггера или свойство базы данных AutoShrink.
4. Условие (Condition) – критерий для аспектов цели управления. Например, можно создать для факта условие, по которому все имена хранимых процедур в схеме «Banking» должны начинаться с bnk_.

Кроме того, политику можно связать с определенной категорией, что позволяет осуществлять управление набором политик, привязанных к той же самой категории. Политика может принадлежать только к одной категории.

Режим оценки политик

Существует несколько режимов оценки политик:
1. По запросу (On demand) – оценку политики запускает непосредственно администратор.
2. При изменении: запретить (On change: prevent) – для предотвращения нарушения политики используются триггеры DDL.
3. При изменении: только внесение в журнал (On change: log only) – для проверки политики при изменении используются уведомления о событии.
4. По расписанию (On schedule) – для проверки политики на нарушения используется задание агента SQL (SQL Agent).

Преимущества системы управления на основе политик

Система управления на основе политик позволяет АБД в полной мере контролировать процессы, происходящие в базе данных. Администратор получает возможность реализовать

принятые в компании политики на уровне БД. Политики, принятые только на бумаге, помогают определить основные принципы управления базой данных и могут служить прекрасным руководством к действию, но воплощать их в жизнь очень нелегко. Для обеспечения строгого соответствия БД принятым нормативом АБД приходится пристально следить за ее повседневным использованием и функционированием. Система управления на основе политик позволяет раз и навсегда выработать политики управления и быть уверенным в том, что они будут применяться постоянно и в полном объеме.

Шифрование

SQL Server поддерживает три механизма шифрования:
1. на основе сертификатов (стандарт X. 509v3),
2. на основе асимметричных ключей (алгоритм RSA),
3. на основе симметричных ключей (алгоритмы шифрования RC4, RC2, DES, AES).

Для работы с этими сущностями используются следующие операторы T-SQL:

CREATE/ALTER/DROP/BACKUP CERTIFICATE CREATE/ALTER/DROP/OPEN/CLOSE ASYMMETRIC KEY CREATE/ALTER/DROP/OPEN/CLOSE SYMMETRIC KEY

Детальное описание иерархической схемы шифрования данных в SQL Server 2005 см. в разделе «Encryption Hierarchy» MSDN.

Контекстное переключение

SQL Server позволяет задать контекст выполнения хранимых процедур и пользовательских функций с помощью выражения EXECUTE AS, помещенного в заголовок определения модуля. Данный механизм позволяет одному пользователю выполнять действия внутри модуля так, будто он аутентифицирован как другой пользователь. Параметр EXECUTE AS имеет четыре возможных значения:

- CALLER. Если программист указывает EXECUTE AS CALLER, операторы внутри модуля выполняются в контексте пользователя, вызвавшего процедуру. Поэтому пользователь, выполняющий процедуру, должен иметь соответствующие разрешения не только на запуск процедуры, но и на любые объекты базы данных, на которые она ссылается.

- USER. Если используется значение EXECUTE AS USER имя_пользователя, процедура выполняется в контексте того пользователя, чье имя указано в параметре. При выполнении процедуры SQL Server сначала проверяет, имеет ли пользователь разрешение EXECUTE на данную процедуру, затем проверяет разрешения на операторы внутри процедуры для пользователя, указанного в параметре EXECUTE AS USER. Для того чтобы иметь возможность указать AS имя_пользователя, необходимо иметь специальные разрешения (например, IMPERSONATE) или быть членом специальной роли (sysadmin или db_owner).

- SELF. EXECUTE AS SELF, аналогично EXECUTE AS USER имя_пользователя, где имя_пользователя является именем человека, создающего или изменяющего модуль. Система сохраняет идентификатор пользователя (UID) вместо самого значения SELF. Пользователь, указанный в параметре SELF, не обязательно должен быть владельцем объекта. На самом деле объекты в SQL Server 2005 не имеют владельцев, но можно думать о владельцах схем так, как будто они владеют всеми объектами в схемах.

- OWNER. Применение EXECUTE AS OWNER хорошо подходит в ситуациях, если пользователь создает хранимые процедуры и таблицы в ходе своей работы, тогда, переходя на другое место, он сможет передать их в собственность другому пользователю, будучи уверенным, что они всегда будут выполняться в контексте безопасности собственника.

Олицетворение

Можно настроить SQL Server и Windows таким образом, чтобы экземпляр SQL Server мог подключаться к другому экземпляру SQL Server в контексте пользователя Windows, прошедшего проверку подлинности. Это называется олицетворением или делегированием. При использовании делегирования экземпляр SQL Server, к которому подключается пользователь Windows с проверкой подлинности Windows, выполняет олицетворение этого пользователя при обмене данными с другим экземпляром SQL Server или поставщиком SQL Server. Этот второй экземпляр или поставщик могут располагаться на том же компьютере или на удаленном компьютере в том же домене Windows, что и первый экземпляр.

Встроенная поддержка прозрачного шифрования

Механизм прозрачного шифрования основан на использовании мастер-ключа, сертификата, защищенного мастер- ключом, и ключа для шифрования базы данных. Для работы с мастер-ключом используются следующие операторы T- SQL:

CREATE/ALTER/DROP/OPEN/CLOSE MASTER KEY

Прозрачное шифрование позволяет создавать индексы и выполнять поиск по зашифрованным даным без использования кких-либо дополнительных функций.

Распределенные вычисления. Hadoop

Hadoop не так уж сложен, ядро состоит из файловой системы HDFS и MapReduce фреймворка для обработки данных из этой файловой системы.

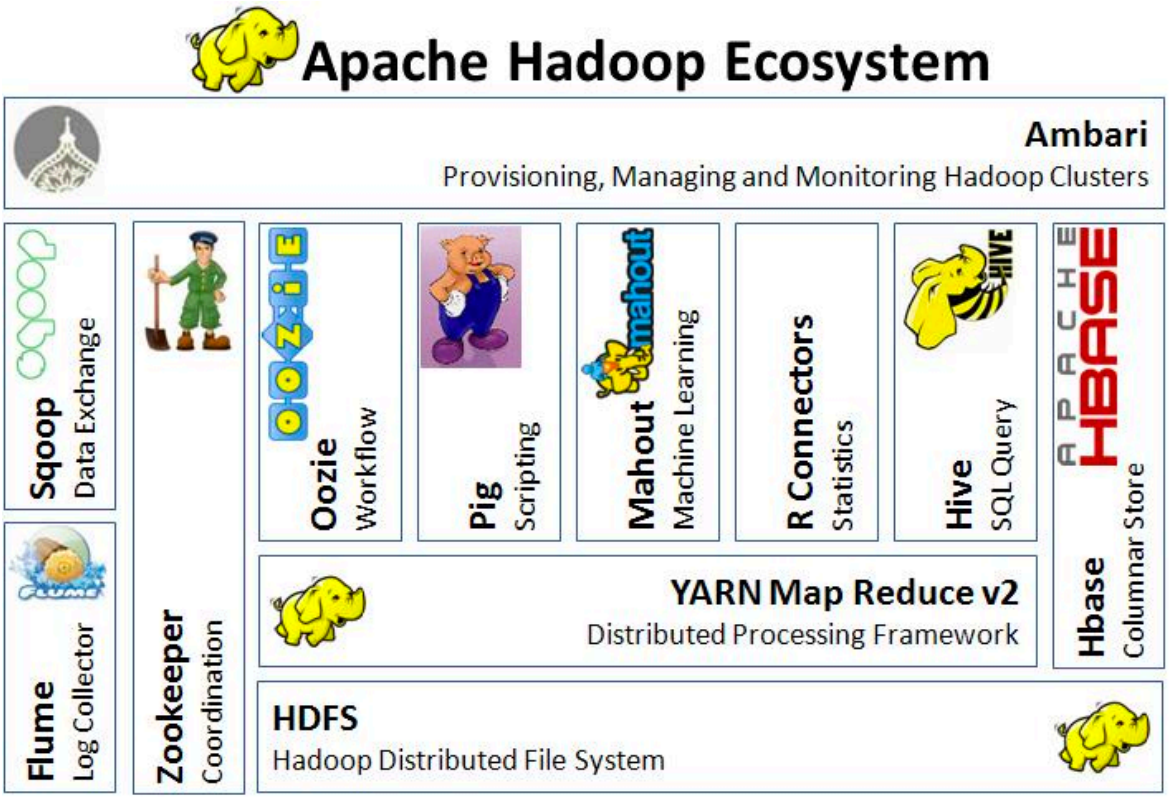
Hadoop следует использовать, если:

- Вычисления должны быть компонуемыми, другими словами, вы должны иметь возможность запустить вычисления на подмножестве данных, а затем слить результаты.
- Вы планируете обрабатывать большой объем неструктурированных данных — больше, чем можно уместить на одной машине (> нескольких терабайт данных).

Hadoop не следует использовать:

- Для некомпонуемых задач — например, для задач рекуррентных.
- Если весь объем данных умещается на одной машине. Существенно сэкономите время и ресурсы.
- Hadoop в целом — система для пакетной обработки и не подходит для анализа в режиме реального времени (здесь на помощь приходит система Storm).

Архитектура HDFS и типичный Hadoop кластер



HDFS подобна другим традиционным файловым системам: файлы хранятся в виде блоков, существует маппинг между блоками и именами файлов, поддерживается древовидная структура, поддерживается модель доступа к файлам основанная на правах и т. п.

Отличия HDFS:

- Предназначена для хранения большого количества огромных (>10GB) файлов. Одним Следствие — большой размер блока по сравнению с другими файловыми системами (>64MB)

- Оптимизирована для поддержки потокового доступа к данным (high-streaming read), соответственно производительность операций произвольного чтения данных начинает хромать.

- Ориентирована на использование большого количество недорогих серверов. В частности, серверы используют JBOB структуру (Just a bunch of disk) вместо RAID — зеркалирование и репликация осуществляются на уровне кластера, а не на уровне отдельной машины.

- Многие традиционные проблемы распределенных систем заложены в дизайн — уже по дефолту все выход отдельных нод из строя является совершенно нормальной и естественной операцией, а не чем-то из ряда вон.

Hadoop-кластер состоит из нод трех типов: NameNode, Secondary NameNode, Datanode.

Namenode — мозг системы. Как правило, одна нода на кластер (больше в случае Namenode Federation, но мы этот случай оставляем за бортом). Хранит в себе все метаданные системы — непосредственно маппинг между файлами и блоками. Если нода 1 то она же и является Single Point of Failure. Эта проблема решена во второй версии Hadoop с помощью Namenode Federation.

Secondary NameNode — 1 нода на кластер. Принято говорить, что «Secondary NameNode» — это одно из самых неудачных названий за всю историю программ. Действительно, Secondary NameNode не является репликой NameNode. Состояние файловой системы хранится непосредственно в файле fsimage и в лог файле edits, содержащим последние изменения файловой системы (похоже на лог транзакций в мире РСУБД). Работа Secondary NameNode заключается в периодическом мерже fsimage и edits — Secondary NameNode поддерживает размер edits в разумных пределах. Secondary NameNode необходима для быстрого ручного восстанавления NameNode в случае выхода NameNode из строя.

В реальном кластере NameNode и Secondary NameNode — отдельные сервера, требовательные к памяти и к жесткому диску. А заявленное “commodity hardware” — уже случай DataNode.

DataNode — Таких нод в кластере очень много. Они хранят непосредственно блоки файлов. Нода регулярно отправляет NameNode свой статус (показывает, что еще жива) и ежечасно — репорт, информацию обо всех хранимых на этой ноде блоках. Это необходимо для поддержания нужного уровня репликации.

Посмотрим, как происходит запись данных в HDFS:

1. Клиент разрезает файл на цепочки блокового размера.

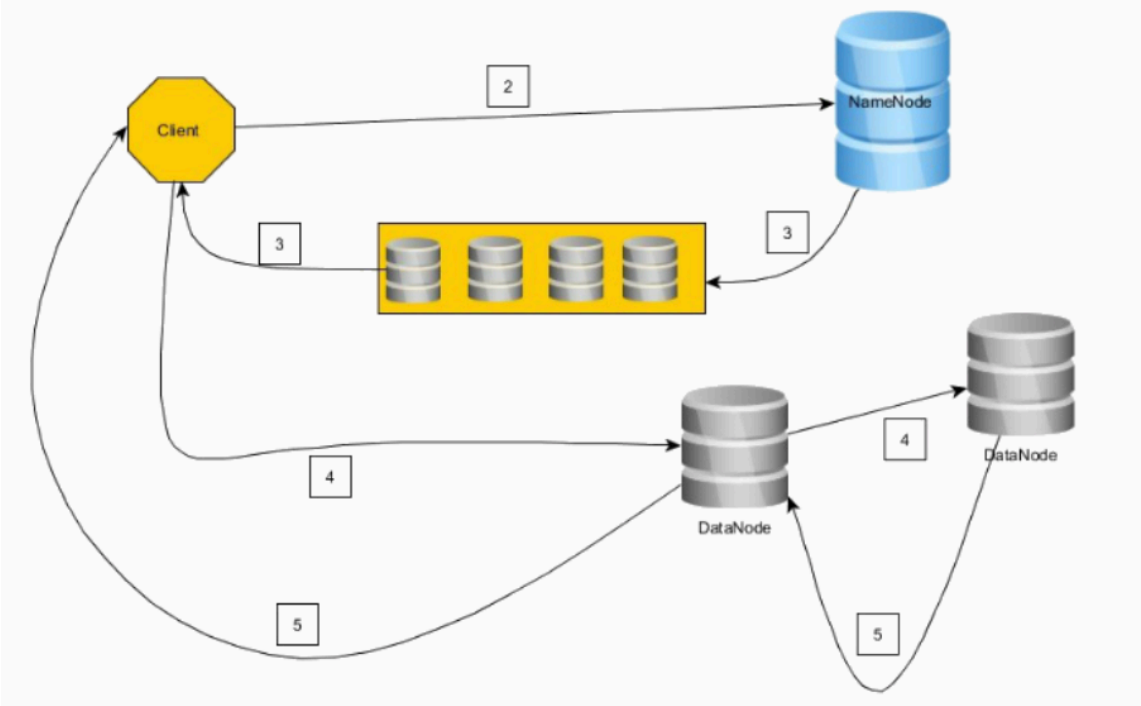
2. Клиент соединяется с NameNode и запрашивает операцию записи, присылая количество блоков и требуемый уроень репликации

3. NameNode отвечает цепочкой из DataNode.

4. Клиент соединяется с первой нодой из цепочки (если не получилось с первой, со второй и т. д. не получилось совсем — откат). Клиент делает запись первого блока на первую ноду, первая нода — на вторую и т. д.

5. По завершении записи в обратном порядке (4 -> 3, 3 -> 2, 2 -> 1, 1 -> клиенту) присылаются сообщения об успешной записи.

6. Как только клиент получит подтверждение успешной записи блока, он оповещает NameNode о записи блока, затем получает цепочку DataNode для записи второго блока и т.д.



Клиент продолжает записывать блоки, если сумеет записать успешно блок хотя бы на одну ноду, т. е. репликация будет работать по хорошо известному принципу «eventual», в дальнейшем NameNode обязуется компенсировать и таки достичь желаемого уровня репликации.

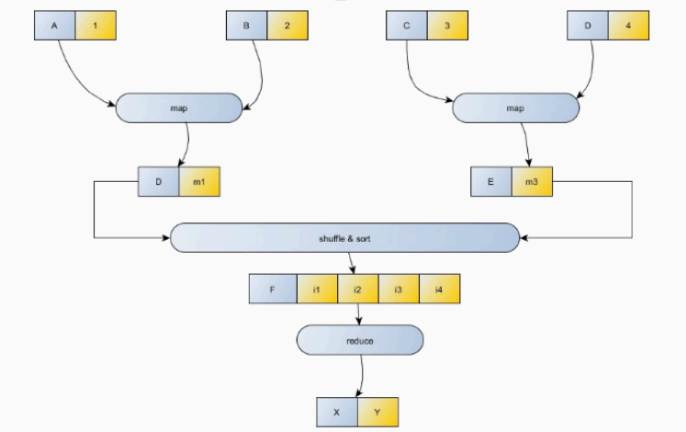
Завершая обзор HDFS и кластера, обратим внимание на еще одну замечательную особенность Hadoop'a — rack awareness. Кластер можно сконфигурировать так, чтобы NameNode имел представление, какие ноды на каких rack'ах находятся, тем самым обеспечив лучшую защиту от сбоев.

MapReduce

Единица работы job — набор map (параллельная обработка данных) и reduce (объединение выводов из map) задач. Map-задачи выполняют mapper'ы, reduce — reducer'ы. Job состоит минимум из одного mapper'a, reducer'ы опциональны. Здесь разобран вопрос разбиения задачи на map'ы и reduce'ы. Если слова «map» и «reduce» вам совсем непонятны, можно посмотреть классическую статью на эту тему.

Модель MapReduce

- Ввод/вывод данных происходит в виде пар (key, value)



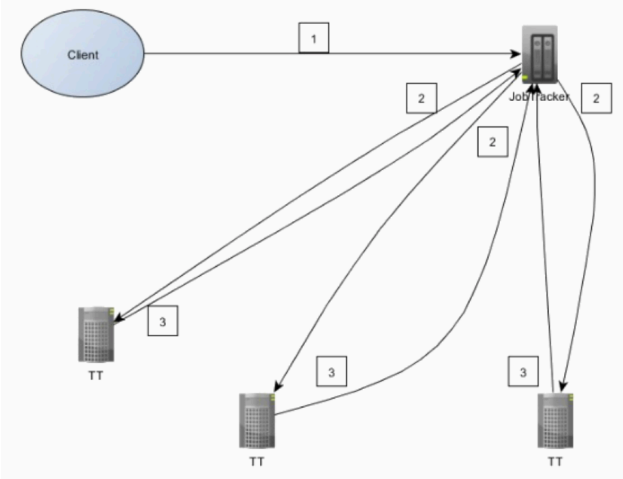
- Используются две функции map: (K1, V1) -> ((K2, V2), (K3,V3), ...) — отображающая пару ключ-значение на некое множество промежуточных пар ключей и значений, а также reduce: (K1, (V2,V3,V4,VN)) -> (K1, V1), отображающая некоторое множество значений, имеющий общий ключ на меньшее множество значений.

- Shuffle and sort нужна для сортировки ввода в reducer по ключу, другими словами, нет смысла отправлять значение (K1, V1) и (K1, V2) на два разных reducer'a. Они должны быть обработаны вместе.

Посмотрим на архитектуру MapReduce

1. Для начала расширим представление о hadoop-кластере, добавив в кластер два новых элемента — JobTracker и TaskTracker. JobTracker непосредственно запросы от клиентов и управляет map/reduce тасками на TaskTracker'ах. JobTracker и NameNode разносится на разные машины, тогда как DataNode и TaskTracker находятся на одной машине.

Взаимодействие клиента и кластера выглядит следующим образом:



1. Клиент отправляет job на JobTracker. Job представляет из себя jar-файл.

2. JobTracker ищет TaskTracker'ы с учетом локальности данных, т.е. предпочитая те, которые уже хранят данные из HDFS. JobTracker назначает map и reduce задачи TaskTracker'ам

3. TaskTracker'ы отправляют отчет о выполнении работы JobTracker'у.

Неудачное выполнение задачи — ожидаемое поведение, провалившиеся таски автоматически перезапускаются на других машинах.

В Map/Reduce 2 (Apache YARN) больше не используется терминология «JobTracker/ TaskTracker». JobTracker разделен на **ResourceManager** — управление ресурсами и **Application Master** — управление приложениями (одним из которых и является непосредственно MapReduce). MapReduce v2 использует новое API