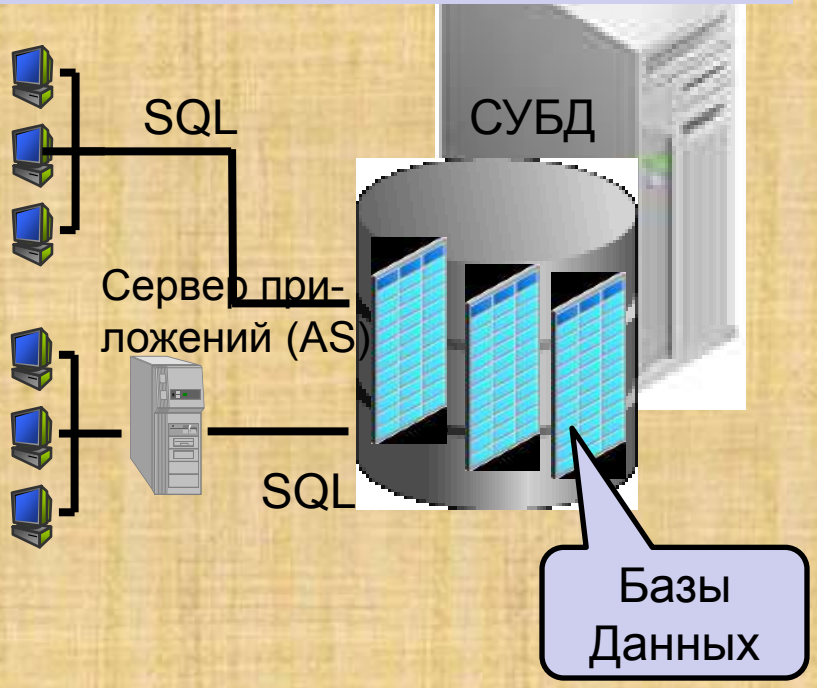
Системы управления базами данных

Батраева Инна Александровна
Кафедра технологий программирования
факультет КНиИТ

Очень общие понятия

База данных (БД) – набор постоянных данных, которые используются прикладными системами для какого-либо предприятия



Система управления базами данных (СУБД) (сервер БД) – программно-аппаратный комплекс - обеспечивает сохранность, целостность данных, доступ пользователей к данным

История развития БД

Система баз данных (database system) – это, по сути, не что иное, как компьютеризированная система хранения записей. Саму же базу данных можно рассматривать как подобие электронной картотеки, т.е. хранилище для некоторого набора занесенных в компьютер файлов данных (где файл – абстрактный набор данных) [К.Дейт].

История развития БД - история развития систем управления данными во внешней памяти.

В первых компьютерах для записи использовались магнитные ленты и барабаны.

Магнитные ленты допускали только последовательный доступ к памяти (и низкую скорость I/O), но обладали достаточно большой емкостью.

Магнитные барабаны давали возможность произвольного доступа к данным, но имели ограниченный объем хранимой информации.

История баз данных фактически началась с появлением магнитных дисков, которые обладают существенно большей емкостью и скоростью доступа.

В **1968** году была введена в эксплуатацию первая промышленная **СУБД** - система IMS фирмы IBM.
В **1975** году появился первый стандарт **СУБД**, разработанный ассоциацией по языкам систем обработки данных - Conference of Data System Language (**CODASYL**).
В **1981** году Э.Ф.Кодд создал реляционную модель данных и применил к ней операции реляционной алгебры.

Этапы развития БД

Файлы и файловые системы

Первые БД были созданы на основе файловых систем. Для каждой прикладной программы предоставлялся свой набор данных, оформленный в виде файла со своей структурой.

Основные операции с файлами в ФС позволяли осуществлять доступ и редактирование данных, но перед разработчиками БД встали проблемы связанные с особенностями организации ФС:

ФС не знает конкретной структуры файла. Структура записи файла известна только программе, которая с ним работает. Каждая программа, работающая с файлом, должна иметь внутри себя структуру данных, соответствующую структуре этого файла. При изменении структуры файла требуется изменять структуру программы, входящие в нее алгоритмы.

Децентрализованное управление доступом к файлу (администрирование).
В файловой системе для каждого файла имеется информация о пользователе – «владельце» файла и определены действия доступные для других пользователей. Это принцип приводит к известным сложностям при построении информационных систем: файл созданный одним пользователем (одной прикладной программой, работающей «от имени» пользователя) оказывался недоступен для другой.

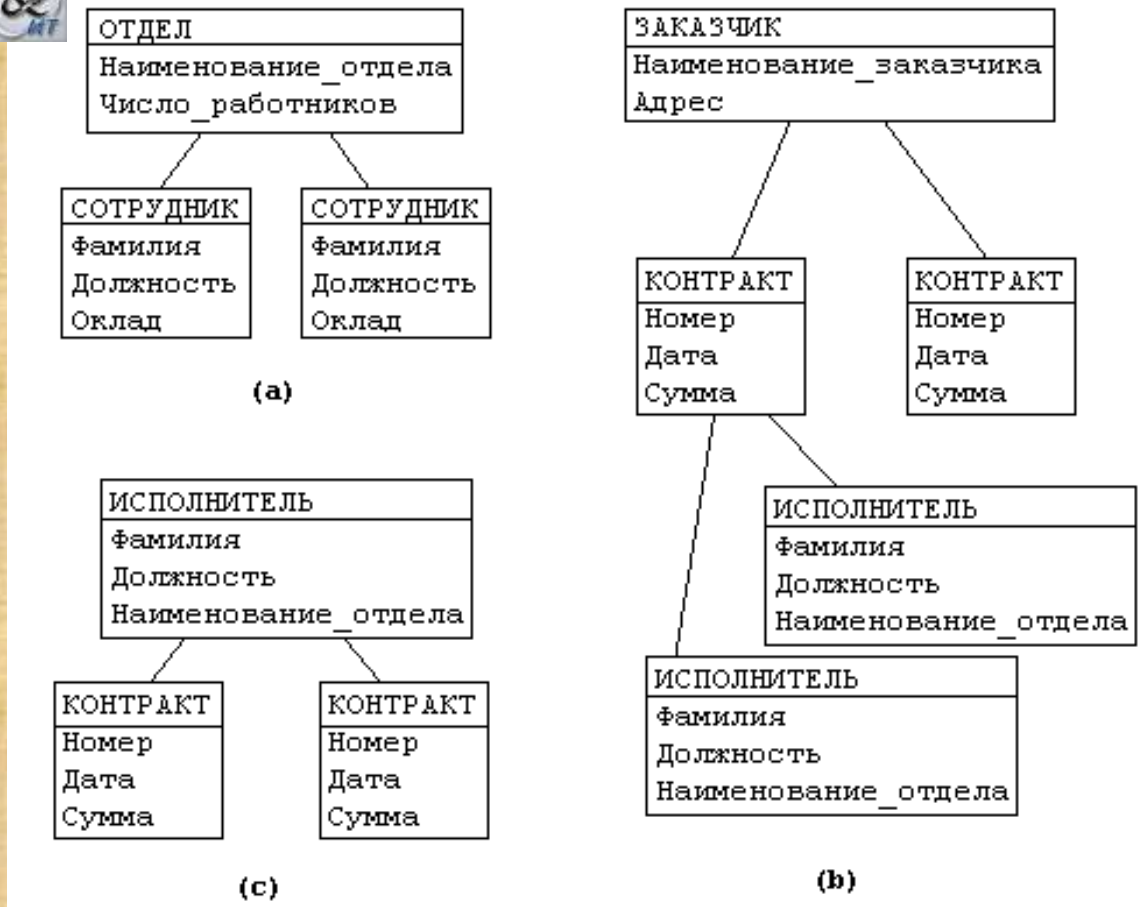
Режим многопользовательского доступа в ФС. Если файл уже используется в режиме изменения, то всем другим пользователям, при попытке открыть файл для изменения о либо недоступен, либо доступен только для чтения. При подобном способе организации одновременная работа нескольких пользователей, связанная с модификацией данных в файле, либо вообще не реализуема, либо сильно замедлена.

Иерархические БД

Рассмотрим следующую модель данных предприятия: предприятие состоит из отделов, в которых работают сотрудники. В каждом отделе может работать несколько сотрудников, но сотрудник не может работать более чем в одном отделе.

Поэтому, для информационной системы управления персоналом необходимо создать структуру, состоящую из родительской записи ОТДЕЛ (НАИМЕНОВАНИЕ_ОТДЕЛА, ЧИСЛО_РАБОТНИКОВ) и дочерней записи СОТРУДНИК (ФАМИЛИЯ, ДОЛЖНОСТЬ, ОКЛАД).

Для автоматизации учета контрактов с заказчиками необходимо создание еще одной иерархической структуры: заказчик - контракты с ним - сотрудники, задействованные в работе над контрактом. Это дерево будет включать записи ЗАКАЗЧИК(НАИМЕНОВАНИЕ_ЗАКАЗЧИКА, АДРЕС), КОНТРАКТ(НОМЕР, ДАТА,СУММА), ИСПОЛНИТЕЛЬ (ФАМИЛИЯ, ДОЛЖНОСТЬ, НАИМЕНОВАНИЕ_ОТДЕЛА).

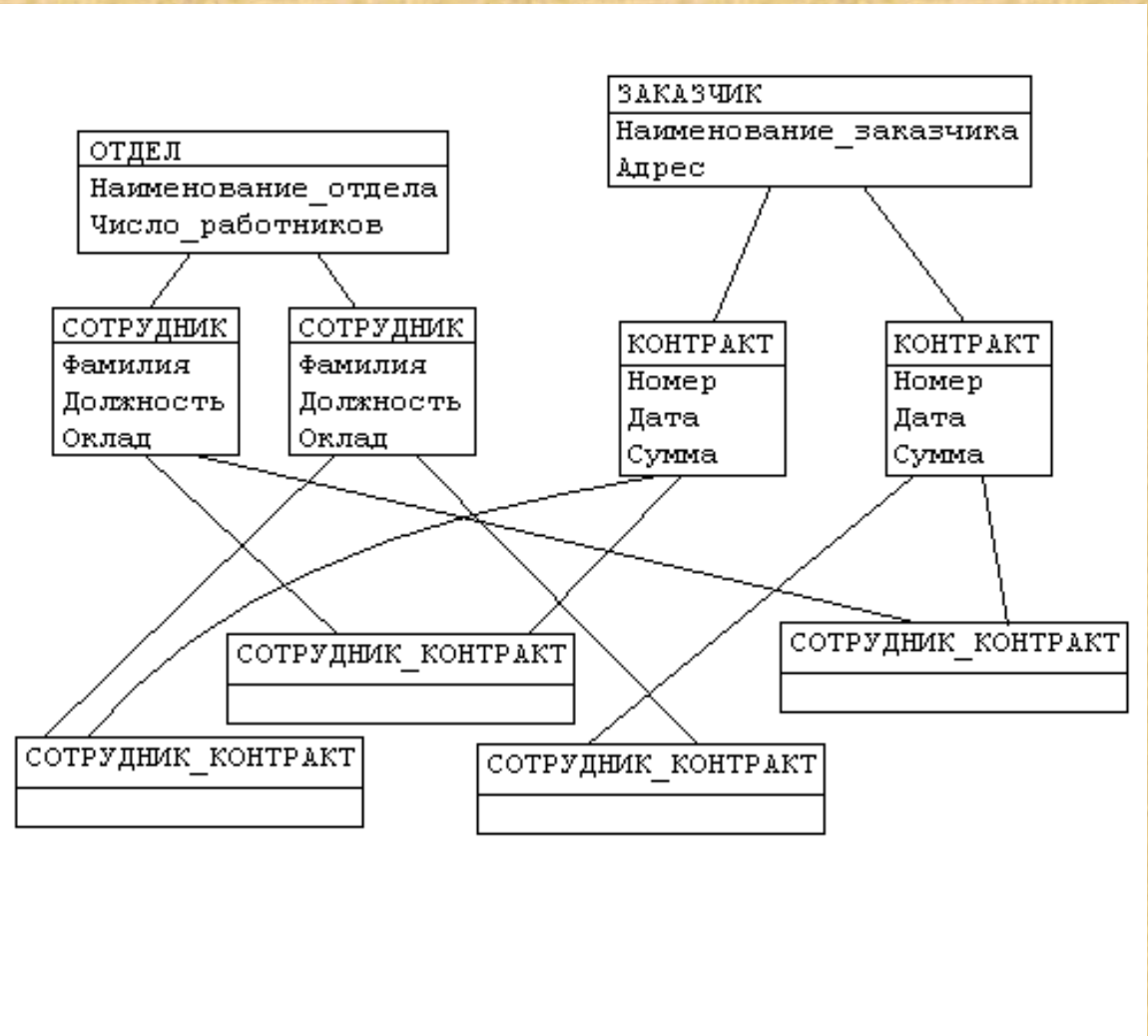


Недостатки иерархических БД:

1. Частично дублируется информация между записями СОТРУДНИК и ИСПОЛНИТЕЛЬ, причем в иерархической модели данных не предусмотрена поддержка соответствия между парными записями.

2. Иерархическая модель реализует отношение между исходной и дочерней записью по схеме **1:N**. Допустим, что исполнитель может принимать участие более чем в одном контракте (связь типа **M:N**). В этом случае в базу данных необходимо ввести еще одно групповое отношение, в котором ИСПОЛНИТЕЛЬ будет являться исходной записью, а КОНТРАКТ - дочерней (рис. **(c)**). Таким образом, опять дублируется информация.

Сетевые БД



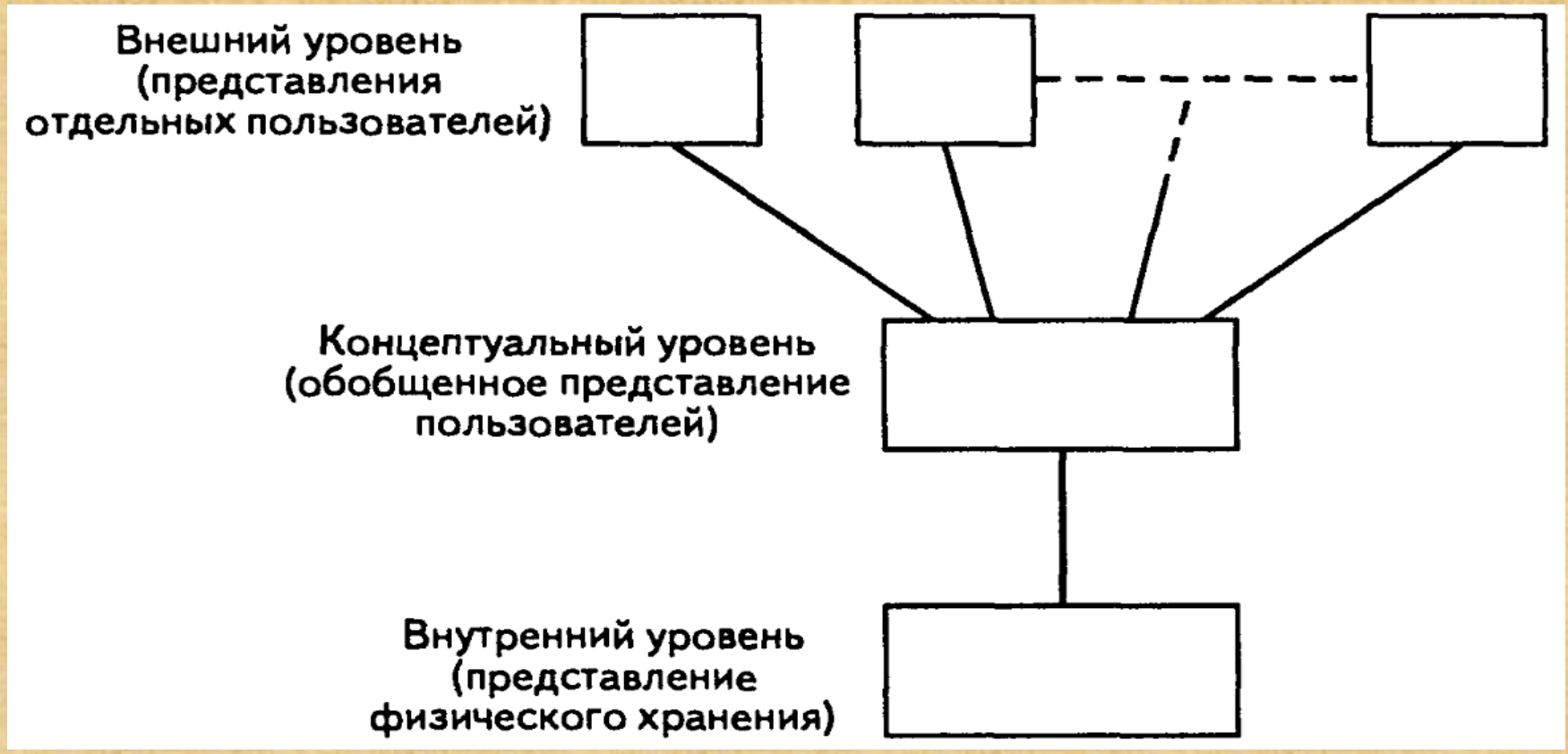
Иерархическая структура преобразовывается в сетевую следующим образом:

1) деревья (a) и (b) заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в обе группы;

2) для отображения типа M:N вводится запись СОТРУДНИК_КОНТРАКТ, которая не имеет полей и служит только для связи записей КОНТРАКТ и СОТРУДНИК.

«Если Вы хотите запутать базу данных – сделайте ее сетевой» - разработчик БД из IBM.

Архитектура Систем Баз Данных

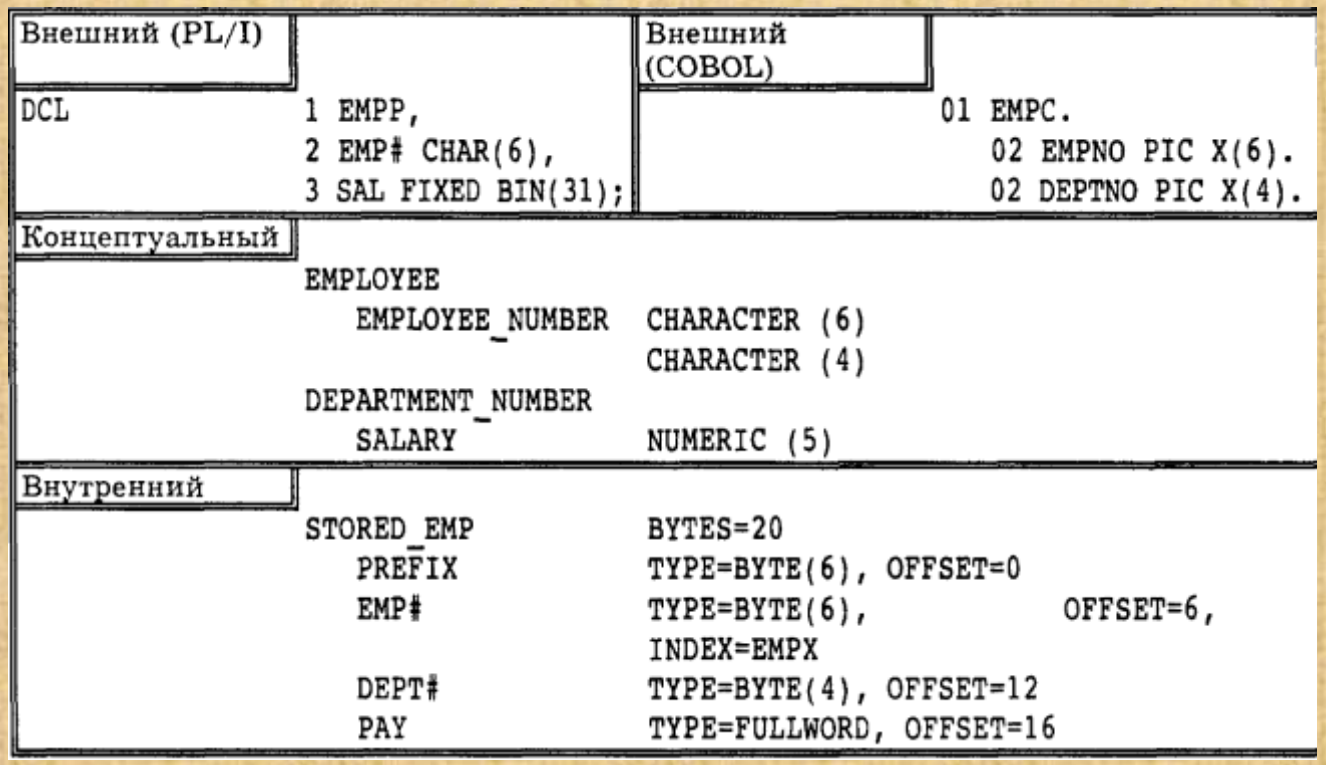


Трехуровневая архитектура ANSI/SPARC,
предложенная американским комитетом по стандартизации ANSI.

Внутренний уровень - уровень наиболее близкий к физическому хранению данных, т.е. связанный с со способами сохранения информации на физических устройствах хранения. Внутренний уровень - собственно данные, расположенные в файлах или в страничных структурах, расположенных на внешних носителях информации.

Концептуальный уровень - уровень, на котором база данных представлена в наиболее общем виде, который объединяет данные, используемые всеми приложениями, работающими с данной базой данных. Фактически концептуальный уровень отражает обобщенную модель предметной области (объектов реального мира), для которой создавалась база данных. Как любая модель, концептуальная модель отражает только существенные, с точки зрения обработки, особенности объектов реального мира.

Внешний уровень - самый верхний уровень, где каждое представление имеет свое "видение" данных. Этот уровень определяет точку зрения на БД отдельных приложений. Каждое приложение видит и обрабатывает только те данные, которые необходимы именно этому приложению. Например, система распределения работ использует сведения о квалификации сотрудника, но ее не интересуют сведения об окладе, домашнем адресе и телефоне сотрудника, и наоборот, именно эти сведения используются в подсистеме отдела кадров.



Трехуровневая архитектура БД позволяет обеспечить логическую (между 1 и 2 ур.) и физическую (между 2 и 3 ур.) независимость при работе с данными.

Логическая независимость предполагает возможность изменения одного приложения без корректировки других приложений, работающих с этой же БД.

Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с данной БД.

Выделение концептуального уровня позволило разработать аппарат централизованного управления БД.

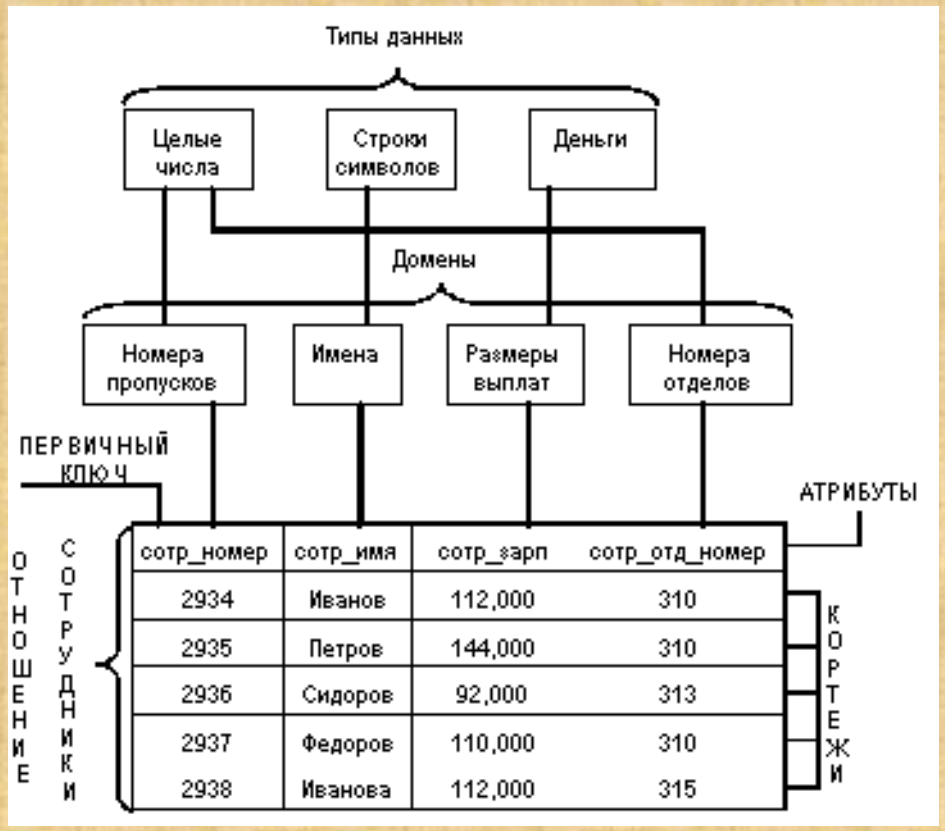
Система управления базами данных представляет собой программное обеспечение, которое управляет доступом к БД [К.Дейт].

Основы реляционной модели данных

1. Базовые понятия реляционных баз данных

Основными понятиями реляционных баз данных являются тип данных, домен, атрибут, кортеж, первичный ключ и отношение.

Покажем смысл этих понятий на примере отношения СОТРУДНИКИ, содержащего информацию о сотрудниках некоторой организации:



сотр_номер	сотр_имя	сотр_зарп	сотр_отд_номер
2934	Иванов	112,000	310
2935	Петров	144,000	310
2936	Сидоров	92,000	313
2937	Федоров	110,000	310
2938	Иванова	112,000	315

1.1. Тип данных

Понятие тип данных в реляционной модели данных полностью адекватно понятию типа данных в языках программирования. Обычно в современных РБД допускается хранение символьных, числовых данных, битовых строк, специализированных числовых данных (таких как "деньги"), а также специальных "темпоральных" данных (дата, время, временной интервал).

1.2. Домен

Понятие домена более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования.

Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа. Например, в число значений домена "Имена« могут входить только те строки, которые могут изображать имя.

Данные считаются сравнимыми только в том случае, когда они относятся к одному домену.

1.3. Кортеж, отношение

Фундаментальным понятием реляционной модели данных является понятие ***отношения***. В определении понятия отношения будем следовать книге К. Дейта [11].

Определение 1. ***Атрибут отношения*** *есть пара вида*

<Имя_атрибута : Имя_домена>.

Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.

Определение 2. ***Отношение*** *R*, определенное на множестве доменов *D1, … Dn* (не обязательно различных), содержит две части: заголовок и тело.

Заголовок отношения содержит фиксированное количество атрибутов отношения:

(<A1:D1>,…, <An:Dn>)

Тело отношения содержит множество кортежей отношения. Каждый ***кортеж отношения*** представляет собой множество пар вида <Имя_атрибута : Значение_атрибута>:

(<A1:Val1>,…, <An:Valn>)

таких что значение *Vali* атрибута *Ai* принадлежит домену *Di*

Отношение обычно записывается в виде:

R(<A1:D1>,…, <An:Dn>) , или короче R*(A1,…,An)*, или просто *R.*

Число атрибутов в отношении называют ***степенью*** (или ***-арностью***) отношения.

Мощность множества кортежей отношения называют ***мощностью*** отношения.

Определение 3. ***Реляционной базой данных*** называется набор отношений.

Определение 4. ***Схемой реляционной базы*** данных называется набор заголовков отношений, входящих в базу данных.

1.4. Свойства отношений

1. В отношении нет одинаковых кортежей. Тело отношения есть *множество* кортежей и, как всякое множество, не может содержать неразличимые элементы. Таблицы в отличие от отношений могут содержать одинаковые строки.

2. Кортежи не упорядочены (сверху вниз). Действительно, несмотря на то, что мы изобразили отношение "Сотрудники" в виде таблицы, нельзя сказать, что сотрудник Иванов "предшествует" сотруднику Петрову. Одно и то же отношение может быть *изображено* разными таблицами, в которых *строки идут в различном порядке*.

3. Атрибуты не упорядочены (слева направо). Т.к. каждый атрибут имеет уникальное имя в пределах отношения, то порядок атрибутов не имеет значения. Одно и то же отношение может быть *изображено* разными таблицами, в которых *столбцы идут в различном порядке*.

4. Все значения атрибутов атомарны. Это четвертое отличие отношений от таблиц - в ячейки таблиц можно поместить что угодно - массивы, структуры, и даже другие таблицы.

1.5. Первая нормальная форма

Труднее всего дать определение вещей, которые всем понятны. Именно такая ситуация с определением отношения в ***Первой Нормальной Форме*** (***1НФ***). Совсем не говорить об этом нельзя, т.к. на основе 1НФ строятся более высокие нормальные формы. Дать определение 1НФ сложно ввиду его тривиальности. Поэтому, дадим просто несколько объяснений.

Объяснение 1. Говорят, что отношение *R* находится в 1НФ, если оно удовлетворяет определению 2.

Это, собственно, тавтология, ведь из определения 2 следует, что других отношений не бывает. Действительно, определение 2 описывает, что является отношением, а что - нет, следовательно, отношений в непервой нормальной форме просто нет.

Объяснение 2. Говорят, что отношение *R* находится в 1НФ, если его атрибуты содержат только скалярные (атомарные) значения.

2. Целостность реляционных данных

Существуют два ограничения, которые должны выполняться в *любой* реляционной базе данных. Это:

Целостность сущностей.

Целостность внешних ключей.

Прежде, чем говорить о целостности сущностей, опишем использование null-значений в реляционных базах данных.

2.1. Null-значения

Достаточно часто встречается ситуация, когда данные неизвестны или не полны. Например, место жительства или дата рождения человека могут быть неизвестны (база данных разыскиваемых преступников). Если вместо неизвестного адреса уместно было бы вводить пустую строку, то что вводить вместо неизвестной даты? Ответ - пустую дату – не удовлетворителен, т.к. простейший запрос "выдать список людей в порядке возрастания дат рождения" даст заведомо неправильных ответ.

Для того чтобы обойти проблему неполных или неизвестных данных, в базах данных могут использоваться типы данных, пополненные так называемым ***null-значением***. Null-значение - это, собственно, не значение, а некий маркер, показывающий, что значение неизвестно.

2.2. Трехзначная логика (3VL)

Т.к. null-значение обозначает на самом деле тот факт, что значение неизвестно, то любые алгебраические операции (сложение, умножение, конкатенация строк и т.д.) должны давать также неизвестное значение, т.е. null. Действительно, если, например, вес детали неизвестен, то неизвестно также, сколько весят 10 таких деталей.

При сравнении выражений, содержащих null-значения, результат также может быть неизвестен, например, значение истинности для выражения есть null, если один или оба аргумента есть null. Таким образом, определение истинности логических выражений базируется на ***трехзначной логике*** (***three-valued logic***, ***3VL***), в которой кроме значений T - ИСТИНА и F - ЛОЖЬ, введено значение U - НЕИЗВЕСТНО. Логическое значение U - это то же самое, что и null-значение.

AND	F	T	U
F	F	F	F
T	F	T	U
U	F	U	U

OR	F	T	U
F	F	T	U
T	T	T	T
U	U	T	U

	NOT
F	T
T	F
U	U

Имеется несколько парадоксальных следствий применения трехзначной логики.

Парадокс 1. Null-значение не равно самому себе. Действительно, выражение null = null дает значение не ИСТИНА, а НЕИЗВЕСТНО. Значит выражение не обязательно ИСТИНА!

Парадокс 2. Неверно также, что null-значение не равно самому себе! Действительно, выражение null <> null также принимает значение не ИСТИНА, а НЕИЗВЕСТНО! Значит также, что и выражение тоже не обязательно ЛОЖЬ!

Парадокс 3. *a or (not a)* не обязательно ИСТИНА. Значит, в трехзначной логике не работает принцип исключенного третьего (любое высказывание либо истинно, либо ложно).

2.3. Потенциальные ключи

По определению, тело отношения есть *множество* кортежей, поэтому отношения не могут содержать одинаковые кортежи. Это значит, что каждый кортеж должен обладать *свойством уникальности*. На самом деле, свойством уникальности в пределах отношения могут обладать отдельные атрибуты кортежей или группы атрибутов. Такие уникальные атрибуты удобно использовать для идентификации кортежей.

Определение 1. Пусть дано отношение *R*. Подмножество атрибутов *K* отношения *R* будем называть ***потенциальным ключом***, если *K* обладает следующими свойствами:

1) *Свойством уникальности* - в отношении *R* не может быть двух различных кортежей, с одинаковым значением *K*.

2) *Свойством неизбыточности* - никакое подмножество в *K* не обладает свойством уникальности.

Любое отношение имеет, по крайней мере, один потенциальный ключ. Действительно, если никакой атрибут или группа атрибутов не являются потенциальным ключом, то, в силу уникальности кортежей, все атрибуты вместе образуют потенциальный ключ.

Потенциальный ключ, состоящий из одного атрибута, называется ***простым***. Потенциальный ключ, состоящий из нескольких атрибутов, называется ***составным***.

Отношение может иметь несколько потенциальных ключей. Традиционно, один из потенциальных ключей объявляется ***первичным***, а остальные - ***альтернативными***. Различия между первичным и альтернативными ключами могут быть важны в конкретной реализации реляционной СУБД, но с точки зрения реляционной модели данных, нет оснований выделять таким образом один из потенциальных ключей.

Замечание. Понятие потенциального ключа является *семантическим* понятием и отражает некоторый смысл (трактовку) понятий из конкретной предметной области.

Таб. номер	Фамилия	Зарплата
1	Иванов	2000
2	Миронов	1000
3	Федоров	2500

Отношение 1

A	B	C
1	Иванов	2000
2	Миронов	1000
3	Федоров	2500

Отношение 2

Что является первичным ключом в отношение 1?

К пониманию того, что в отношении 1 только один потенциальный ключ – «Табельный номер» привело *понимание смысла данных*, содержащихся в отношении.

Что является первичным ключом в отношении 2?

Очевидно, что невозможно судить, не понимая смысла данных, может или не может в отношении 2 появиться кортеж (1, Иванов, 300). И мы не сможем сказать, что же *является* первичным ключом.

Целостность сущностей

Т.к. потенциальные ключи фактически служат идентификаторами объектов предметной области (т.е. предназначены для *различения* объектов), то значения этих идентификаторов не могут содержать неизвестные значения.

Это определяет следующее ***правило целостности сущностей***:

> Атрибуты, входящие в состав некоторого потенциального ключа не могут принимать null-значений.

2.4. Внешние ключи

Различные объекты предметной области, информация о которых хранится в базе данных, всегда взаимосвязаны друг с другом. Например, накладная на поставку товара *содержит* список товаров с количествами и ценами, сотрудник предприятия *имеет* детей, *числится* в подразделении и т.д. Термины "содержит", "имеет", "числится" отражают взаимосвязи между понятиями "накладная" и "список товаров", "сотрудник" и "дети", "сотрудник" и "подразделение". Такие взаимосвязи отражаются в реляционных базах данных при помощи ***внешних ключей***, связывающих несколько отношений.

Пример.

Требуется хранить информацию о наименовании поставщиков фильмов, наименовании поставляемых дисков, причем каждый поставщик может поставлять разные фильмы и каждый фильм может поставляться несколькими поставщиками. Можно предложить хранить данные в следующем отношении:

Номер товара	Товар	Кол-во	Номер пост-ка	Поставщик
1	«Властелин колец»	100	1	ООО «Премьер-Видео»
2	«Герой»	200	2	ООО «Ди Ви Ди Клуб»
2	«Герой»	130	3	ООО «Диск ПРО»
3	«Индиана Джонс»	150	1	ООО «Премьер-Видео»
4	«Назад в будущее»	200	3	ООО «Диск ПРО»

Потенциальный ключ – (Номер товара, Номер поставщика).

Проблемы:

1. Если изменилось наименование поставщика – необходимо внести изменение во все строки таблицы
2. Если поставщик прекратил поставки – удаление информации о поставках приведет к удалению информации о поставщике.
3. Если товар временно никем не поставляется – будет потеряна информация о товаре.

Причина: смешана разнородная информация

Связь данных в соответствии с семантикой предметной области в отношении определяется фразами: "Поставщики *выполняют* Поставки", «Товар *поставляются через* Поставки". Эти две взаимосвязи косвенно определяют новую взаимосвязь между "Поставщиками" и «Товаром": «Товар *поставляется* Поставщиками".

Типы взаимосвязей:

"*Один* Поставщик может выполнять *несколько* Поставок",

"*Один Товар* может поставляться *несколькими* Поставками".

Это взаимосвязь типа "***один-ко-многим***« (1:N)

"*Несколько Товаров* могут поставляться *несколькими* Поставщиками". Это взаимосвязь типа "***много-ко-многим***“ (M:N).

Взаимосвязи типа "много-ко-многим" реализуются в РБД использованием нескольких взаимосвязей типа "один-ко-многим".

Отношение, входящее в связь со стороны "один" ("Поставщики"), называют ***родительским отношением***.

Отношение, входящее в связь со стороны "много" ("Поставки"), называется ***дочернем отношением***.

Номер поставщика	Поставщик
1	ООО «Премьер-Видео»
2	ООО «Ди Ви Ди Клуб»
3	ООО «Диск ПРО»

Отношение «Поставщики»

Номер товара	Товар
1	«Властелин колец»
2	«Герой»
3	«Индиана Джонс»
4	«Назад в будущее»

Отношение «Товар»

Номер поставщика (FK)	Номер товара (FK)	Кол-во
1	1	100
2	2	200
3	2	130
1	3	150
3	4	200

Отношение «Поставки»

Определение 2.

Пусть дано отношение *R*. Подмножество атрибутов *FK* отношения *R* будем называть ***внешним ключом***, если:

1) Существует отношение *S* (*R* и *S* не обязательно различны) с потенциальным ключом *K*.

2) Каждое значение *FK* в отношении *R* всегда совпадает со значением *K* для некоторого кортежа из *S*, либо является null-значением.

Отношение S называется ***родительским отношением***, отношение R называется ***дочерним отношением***.

Замечание. Внешний ключ, как правило, *не обладает свойством уникальности*. Это, собственно, и дает тип отношения 1:N.

Замечание. Хотя каждое значение внешнего ключа обязано совпадать со значениями потенциального ключа в некотором кортеже родительского отношения, то обратное, вообще говоря, неверно. Например, могут существовать поставщики, не поставляющие никаких деталей.

Замечание. Для внешнего ключа не требуется, чтобы он был компонентом некоторого потенциального ключа.

Целостность внешних ключей

Т.к. внешние ключи фактически служат ссылками на кортежи в другом (или в том же самом) отношении, то эти ссылки не должны указывать на несуществующие объекты.

Это определяет следующее ***правило целостности внешних ключей***:

> Внешние ключи не должны быть несогласованными, т.е. для каждого значения внешнего ключа должно существовать соответствующее значение первичного ключа в родительском отношении.

Явная формулировка правил целостности помогает четко понять, какие опасности несет в себе пренебрежение этими правилами.

Операции, могущие нарушить ссылочную целостность

Ссылочная целостность может нарушиться в результате операций, изменяющих состояние базы данных. Таких операций три - вставка, обновление и удаление кортежей в отношениях.

Для родительского отношения

Вставка кортежа в родительском отношении. При вставке кортежа в родительское отношение возникает новое значение потенциального ключа. Т.к. допустимо существование кортежей в родительском отношении, на которые нет ссылок из дочернего отношения, то вставка кортежей в родительское отношение *не нарушает ссылочной целостности*.

Обновление кортежа в родительском отношении. При обновлении кортежа в родительском отношении может измениться значение потенциального ключа. Если есть кортежи в дочернем отношении, ссылающиеся на обновляемый кортеж, то значения их внешних ключей станут некорректными. Обновление кортежа в родительском отношении *может привести к нарушению ссылочной целостности*, если это обновление затрагивает значение потенциального ключа.

Удаление кортежа в родительском отношении. При удалении кортежа в родительском отношении удаляется значение потенциального ключа. Если есть кортежи в дочернем отношении, ссылающиеся на удаляемый кортеж, то значения их внешних ключей станут некорректными. Удаление кортежей в родительском отношении *может привести к нарушению ссылочной целостности*.

Для дочернего отношения

Вставка кортежа в дочернее отношение. Нельзя вставить кортеж в дочернее отношение, если вставляемое значение внешнего ключа некорректно. Вставка кортежа в дочернее отношение *привести к нарушению ссылочной целостности*.

Обновление кортежа в дочернем отношении. При обновлении кортежа в дочернем отношении можно попытаться некорректно изменить значение внешнего ключа. Обновление кортежа в дочернем отношении *может привести к нарушению ссылочной целостности*.

Удаление кортежа в дочернем отношении. При удалении кортежа в дочернем отношении *ссылочная целостность не нарушается*.

Таким образом, ссылочная целостность в принципе может быть нарушена при выполнении одной из четырех операций:

- Обновление кортежа в родительском отношении.
- Удаление кортежа в родительском отношении.
- Вставка кортежа в дочернее отношение.
- Обновление кортежа в дочернем отношении.

Стратегии поддержания ссылочной целостности

RESTRICT (ОГРАНИЧИТЬ)- не разрешать выполнение операции, приводящей к нарушению ссылочной целостности. Это самая простая стратегия, требующая только проверки, имеются ли кортежи в дочернем отношении, связанные с некоторым кортежем в родительском отношении.

CASCADE (КАСКАДИРОВАТЬ)- разрешить выполнение требуемой операции, но внести при этом необходимые поправки в других отношениях так, чтобы не допустить нарушения ссылочной целостности и сохранить все имеющиеся связи.

Изменение начинается в родительском отношении и каскадно выполняется в дочернем отношении. Необходимо учитывать, что дочернее отношение само может быть родительским для некоторого третьего отношения. При этом может дополнительно потребоваться выполнение какой-либо стратегии и для этой связи и т.д. Если при этом какая-либо из каскадных операций (любого уровня) не может быть выполнена, то необходимо отказаться от первоначальной операции и вернуть базу данных в исходное состояние.

Стратегии поддержания ссылочной целостности реализуются в основном двумя способами:

- на уровне определения данных

```
alter table POSTAVKI add foreign key (fkidTovar) references
TOVAR(idTovar) on delete cascade
```

- с использованием триггеров

```
create trigger trTovar for Tovar before update as
declare variable n;
begin
select count(*) from Postavki where old.idTovar = Postavki.fkidTovar into n;
if n>0 then delete from Postavki where old.idTovar=Postavki.fkidTovar ;
end
```

3. Реляционная алгебра

В описании реляционной модели утверждается, что доступ к реляционным данным осуществляется при помощи реляционной алгебры (РА) или эквивалентного ему реляционного исчисления (РИ). В реализациях реляционных СУБД сейчас не используется в чистом виде ни РА, ни РИ.

Фактическим стандартом доступа к реляционным данным стал язык SQL (Structured Query Language). Язык SQL представляет собой смесь операторов реляционной алгебры и выражений реляционного исчисления, использующий синтаксис, близкий к фразам английского языка и расширенный дополнительными возможностями, отсутствующими в реляционной алгебре и реляционном исчислении. Вообще, язык доступа к данным называется ***реляционно полным***, если он по выразительной силе не уступает реляционной алгебре (или, что то же самое, реляционному исчислению), т.е. любой оператор реляционной алгебры может быть выражен средствами этого языка. Именно таким и является язык SQL.

РА представляет собой набор операторов, использующих отношения в качестве аргументов, и возвращающие отношения в качестве результата:

$$R=f(R_1,\ldots,Rn)$$

РА является замкнутой: $R=f(f_1(R_{11},R_{12},\ldots),\ldots, f_n(Rn_1,Rn_2,\ldots))$

Традиционно, вслед за Коддом, определяют восемь реляционных операторов, объединенных в две группы.

Теоретико-множественные операторы:

- Объединение
- Пересечение
- Вычитание
- Декартово произведение

Специальные реляционные операторы:

- Выборка
- Проекция
- Соединение
- Деление

Объединением *(A UNION B)* двух совместимых по типу отношений *A* и *B* называется отношение с тем же заголовком, что и у отношений *A* и *B*, и телом, состоящим из кортежей, принадлежащих или *A*, или *B*, или обоим отношениям.

Пересечением *(A INTERSECT B)* двух совместимых по типу отношений *A* и *B* называется отношение с тем же заголовком, что и у отношений *A* и *B*, и телом, состоящим из кортежей, принадлежащих отношениям *A* и *B* одновременно.

Вычитанием *(A MINUS B)* двух совместимых по типу отношений *A* и *B* называется отношение с тем же заголовком, что и у отношений *A* и *B*, и телом, состоящим из кортежей, принадлежащих отношению *A* и не принадлежащих отношению *B*.

Декартовым произведением *(A TIMES B)* двух отношений *A(A1,…,An)* и *B(B1,…,Bm)* называется отношение, заголовок которого является ***сцеплением заголовков*** отношений *A* и *B*:

$$(A1,\ldots,An, B1,\ldots,Bm)$$

а тело состоит из кортежей, являющихся ***сцеплением кортежей*** отношений *A* и *B*:

$$(a1,\ldots,an, b1,\ldots,bm)$$

таких, что $(a1,\ldots,an) \in A$, $(b1,\ldots,bm) \in B$.

, .

Выборкой *(A where c)* на отношении *A* с условием *c* называется отношение с тем же заголовком, что и у отношения *A*, и телом, состоящем из кортежей, значения атрибутов которых при подстановке в условие *c* дают значение ИСТИНА. Условие *c* представляет собой логическое выражение, в которое могут входить атрибуты отношения и (или) скалярные выражения.

Проекцией *(A[Ai,…,Ak])* отношения *A* по атрибутам *Ai,…,Ak*, где каждый из атрибутов принадлежит отношению *A*, называется отношение с заголовком *(Ai,…,Ak)* и телом, содержащим множество кортежей вида *(ai,…,ak)*, таких, для которых в отношении *A* найдутся кортежи со значением атрибута *Ai* равным *ai*, …, значением атрибута *Ak* равным *ak*.

Соединением отношений *A* и *B* по условию с называется отношение *(A TIMES B) where c.*

Наиболее важной разновидностью соединения является ***естественное соединение.***

Пусть даны отношения *A(A1,..An,X1,…Xk)* и *B(X1,…Xk,B1,..Bn)*, имеющие одинаковые атрибуты *X1,…Xk* (т.е. атрибуты с одинаковыми именами и определенные на одинаковых доменах). Тогда ***естественным соединением*** *(A JOIN B)* отношений *A* и *B* называется отношение с заголовком *(A1,..An,X1,…Xk,B1,..Bn)*, и телом, содержащим множество кортежей *(a1,..an,x1,…xk,b1,..bn)*, таких, что *(a1,..an,x1,…xk)* $\in$ *A* и *(x1,…xk,b1,..bn)* $\in$ *B.*

Поставщики (Номер поставщика, Поставщик) JOIN
Поставки (Номер поставщика, Номер товара, Кол-во) JOIN
Товар(Номер товара, Товар)

(Номер поставщика, Поставщик, Номер товара, Товар, Кол-во)

Пусть даны отношения *A(X1,…Xn, Y1,…Yk)* и *B(Y1,…,Yk)*, причем атрибуты *(Y1,…,Yk)* - общие для *A* и *B*. ***Делением отношений*** *(A DIVIDEBY B)* на называется отношение с заголовком *(X1,…Xn)* и телом, содержащим множество кортежей *(x1,…xn)*, таких, что для *всех* кортежей *(y1,..yk)* $\in$ *B* в отношении *A* найдется кортеж *(x1,…xn, y1,…yk)*. Отношение *A* выступает в роли ***делимого***, отношение *B* выступает в роли ***делителя***. Деление отношений аналогично делению чисел с остатком.

X	Y
1	A
2	B
3	C
1	B
2	A
1	C

DIVIDE BY

Y
A
B
C

=

X
1

4. Нормальные формы отношений

4.1. Этапы разработки БД

При разработке базы данных обычно выделяется несколько уровней моделирования, при помощи которых происходит переход от предметной области к конкретной реализации базы данных средствами конкретной СУБД. Можно выделить следующие уровни:

- Сама предметная область
- Модель предметной области
- Логическая модель данных
- Физическая модель данных
- Собственно база данных и приложения

Предметная область - это часть реального мира, данные о которой мы хотим отразить в базе данных. Например, в качестве предметной области можно выбрать бухгалтерию какого-либо предприятия, отдел кадров, банк, магазин и т.д. Предметная область бесконечна и содержит как существенно важные понятия и данные, так и малозначащие или вообще не значащие данные.

Модель предметной области. Модель предметной области - это наши знания о предметной области. Знания могут быть как в виде неформальных знаний в мозгу эксперта, так и выражены формально при помощи каких-либо средств (текстовые описания предметной области, наборы должностных инструкций, и т.п.)

Описания предметной области обычно выполняются при помощи методик описания предметной области. Из наиболее известных можно назвать методику структурного анализа SADT и основанную на нем IDEF0, диаграммы потоков данных Гейна-Сарсона, методику объектно-ориентированного анализа UML, и др.

Модель предметной области описывает скорее процессы, происходящие в предметной области и данные, используемые этими процессами. От того, насколько правильно смоделирована предметная область, зависит успех дальнейшей разработки приложений.

Логическая модель данных. На следующем, более низком уровне находится логическая модель данных предметной области. Логическая модель описывает понятия предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью.

Примеры понятий – «актер», «фильм», «режиссер", "зарплата". Примеры взаимосвязей между понятиями - «актер снимается в нескольких фильмах", «режиссер снимает фильм», «в одном фильме снимается несколько актеров». Примеры ограничений - "возраст режиссера не менее 18 лет".

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но *без привязки к конкретной СУБД*.

Решения, принятые на предыдущем уровне, при разработке модели предметной области, определяют некоторые границы, в пределах которых можно развивать логическую модель данных, в пределах же этих границ можно принимать различные решения.

Физическая модель данных. На еще более низком уровне находится физическая модель данных. Физическая модель данных описывает данные средствами конкретной СУБД. Будем считать, что физическая модель данных реализована средствами именно *реляционной* СУБД.

Отношения, разработанные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены преображаются в типы данных, принятые в конкретной СУБД.

Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например, при помощи индексов, декларативных ограничений целостности, триггеров, хранимых процедур. При этом решения, принятые на уровне логического моделирования определяют некоторые границы, в пределах которых можно развивать физическую модель данных и принимать различные решения. Например, отношения, содержащиеся в логической модели данных, должны быть преобразованы в таблицы, но для каждой таблицы можно дополнительно объявить различные индексы, повышающие скорость обращения к данным.

Собственно база данных и приложения. И, наконец, как результат предыдущих этапов появляется собственно сама база данных. База данных реализована на конкретной программно-аппаратной основе, и выбор этой основы позволяет существенно повысить скорость работы с базой данных. Например, можно выбирать различные типы компьютеров, менять количество процессоров, объем оперативной памяти, дисковые подсистемы и т.п. Очень большое значение имеет также настройка СУБД в пределах выбранной программно-аппаратной платформы.

Но опять решения, принятые на предыдущем уровне - уровне физического проектирования, определяют границы, в пределах которых можно принимать решения по выбору программно-аппаратной платформы и настройки СУБД.

Таким образом ясно, что решения, принятые на каждом этапе моделирования и разработки базы данных, будут сказываться на дальнейших этапах. Поэтому особую роль играет принятие правильных решений *на ранних этапах моделирования*.

Опишем некоторые принципы построения *хороших логических моделей данных*. Хороших в том смысле, что решения, принятые в процессе логического проектирования приводили бы к хорошим физическим моделям и в конечном итоге к хорошей работе базы данных.

Основной пример

Модель предметной области (мегакинокомпании Голливуд)была описана заказчиком следующим образом:

- Режиссеры кинокомпании снимают фильмы, в которых заняты разные актеры.
- Каждый режиссер одновременно может снимать только один фильм или не снимать ни одного.
- Каждый актер может сниматься в одном или нескольких фильмах одновременно, или временно не участвовать ни в одном.
- В каждом фильме может быть занято несколько актеров, но режиссер может быть только один.

Все сведения были даны в виде отношения с заголовком:
(Н_Ф,Фильм,Н_Р, Режиссер, Сайт_Реж,Н_А,Актер)

Н_Ф	Фильм	Н_Р	Режиссер	Сайт_Реж	Н_А	Актер
1	Леон	1	Люк Бессон	www.lic-besson.com	3	Жан Рено
1	Леон	1	Люк Бессон	www.lic-besson.com	5	Гэри Олдмэн
2	Назад в будущее	2	Роберт Земекис	www.imdb.com/name/nm0000709/	1	Майкл Дж. Фокс
2	Назад в будущее	2	Роберт Земекис	www.imdb.com/name/nm0000709/	2	Кристофер Ллойд
3	Назад в будущее-2	2	Роберт Земекис	www.imdb.com/name/nm0000709/	1	Майкл Дж. Фокс
3	Назад в будущее-2	2	Роберт Земекис	www.imdb.com/name/nm0000709/	2	Кристофер Ллойд
5	Пролетая над гнездом кукушки	3	Милош Форман	www.imdb.com/name/nm0001232	2	Кристофер Ллойд
6	Индиана Джонс и храм судьбы	4	Стивен Спилберг	www.imdb.com/name/nm0000229/	4	Харрисон Форд

В качестве потенциального ключа отношения необходимо взять пару атрибутов {***Н_Ф***, ***Н_А***}, так как каждый актер может сниматься одновременно в нескольких фильмах.

Данные в отношении **ФИЛЬМЫ_РЕЖИССЕРЫ_АКТЕРЫ** хранятся с *большой избыточностью*.

Повторяются фамилии режиссеров, актеров, информация об агентах, названия фильмов. Одновременно в отношении харниться информация о независимых данных (объектах) – актеры, фильмы, режиссеры и т.д. Пока никаких действий с отношением не производится, это допустимо. Но как только состояние предметной области изменяется, то, при попытках соответствующим образом изменить состояние базы данных, возникает большое количество проблем:

- Аномалии вставки (INSERT)
- Аномалии обновления (UPDATE)
- Аномалии удаления (DELETE)

Отношение **ФИЛЬМЫ_РЕЖИССЕРЫ_АКТЕРЫ** находится в 1НФ, но при этом логическая модель данных не адекватна модели предметной области. Таким образом, первой нормальной формы *недостаточно* для правильного моделирования данных.

Для правильного проектирования модели данных применяется метод нормализации отношений. Нормализация основана на понятии функциональной зависимости атрибутов отношения.

Определение 1. Пусть R - отношение. Множество атрибутов Y ***функционально зависимо*** от множества атрибутов X (X ***функционально определяет Y***) т. и т.т., к. *для любого состояния* отношения R для любых кортежей r1,r2 $\in R$ из того, что r1.X=r2.X следует что r1.Y=r2.Y (т.е. во всех кортежах, имеющих одинаковые значения атрибутов X, значения атрибутов Y также совпадают *в любом состоянии* отношения). Символически функциональная зависимость (ФЗ) записывается X->Y. Множество атрибутов X называется ***детерминантом функциональной зависимости***, а множество атрибутов Y называется ***зависимой частью***.

Если атрибуты X составляют потенциальный ключ отношения R, то *любой* атрибут отношения R функционально зависит от X.

В отношении **ФИЛЬМЫ_РЕЖИССЕРЫ_АКТЕРЫ** можно привести следующие примеры функциональных зависимостей (на самом деле их больше):

Зависимость атрибутов от первичного ключа:

{*Н_Ф*, *Н_А*} -> Актер

{*Н_Ф*, *Н_А*} -> Н_Р

{*Н_Ф*, *Н_А*} -> Фильм,Тел_Реж

Зависимость атрибутов, связанных с актером от его номера:

***Н_А* -> Актер**

Зависимость названия фильма от его номера:

***Н_Ф* -> Фильм**

Приведенные зависимости не были выведены из внешнего вида отношения, они являются результатом исследования взаимосвязей между объектами предметной области. Таким образом, ФЗ является семантическим понятием.

Определение 2. Отношение R находится во ***второй нормальной форме*** (***2НФ***) т. и т.т., к. отношение R находится в 1НФ и *нет неключевых атрибутов, зависящих от части сложного ключа*. (***Неключевой атрибут*** - это атрибут, не входящий в состав никакого потенциального ключа).

Если потенциальный ключ отношения является простым, то отношение автоматически находится в 2НФ.

Отношение **ФИЛЬМЫ_РЕЖИССЕРЫ_АКТЕРЫ** не находится в 2НФ, т.к. есть атрибуты, зависящие от части сложного ключа:

- ***Н_А -> Актер***
- ***Н_Ф -> Фильм***

Для того, чтобы устранить зависимость атрибутов от части сложного ключа, произведем ***декомпозицию*** отношения на несколько отношений. При этом *те атрибуты, которые зависят от части сложного ключа,* выносятся в отдельное отношение.

Отношение **ФИЛЬМЫ_РЕЖИССЕРЫ_АКТЕРЫ** разделим на три отношения – **ФИЛЬМЫ_РЕЖИССЕРЫ, АКТЕРЫ, СЪЕМКИ.**

Н_Ф	Фильм	Н_Р	Режиссер	Сайт_Реж
1	Леон	1	Люк Бессон	www.lic-besson.com
2	Назад в будущее	2	Роберт Земекис	www.imdb.com/name/nm0000709/
3	Назад в будущее-2	2	Роберт Земекис	www.imdb.com/name/nm0000709/
5	Пролетая над гнездом кукушки	3	Милош Форман	www.imdb.com/name/nm0001232
6	Индиана Джонс и храм судьбы	4	Стивен Спилберг	www.imdb.com/name/nm0000229/

ФИЛЬМЫ_РЕЖИССЕРЫ

Н_А	Актер
1	Майкл Дж.Фокс
2	Кристофер Ллойд
3	Жан Рено
4	Харрисон Форд
5	Гэри Олдмен

АКТЕРЫ

Н_Ф	Н_А
1	3
1	5
2	1
2	2
3	1
3	2
5	2
6	4

ЗАНЯТОСТЬ

Анализ декомпозированных отношений

Отношения, полученные в результате декомпозиции, находятся в 2НФ. Отношения **Фильмы_Режиссеры**, **Актеры** имеют простые ключи, следовательно автоматически находятся в 2НФ. Отношение **ЗАНЯТОСТЬ** имеет только два атрибута, которые входят в составной ключ, и поэтому тоже находится в 2НФ.

Часть аномалий осталась. Например, в отношение **Фильмы_Режиссеры** нельзя вставить кортеж (7, ‘Такси’,’Люк Бессон’, ‘www.imdb.com’), так как при этом получиться, что у режиссера Люка Бессона два различных сайта.

Если же сайт действительно изменился, то это изменение необходимо внести во все кортежи, где упоминается Бессон.

Причина аномалии - хранение в одном отношении разнородной информации.

Определение 3. Атрибуты называются ***взаимно независимыми***, если ни один из них не является функционально зависимым от другого.

Определение 4. Отношение находится в ***третьей нормальной форме*** (***3НФ***) тогда и только тогда, когда отношение находится в 2НФ и *все неключевые атрибуты взаимно независимы*.

Отношение **ФИЛЬМЫ_РЕЖИССЕРЫ** не находится в 3НФ, т.к. имеется функциональная зависимость неключевых атрибутов:

Н_Р -> Режиссер, Сайт_Реж

Для того, чтобы устранить зависимость неключевых атрибутов, нужно произвести декомпозицию отношения на несколько отношений. При этом *те неключевые атрибуты, которые являются зависимыми,* выносятся в отдельное отношение.

Отношение **ФИЛЬМЫ_РЕЖИССЕРЫ** разделим на два отношения – **ФИЛЬМЫ(Н_Ф, Фильм, Н_Р), РЕЖИССЕРЫ(Н_Р, Режиссер, Сайт_Реж)**.

Н_Ф	Фильм	Н_Р
1	Леон	1
2	Назад в будущее	2
3	Назад в будущее-2	2
5	Пролетая над гнездом кукушки	3
6	Индиана Джонс и храм судьбы	4

Н_Ф	Н_А
1	3
1	5
2	1
2	2
3	1
3	2
5	2
6	4

Н_Р	Режиссер	Сайт_Реж
1	Люк Бессон	www.lic-besson.com
2	Роберт Земекис	www.imdb.com/name/nm0000709/
2	Роберт Земекис	www.imdb.com/name/nm0000709/
3	Милош Форман	www.imdb.com/name/nm0001232
4	Стивен Спилберг	www.imdb.com/name/nm0000229/

Н_А	Актер
1	Майкл Дж.Фокс
2	Кристофер Ллойд
3	Жан Рено
4	Харрисон Форд
5	Гэри Олдмен

```
create table ACT (ACT_ID INTEGER not null primary key ,
        ACTOR VARCHAR(30) character set win1251 not null);

create table FILMS (FILM_ID INTEGER not null primary key ,
        REG_ID INTEGER not null,
        NAZV VARCHAR(20) character set win1251 not null);

create table REG (REG_ID INTEGER not null primary key ,
        REG VARCHAR(30) character set win1251 not null,
        SITE_REG VARCHAR(20) character set win1251 not null);

create table ZAN (FILM_ID INTEGER not null,
        ACT_ID INTEGER not null,
        primary key (FILM_ID, ACT_ID),
        foreign key (FILM_ID) references FILMS(FILM_ID),
        foreign key (ACT_ID) references ACT(ACT_ID));
```

Алгоритм нормализации (приведение к 3НФ)

Шаг 1 (Приведение к 1НФ). На первом шаге задается одно или несколько отношений, отображающих понятия предметной области. По модели предметной области выписываются обнаруженные функциональные зависимости. Все отношения автоматически находятся в 1НФ.

Шаг 2 (Приведение к 2НФ). Если в некоторых отношениях обнаружена зависимость атрибутов от части сложного ключа, то проводим декомпозицию этих отношений на несколько отношений следующим образом: те атрибуты, которые зависят от части сложного ключа выносятся в отдельное отношение вместе с этой частью ключа. В исходном отношении остаются все ключевые атрибуты.

Исходное отношение R(K1,K2,A1,…An,B1,…,Bn) .

Ключ {K1,K2} - сложный.

Функциональные зависимости:

{K1,K2} -> (A1,…An,B1,…,Bn) - зависимость всех атрибутов от ключа отношения.

{K1} -> (A1,…An) - зависимость некоторых атрибутов от части сложного ключа.

Декомпозированные отношения:

R1(K1,K2,B1,…,Bn) - остаток от исходного отношения. Ключ {K1,K2}.

R2(K1,A1,…An) - атрибуты, вынесенные из исходного отношения вместе с частью сложного ключа. Ключ K1.

Шаг 3 (Приведение к 3НФ). Если в некоторых отношениях обнаружена зависимость некоторых неключевых атрибутов от других неключевых атрибутов, то проводим декомпозицию этих отношений следующим образом: те неключевые атрибуты, которые зависят от других неключевых атрибутов выносятся в отдельное отношение. В новом отношении ключом становится детерминант функциональной зависимости:

Исходное отношение R(K,A1,…An,B1,…,Bn).

Ключ K.

Функциональные зависимости:

K -> (A1,…An,B1,…,Bn) - зависимость всех атрибутов от ключа отношения.

{A1,…An} -> (B1,…,Bn) - зависимость некоторых неключевых атрибутов от других неключевых атрибутов.

Декомпозированные отношения:

R1(K,A1,…,An) - остаток от исходного отношения. Ключ K.

R2(A1,…An,B1,…,Bn) - атрибуты, вынесенные из исходного отношения вместе с детерминантом функциональной зависимости. Ключ (A1,…An).

Замечание. Обычно при проектировании логической модели данных разработчики сразу строят отношения в 3НФ, не следуя алгоритму нормализации. Несмотря на это алгоритм важен и для практики:

- алгоритм показывает, какие проблемы возникают при разработке слабо нормализованных отношений.

- во многих случаях модель предметной области не бывает правильно разработана с первого шага. Забыли о чем-то упомянуть эксперты в предметной области, между разработчиками и экспертами возможно недопонимание, изменились правила и требования к предметной области. В результате могут появиться новые зависимости. В этом случае и необходимо использовать алгоритм нормализации, чтобы проверить, что отношения остались в 3НФ.

Корректность процедуры нормализации - декомпозиция без потерь.

При использовании алгоритма нормализации возникает вопрос – не будут ли при декомпозиции потеряны данные, и можно ли вернуться к исходным отношениям.

Определение 5. Проекция R[X] отношения R на множество атрибутов X называется ***собственной***, если множество атрибутов X является *собственным подмножеством* множества атрибутов отношения R (т.е. множество атрибутов X не совпадает с множеством *всех* атрибутов отношения R).

Определение 6. *Собственные* проекции R1 и R2 отношения R называются ***декомпозицией без потерь***, если отношение R *точно восстанавливается* из них при помощи естественного соединения *для любого состояния* отношения R:

R1 JOIN R2 = R

Теорема (Хеза). Пусть R(A,B,C) является отношением, и A,B,C - атрибуты или множества атрибутов этого отношения. Если имеется функциональная зависимость A -> B, то проекции R1[A,B] и R2[A,C] образуют декомпозицию без потерь.

Сравнение нормализованных и ненормализованных моделей

Критерий	Отношения слабо норм. (1НФ, 2НФ)	Отношения сильно норм. (3НФ)
Адекватность базы данных предметной области	(-)	ЛУЧШЕ (+)
Легкость разработки и сопровождения базы данных	СЛОЖНЕЕ (-)	ЛЕГЧЕ (+)
Скорость выполнения вставки, обновления, удаления	МЕДЛЕННЕЕ (-)	БЫСТРЕЕ (+)
Скорость выполнения выборки данных	БЫСТРЕЕ (+)	МЕДЛЕННЕЕ (-)

Таким образом, выбор степени нормализации отношений зависит от характера запросов, с которыми чаще всего обращаются к базе данных.

OLTP и OLAP-системы

Можно выделить некоторые классы систем, для которых больше подходят сильно или слабо нормализованные модели данных.

Сильно нормализованные модели данных хорошо подходят для так называемых ***OLTP-приложений*** (***On-Line Transaction Processing*** (***OLTP***)- ***оперативная обработка транзакций***). Типичными примерами OLTP-приложений являются системы складского учета, системы заказов билетов, банковские системы, выполняющие операции по переводу денег, и т.п. Основная функция подобных систем заключается в выполнении большого количества коротких транзакций. Практически все запросы к базе данных в OLTP-приложениях состоят из команд вставки, обновления, удаления. Критическим для OLTP-приложений является скорость и надежность выполнения коротких операций обновления данных. Чем выше уровень нормализации данных в OLTP-приложении, тем оно, как правило, быстрее и надежнее.

Другим типом приложений являются так называемые ***OLAP-приложения*** (***On-Line Analitical Processing*** (***OLAP***) - ***оперативная аналитическая обработка данных***). Это обобщенный термин, характеризующий принципы построения ***систем поддержки принятия решений*** (***Decision Support System*** - ***DSS***), ***хранилищ данных*** (***Data Warehouse***), ***систем интеллектуального анализа данных*** (***Data Mining***). Такие системы предназначены для нахождения зависимостей между данными (например, можно попытаться определить, как связан объем продаж товаров с характеристиками потенциальных покупателей), для проведения анализа "что если…". OLAP-приложения оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми их электронных таблиц или из других источников данных.

Такие системы характеризуются следующими признаками:

- Добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложения).
- Данные, добавленные в систему, обычно никогда не удаляются.
- Перед загрузкой данные проходят различные процедуры "очистки", связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления для одних и тех же понятий, данные могут быть некорректны, ошибочны.
- Запросы к системе являются нерегламентированными и, как правило, достаточно сложными. Очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса.
- Скорость выполнения запросов важна, но не критична.

Манипулирование реляционными данными

Оператор SELECT является самым важным для пользователя и самым сложным оператором SQL. Он предназначен для выборки данных из таблиц, т.е. он, собственно, и реализует одно их основных назначение базы данных - предоставлять информацию пользователю.

Оператор SELECT всегда выполняется над некоторыми таблицами, входящими в базу данных (постоянными, временными, представлениями). Результатом выполнения оператора SELECT всегда является таблица.

Порядок выполнения оператора SELECT

Для того чтобы понять, как получается результат выполнения оператора SELECT, рассмотрим концептуальную схему его выполнения. Эта схема является именно концептуальной, т.к. гарантируется, что результат будет таким, как если бы он выполнялся шаг за шагом в соответствии с этой схемой. На самом деле, реально результат получается более изощренными алгоритмами, которыми "владеет" конкретная СУБД.

Стадия 1. Выполнение одиночного оператора SELECT

Если в операторе присутствуют ключевые слова UNION, EXCEPT и INTERSECT, то запрос разбивается на несколько независимых запросов, каждый из которых выполняется отдельно:

Шаг 1 (FROM). Вычисляется прямое декартовое произведение всех таблиц, указанных в обязательном разделе FROM. Если таблица одна – результат она сама. В результате шага 1 получаем таблицу A.

select * **from** FILMS

FILM_ID	REG_ID	NAZV
1	1	Леон
2	2	Назад в будущее
3	2	Назад в будущее - 2
5	3	Пролетая над гнездом кукушки
6	4	Индиана Джонс и храм судьбы

Шаг 2 (WHERE). Если в операторе SELECT присутствует раздел WHERE, то сканируется таблица A, полученная при выполнении шага 1. При этом для каждой строки из таблицы A вычисляется условное выражение, приведенное в разделе WHERE. Только те строки, для которых условное выражение возвращает значение TRUE, включаются в результат. Если раздел WHERE опущен, то сразу переходим к шагу 3. Если в условном выражении участвуют вложенные подзапросы, то они вычисляются в соответствии с данной концептуальной схемой. В результате шага 2 получаем таблицу B.

```
select NAZV, 'Режиссер Роберт Земекис' REG from FILMS where
    REG_ID=2
```

```
NAZV                REG
=====================================
Леон                Режиссер Роберт Земекис
```

Шаг 3 (GROUP BY). Если в операторе SELECT присутствует раздел GROUP BY, то строки таблицы B, полученной на втором шаге, группируются в соответствии со списком группировки, приведенным в разделе GROUP BY. Если раздел GROUP BY опущен, то сразу переходим к шагу 4. В результате шага 3 получаем таблицу C.

select FILM_ID, COUNT(ACT_ID) ACT_COUNT **from** ZAN group by FILM_ID

FILM_ID	ACT_COUNT
1	2
2	2
3	2
5	1
6	1

Шаг 4 (HAVING). Если в операторе SELECT присутствует раздел HAVING, то группы, не удовлетворяющие условному выражению, приведенному в разделе HAVING, исключаются. Если раздел HAVING опущен, то сразу переходим к шагу 5. В результате шага 4 получаем таблицу D.

select FILM_ID, COUNT(ACT_ID) ACT_COUNT **from** ZAN **group by** FILM_ID **having** COUNT(ACT_ID) >1

FILM_ID	ACT_COUNT
1	2
2	2
3	2

Шаг 5 (SELECT). Каждая группа, полученная на шаге 4, генерирует одну строку результата следующим образом. Вычисляются все скалярные выражения, указанные в разделе SELECT. По правилам использования раздела GROUP BY, такие скалярные выражения должны быть одинаковыми для всех строк внутри каждой группы. Для каждой группы вычисляются значения агрегатных функций, приведенных в разделе SELECT. Если раздел GROUP BY отсутствовал, но в разделе SELECT есть агрегатные функции, то считается, что имеется всего одна группа. Если нет ни раздела GROUP BY, ни агрегатных функций, то считается, что имеется столько групп, сколько строк отобрано к данному моменту. В результате шага 5 получаем таблицу E, содержащую столько колонок, сколько элементов приведено в разделе SELECT и столько строк, сколько отобрано групп.

Стадия 2. Выполнение операций UNION, EXCEPT, INTERSECT

Если в операторе SELECT присутствовали ключевые слова UNION, EXCEPT и INTERSECT, то таблицы, полученные в результате выполнения 1-й стадии, объединяются, вычитаются или пересекаются.

Стадия 3. Упорядочение результата

Если в операторе SELECT присутствует раздел ORDER BY, то строки полученной на предыдущих шагах таблицы упорядочиваются в соответствии со списком упорядочения, приведенном в разделе ORDER BY.

select NAZV,COUNT(ACT_ID) ACT_COUNT **from** ZAN **join** FILMS **on** ZAN.FILM_ID=FILMS.FILM_ID **group by** NAZV **order by** NAZV

NAZV	ACT_COUNT
Индиана Джонс и храм судьбы	1
Леон	2
Назад в будущее	2
Назад в будущее - 2	2
Пролетая над гнездом кукушки	1

Как на самом деле выполняется оператор SELECT

Концептуальный алгоритм вычисления результата оператора SELECT выполнять непосредственно чрезвычайно накладно. Даже на самом первом шаге, когда вычисляется декартово произведение таблиц, приведенных в разделе FROM, может получиться таблица огромных размеров, причем практически большинство строк и колонок из нее будет отброшено на следующих шагах.

На самом деле в РСУБД имеется ***оптимизатор***, функцией которого является нахождение такого *оптимального алгоритма* выполнения запроса, который гарантирует получение правильного результата.

Схематично работу оптимизатора можно представить в виде последовательности нескольких шагов:

Шаг 1 (Синтаксический анализ). Поступивший запрос подвергается синтаксическому анализу. На этом шаге определяется, правильно ли вообще (с точки зрения синтаксиса SQL) сформулирован запрос. В ходе синтаксического анализа вырабатывается некоторое внутренне представление запроса, используемое на последующих шагах.

Шаг 2 (Преобразование в каноническую форму). Запрос во внутреннем представлении подвергается преобразованию в некоторую каноническую форму. При преобразовании к канонической форме используются как синтаксические, так и семантические преобразования. Синтаксические преобразования (например, приведения логических выражений к конъюнктивной или дизъюнктивной нормальной форме, замена выражений "x AND NOT x" на "FALSE", и т.п.) позволяют получить новое внутренне представление запроса, синтаксически *эквивалентное* исходному, но стандартное в некотором смысле. Семантические преобразования используют дополнительные знания, которыми владеет система, например, ограничения целостности. В результате семантических преобразований получается запрос, синтаксически *не эквивалентный* исходному, но дающий *тот же самый результат*.

Шаг 3 (Генерация планов выполнения запроса и выбор оптимального плана). На этом шаге оптимизатор генерирует множество возможных планов выполнения запроса. Каждый план строится как комбинация низкоуровневых процедур доступа к данным из таблиц, методам соединения таблиц. Из всех сгенерированных планов выбирается план, обладающий минимальной стоимостью. При этом анализируются данные о наличии индексов у таблиц, статистических данных о распределении значений в таблицах, и т.п. Стоимость плана это, как правило, сумма стоимостей выполнения отдельных низкоуровневых процедур, которые используются для его выполнения.

Шаг 4. (Выполнение плана запроса). На этом шаге план, выбранный на предыдущем шаге, передается на реальное выполнение.

Во многом качество конкретной СУБД определяется качеством ее оптимизатора. Качество оптимизатора определяется тем, какие методы преобразований он может использовать, какой статистической и иной информацией о таблицах он располагает, какие методы для оценки стоимости выполнения плана он знает.

Транзакции и целостность баз данных

Транзакция - это неделимая, с точки зрения воздействия на СУБД, последовательность операций манипулирования данными. Для пользователя транзакция выполняется по принципу "*все или ничего*", т.е. либо транзакция выполняется целиком и переводит БД из одного *целостного состояния* в другое *целостное состояние*, либо, если по каким-либо причинам, одно из действий транзакции невыполнимо, или произошло какое-либо нарушение работы системы, БД возвращается в исходное состояние, которое было до начала транзакции (происходит откат транзакции).

Транзакции важны как в многопользовательских, так и в однопользовательских системах.

В однопользовательских системах транзакции - это логические единицы работы, после выполнения которых БД остается *в целостном состоянии*. Транзакции также являются *единицами восстановления* данных после сбоев - восстанавливаясь, система ликвидирует следы транзакций, не успевших успешно завершиться в результате программного или аппаратного сбоя.

В многопользовательских системах, кроме того, транзакции служат для обеспечения *изолированной* работы отдельных пользователей - пользователям, одновременно работающим с одной БД, кажется, что они работают как бы в однопользовательской системе и не мешают друг другу.

Пример возможного нарушения целостности БД

В системе продажи билетов авиакомпании хранятся данные о количестве непроданных билетов рейсы, и списки пассажиров рейсов. Список рейсов хранится в таблице FLIGHT (FL_Id, FL_Date, FL_Kol), где FL_Id – идентификатор рейса, FL_Date – дата рейса, FL_Kol - количество непроданных билетов. Список пассажиров рейсов хранится в таблице Passenger(PS_Id,PS_Num,FL_Id,FL_Date), где PS_Id – идентификатор пассажира, PS_Num – номер билета, FL_Id, FL_Date – идентификатор и дата рейса, которым полетит пассажир.

Ограничение целостности этой базы данных состоит в том, что поле FL_Kol не может заполняться произвольными значениями - это поле должно содержать количество непроданных билетов.

FL_Id	FL_Date	FL_Kol
1	10.02.2009	10
1	12.02.2009	12
2	15.03. 2009	20

PS_Id	PS_Num	FL_Id	FL_Date
1	10	1	10.02.2009
2	12	2	15.03. 2009

С учетом этого ограничения можно заключить, что продажа билета *не может быть выполнена одной операцией*. При вставке записи о новом пассажире рейса необходимо одновременно уменьшить значение поля FL_Kol:

Шаг 1. Вставить сотрудника в таблицу Passenger: INSERT INTO Passenger (5, 15, 1,’10.02.2009’)

Шаг 2. Уменьшить значение поля FL_Kol: UPDATE Flight SET FL_Kol=FL_Kol WHERE FL_Id=1 and FL_Date = ’10.02.2009’

Если после выполнения первой операции и до выполнения второй произойдет сбой системы, то будет выполнена только первая операция и БД остается в нецелостном состоянии.

Транзакции и целостность баз данных

Опр. 1. ***Транзакция*** - это последовательность операторов манипулирования данными, выполняющаяся *как единое целое* (все или ничего) и переводящая БД *из одного целостного состояния в другое целостное состояние*.

Транзакция обладает четырьмя важными свойствами, известными как ***свойства АСИД***:

(А) Атомарность. Транзакция выполняется как атомарная операция - либо выполняется вся транзакция целиком, либо она целиком не выполняется.

(С) Согласованность. Транзакция переводит БД из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние. Внутри транзакции согласованность БД может нарушаться.

(И) Изоляция. Транзакции разных пользователей не должны мешать друг другу (например, как если бы они выполнялись строго по очереди).

(Д) Долговечность. Если транзакция выполнена, то результаты ее работы должны сохраниться в БД, даже если в следующий момент произойдет сбой системы.

Транзакция обычно начинается автоматически с момента присоединения пользователя к СУБД и продолжается до тех пор, пока не произойдет одно из следующих событий:

- Подана команда COMMIT WORK (зафиксировать транзакцию).
- Подана команда ROLLBACK WORK (откатить транзакцию).
- Произошло отсоединение пользователя от СУБД.
- Произошел сбой системы.

Команда COMMIT WORK завершает текущую транзакцию и автоматически начинает новую транзакцию.

Команда ROLLBACK WORK приводит к тому, что все изменения, сделанные текущей транзакцией откатываются, т.е. отменяются так, *как будто их вообще не было*. При этом автоматически начинается новая транзакция.

При отсоединении пользователя от СУБД происходит автоматическая фиксация транзакций.

При сбое системы происходят более сложные процессы, которые будут рассмотрены позднее.

Ограничения целостности

Свойство (С) - согласованность транзакций определяется наличием понятия согласованности БД.

Опр. 2. ***Ограничение целостности*** - это некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния БД.

Примеры ограничений целостности:

1) Возраст военнослужащего не может быть < 18 и > 60 лет.
2) Каждый студент имеет уникальный номер зачетки.
3) Режиссер одновременно снимает только один фильм.
4) Зарплата сотрудника состоит из оклада и премии.

Некоторые из ограничений целостности являются ограничениями реляционной модели данных . 2-е это ограничение, реализующее целостность сущности. 3-е представляет ограничение, реализующее ссылочную целостность. Другие ограничения являются достаточно произвольными утверждениями (примеры 1 и 4). Любое ограничение целостности является *семантическим* понятием, т.е. появляется как следствие определенных свойств объектов предметной области и/или их взаимосвязей.

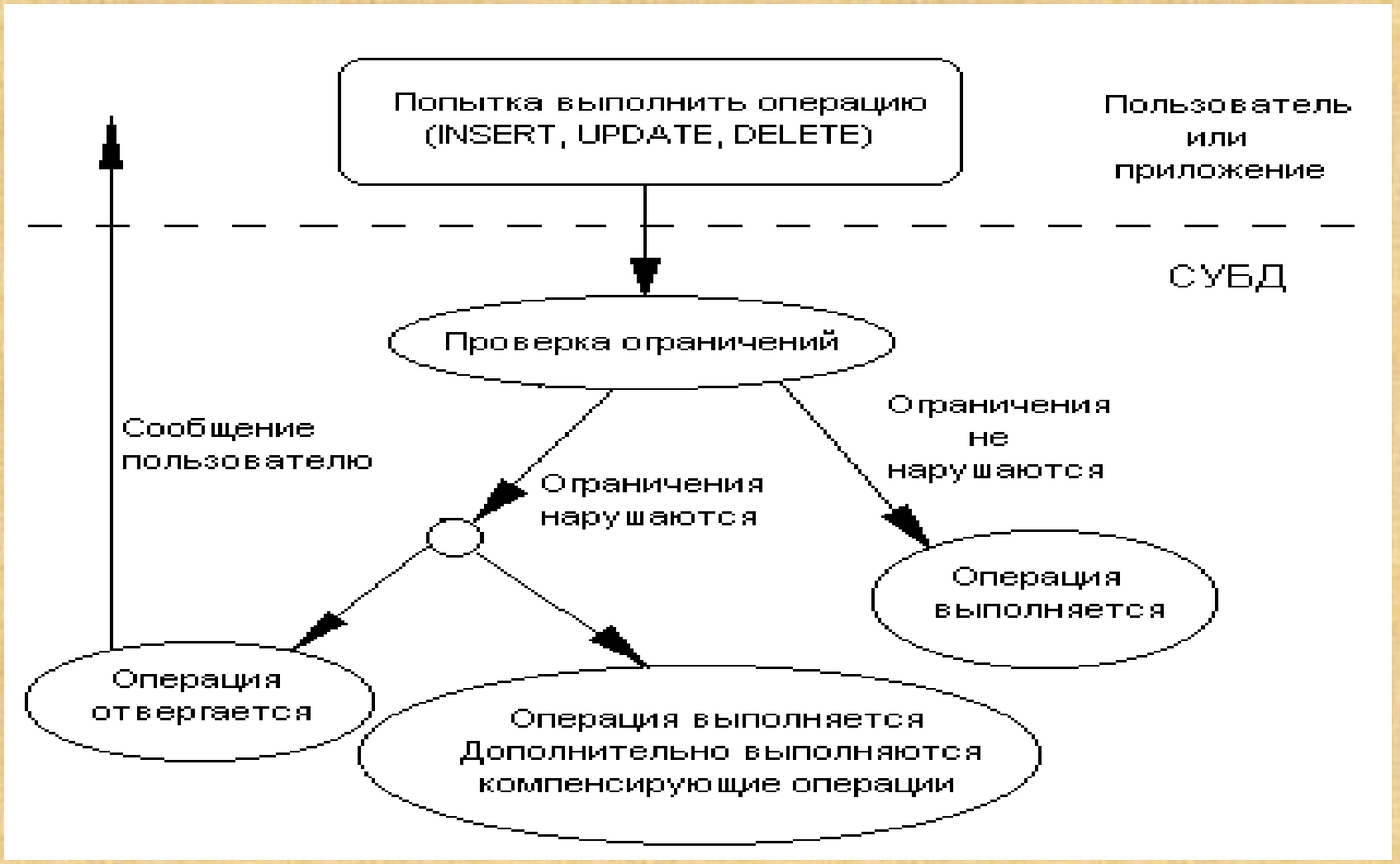
Опр. 3. База данных находится в ***согласованном (целостном) состоянии***, если выполнены (удовлетворены) все ограничения целостности, определенные для базы данных.

Вместе с понятием целостности базы данных возникает понятие ***реакции системы на попытку нарушения целостности***. Система должна не только проверять, не нарушаются ли ограничения в ходе выполнения различных операций, но и должным образом реагировать, если операция приводит к нарушению целостности. Имеется два типа реакции на попытку нарушения целостности:

1. *Отказ* выполнить "незаконную" операцию.
2. Выполнение *компенсирующих* действий.

В поле "Возраст_военнослужащего" должны быть целые числа в диапазоне от 18 до 65, поэтому будет отвергнута попытка ввести значение возраста 66.

Или система допускает вставку записи о продаже билета пассажиру (что приводит к нарушению целостности базы данных), но *автоматически* производит компенсирующие действия, изменяя значение поля FL_Kol в таблице Flight.



В некоторых случаях система может не выполнять проверку на нарушение ограничений, а сразу выполнять компенсирующие операции (при продаже билета целостность обязательно будет нарушена).

Классификация ограничений целостности

Ограничения целостности можно классифицировать несколькими способами:

- По способам реализации.
- По времени проверки.
- По области действия.

Классификация ограничений целостности по способам реализации

Каждая система обладает своими средствами поддержки ограничений целостности. Различают два способа реализации:

- Декларативная поддержка ограничений целостности.
- Процедурная поддержка ограничений целостности.

Опр. 4. ***Декларативная поддержка ограничений целостности*** заключается в определении ограничений средствами языка определения данных (DDL - Data Definition Language). Обычно средства декларативной поддержки целостности (если они имеются в СУБД) определяют ограничения на значения доменов и атрибутов, целостность сущностей (потенциальные ключи отношений) и ссылочную целостность (целостность внешних ключей).

```
 create table Passenger
(PS_Id integer not null,
 PS_Num  integer not null,
 FL_Id references Flight(FL_Id) on update cascade on delete cascade,
 FL_Date references Flight(FL_Date) on update cascade on delete cascade)
```

Объявлены следующие ограничения целостности:

Поле PS_Id не может содержать null-значений

Поле PS_Num не может содержать null-значений.

Поля FL_Id, FL_Date являются внешними ссылками на родительскую таблицу Flight, причем, при изменении или удалении строки в родительской таблице *каскадно* должны быть внесены соответствующие изменения в дочернюю таблицу.

Опр. 5. ***Процедурная поддержка ограничений целостности*** заключается в использовании триггеров и хранимых процедур.

Не все ограничения целостности можно реализовать декларативно. Примером такого ограничения может служить требование, утверждающее, что поле FL_Kol таблицы Flight должно содержать количество непроданных билетов. Для реализации этого ограничения необходимо создать триггер, запускающийся при вставке, модификации и удалении записей в таблице Flight, который корректно изменяет значение поля FL_Kol. Например, при вставке в таблицу Passenger новой строки, триггер уменьшает на единицу значение поля FL_Kol, а при удалении строки - увеличивает.

Классификация ограничений целостности по времени проверки

По времени проверки ограничения делятся на:

- Немедленно проверяемые ограничения.
- Ограничения с отложенной проверкой.

Опр. 6. ***Немедленно проверяемые ограничения*** проверяются непосредственно в момент выполнения операции, могущей нарушить ограничение (проверка уникальности потенциального ключа). Если ограничение нарушается, то такая операция отвергается. Транзакция, внутри которой произошло нарушение немедленно проверяемого утверждения целостности, обычно откатывается.

Опр. 7. ***Ограничения с отложенной проверкой*** проверяется в момент фиксации транзакции оператором COMMIT WORK. Внутри транзакции ограничение может не выполняться. Если в момент фиксации транзакции обнаруживается нарушение ограничения с отложенной проверкой, то транзакция откатывается. Примером ограничения, которое не может быть проверено немедленно является ограничение количество непроданных билетов на отношение Flight.

Классификация ограничений целостности по области действия

По области действия ограничения делятся на:

- Ограничения домена
- Ограничения атрибута
- Ограничения кортежа
- Ограничения отношения
- Ограничения базы данных

Ограничения домена

Опр. 8. ***Ограничения целостности домена*** представляют собой ограничения, накладываемые только на допустимые значения домена. Ограничения домена не проверяются.

Например, ограничение домена «Возраст_военнослужащего» это условие «Возраст не менее 18 и не более 60».

Ограничения атрибута

Опр. 9. ***Ограничение целостности атрибута*** представляют собой ограничения, накладываемые на допустимые значения атрибута вследствие того, что атрибут основан на каком-либо домене. Ограничения атрибута *проверяются*.

Ограничения кортежа

Опр. 10. ***Ограничения целостности кортежа*** представляют собой ограничения, накладываемые на допустимые значения *отдельного* кортежа отношения, и *не являющиеся* ограничением целостности атрибута. Требование, что ограничение относится к *отдельному* кортежу отношения, означает, что для его проверки *не требуется* никакой информации о других кортежах отношения.

Атрибут "Возраст_военнослужащего" в таблице "Спецгруппа", может иметь дополнительное ограничение "Возраст_военнослужащего не менее 25 и не более 45", помимо того, что этот атрибут уже имеет ограничение, определяемое доменом - "Возраст_военнослужащего" не менее 18 и не более 60".

Ограничения отношения

Опр. 11. ***Ограничения целостности отношения*** это ограничения, накладываемые только на допустимые значения *отдельного* отношения, и *не являющиеся* ограничением целостности кортежа (т.е. для его проверки не требуется информации о других отношениях)

Ограничение целостности сущности, задаваемое потенциальным ключом отношения, является ограничением отношения.

Ограничения базы данных

Опр. 12. ***Ограничения целостности базы данных*** представляют ограничения, накладываемые на значения двух или более связанных между собой отношений (в том числе отношение может быть связано само с собой).

Ограничение целостности ссылок, задаваемое внешним ключом отношения, является ограничением базы данных.

Транзакции и параллелизм

Рассмотрим возможности параллельного выполнения транзакций несколькими пользователями, т.е. свойство (И) - изолированность транзакций.

Современные СУБД являются многопользовательскими системами, т.е. допускают параллельную одновременную работу большого количества пользователей. При этом пользователи не должны мешать друг другу. Работа СУБД должна быть организована так, чтобы у пользователя складывалось впечатление, что их транзакции выполняются независимо от транзакций других пользователей.

Простейший и очевидный способ обеспечить такую иллюзию у пользователя состоит в том, чтобы все поступающие транзакции выстраивать в единую очередь и выполнять строго по очереди. Такой способ не годится по очевидным причинам - теряется преимущество параллельной работы. Таким образом, транзакции необходимо выполнять одновременно, но так, чтобы результат был бы такой же, как если бы транзакции выполнялись по очереди.

Работа транзакций в смеси

Транзакция рассматривается как последовательность элементарных атомарных операций. Атомарность отдельной элементарной операции состоит в том, что СУБД гарантирует, что, с точки зрения пользователя, будут выполнены два условия:

1. Эта операция будет выполнена целиком или не выполнена вовсе (атомарность - все или ничего).
2. Во время выполнения этой операции не выполняются никакие другие операции других транзакций (строгая очередность элементарных операций).

Например, элементарными операциями транзакции будут считывание страницы данных с диска или запись страницы данных на диск (страница данных - это минимальная единица для дисковых операций СУБД). Условие 2 на самом деле является именно логическим условием, т.к. реально система может выполнять несколько различных элементарных операций в один и тот же момент. Например, данные могут храниться на нескольких физически различных дисках и операции чтения-записи на эти диски могут выполняться одновременно.

Элементарные операции различных транзакций могут выполняться в произвольной очередности (конечно, внутри каждой транзакции последовательность элементарных операций этой транзакции является строго определенной). Например, если есть несколько транзакций, состоящих из последовательности элементарных операций:

T = {T1,T2,…,Tn},

Q = {Q1,Q2,…,Qm},

то реальная последовательность, в которой СУБД выполняет эти транзакции может быть, например, такой:

{T1,T2,Q1,T3,Q2,…}

Опр. 11/ Набор из нескольких транзакций, элементарные операции которых чередуются друг с другом, называется ***смесью транзакций***.

Опр.12. Последовательность, в которой выполняются элементарные операции заданного набора транзакций, называется ***графиком запуска*** набора транзакций.

Проблемы параллельной работы транзакций

Различают три основные проблемы параллелизма:

- *Проблема потери результатов обновления*
- *Проблема незафиксированной зависимости* (*чтение "грязных" данных*, *неаккуратное считывание*).
- *Проблема несовместимого анализа*

Рассмотрим две транзакции, A и B, запускающиеся в соответствии с некоторыми графиками. Пусть транзакции работают с некоторыми объектами базы данных, например со строками таблицы. Операцию чтение строки *P* будем обозначать $P=P_0$, где P_0 - прочитанное значение. Операцию записи значения P_1 в строку *P* будем обозначать $P_1->P$.

Проблема потери результатов обновления

Две транзакции по очереди записывают некоторые данные в одну и ту же строку и фиксируют изменения.

Транзакция A	Время	Транзакция B
Чтение $P=P_0$	t1	
	t2	Чтение $P=P_0$
Запись $P_1 \rightarrow P$	t3	
	t4	Запись $P_2 \rightarrow P$
Фиксация	t5	
	t6	Фиксация
Потеря результатов обновления		

Результат. После окончания обеих транзакций, строка *P* содержит значение P_2, занесенное более поздней транзакцией B. Транзакция A ничего не знает о существовании транзакции B, и естественно ожидает, что в строке содержится значение P_1. Таким образом, транзакция A потеряла результаты своей работы.

Проблема незафиксированной зависимости (чтение "грязных" данных, неаккуратное считывание)

Транзакция B изменяет данные в строке. После этого транзакция A читает измененные данные и работает с ними. Транзакция B откатывается и восстанавливает старые данные.

Транзакция A	Время	Транзакция B
	t1	Чтение $P=P_0$
	t2	Запись $P_1 \rightarrow P$
Чтение $P=P_1$	t3	
Работа с прочитанными данными P_1	t4	
	t5	Откат транзакции $P_0 \rightarrow P$
Фиксация	t6	
Работа с «грязными» данными		

Результат. Транзакция A в своей работе использовала данные, которых нет в БД. Более того, A использовала данные, которых нет, и *не было* в БД! Действительно, после отката транзакции B, должна восстановиться ситуация, как если бы транзакция B вообще *никогда не выполнялась*. Т.о., результаты работы транзакции A некорректны.

Проблема несовместимого анализа

Проблема несовместимого анализа включает несколько различных вариантов:

- ***Неповторяемое считывание***.
- ***Фиктивные элементы*** (***фантомы***).
- ***Собственно несовместимый анализ***.

Неповторяемое считывание

Транзакция А дважды читает одну и ту же строку. Между этими чтениями вклинивается транзакция В, которая изменяет значения в строке.

Транзакция А	Время	Транзакция В
Чтение $P=P_0$	t1	
	t2	Чтение $P=P_0$
	t3	Запись $P_1 \rightarrow P$
	t4	Фиксация
Повторное чтение $P=P_1$	t5	
Фиксация	t6	
Неповторяемое считывание		

Транзакция А ничего не знает о существовании транзакции В, и, т.к. сама она не меняет значение в строке, то ожидает, что после повторного чтения значение будет тем же самым.

Результат. Транзакция А работает с данными, которые, с точки зрения транзакции А, самопроизвольно изменяются.

Фиктивные элементы (фантомы)

Транзакция А дважды выполняет выборку строк с одним и тем же условием. Между выборками вклинивается транзакция В, которая добавляет новую строку, удовлетворяющую условию отбора.

Транзакция А	Время	Транзакция В
Выборка строк , удовлетворяющих условию *U* (отобрано *n* строк)	t1	
	t2	Вставка новой строки удовлетворяющей условию *U*
	t3	Фиксация
Выборка строк , удовлетворяющих условию *U* (отобрано *n+1* строка)	t4	
Фиксация	t5	
Появились строки, которых раньше не было		

Результат. Транзакция А ничего не знает о существовании транзакции В, и, т.к. сама она не меняет ничего в БД, то ожидает, что после повторного отбора будут отобраны те же самые строки.

Собственно несовместимый анализ

В смеси присутствуют две транзакции – длинная и короткая.

Длинная выполняет некоторый анализ по всей таблице, например, подсчитывает общую сумму денег на счетах клиентов банка для главбуха. Пусть на всех счетах находятся по $100. Короткая транзакция в этот момент выполняет перевод $50 с одного счета на другой так, что общая сумма по всем счетам не меняется.

Транзакция A	Время	Транзакция B
Чтение счета *P1=100* и *SUM=100*	t1	
	t2	Снятие денег с *P3. P3:100 ->50*
	t3	Помещение денег на *P1. P1:100 ->150*
	t4	Фиксация
Чтение счета *P2=100* и *SUM=200*	t5	
Чтение счета *P3=50* и *SUM=250*	t6	
Фиксация	t7	
Сумма по счетам 250, а д. б. 300		

Результат. Хотя транзакция В все сделала правильно - деньги переведены без потери, но в результате транзакция А подсчитала неверную общую сумму.

Т.к. транзакции по переводу денег идут обычно непрерывно, то в данной ситуации следует ожидать, что главный бухгалтер *никогда* не узнает, сколько же денег в банке.

Конфликты между транзакциями

Итак, анализ проблем параллелизма показывает, что если не предпринимать специальных мер, то при работе в смеси нарушается свойство (И) транзакций - изолированность. Транзакции реально мешают друг другу получать правильные результаты.

Однако не всякие транзакции мешают друг другу. Очевидно, что транзакции не мешают друг другу, если они обращаются к *разным данным* или выполняются в *разное время*.

Опр. 13. Транзакции называются ***конкурирующими***, если они пересекаются по времени и обращаются к одним и тем же данным.

В результате конкуренции за данными между транзакциями возникают ***конфликты доступа*** к данным. Различают следующие виды конфликтов:

- ***W-W (Запись - Запись)***. Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается изменить этот объект. Результат - потеря обновления.
- ***R-W (Чтение - Запись)***. Первая транзакция прочитала объект и не закончилась. Вторая транзакция пытается изменить этот объект. Результат - несовместимый анализ (неповторяемое считывание).
- ***W-R (Запись - Чтение)***. Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается прочитать этот объект. Результат - чтение "грязных" данных.

Конфликты типа R-R (Чтение - Чтение) отсутствуют, т.к. данные при чтении не изменяются.

Другие проблемы параллелизма (фантомы и собственно несовместимый анализ) являются более сложными, т.к. принципиальное отличие их в том, что они не могут возникать при работе с одним объектом. Для возникновения этих проблем требуется, чтобы транзакции работали с целыми *наборами* данных.

Опр. 14. График запуска набора транзакций называется ***последовательным***, если транзакции выполняются строго по очереди, т.е. элементарные операции транзакций не чередуются друг с другом.

Опр. 15. Если график запуска набора транзакций содержит чередующиеся элементарные операции транзакций, то такой график называется ***чередующимся***.

При выполнении последовательного графика гарантируется, что транзакции выполняются правильно, т.е. при последовательном графике транзакции не "чувствуют" присутствия друг друга.

Опр. 16. Два графика называются ***эквивалентными***, если при их выполнении будет получен один и тот же результат, *независимо от начального состояния базы данных*.

Опр. 17. График запуска транзакции называется ***верным*** (***сериализуемым***), если он эквивалентен какому-либо последовательному графику.

Задача обеспечения изолированной работы пользователей не сводится просто к нахождению правильных (сериальных) графиков запусков транзакций. Если бы этого было достаточно, то лучшим был бы простейший способ сериализации - ставить транзакции в общую очередь по мере их поступления и выполнять строго последовательно. Таким способом автоматически будет получен правильный (сериальный) график. Проблема в том, что этот график будет неоптимальным с точки зрения общей производительности системы.

Рассмотрим другой случай - попытаемся достичь оптимального графика - т.е. графика с максимальной эффективностью выполнения транзакций. Для этого сначала нужно уточнить понятие "оптимальность". С каждым возможным графиком запуска транзакций мы можем связать значение некоей стоимостной функции. В качестве стоимостной функции можно взять, например, суммарное время выполнения всех транзакций в наборе. Время выполнения одной транзакции считается от момента, когда транзакция возникла и до момента, когда транзакция выполнила свою последнюю элементарную операцию. Это время складывается из следующих компонентов:

1. Время ожидания начала транзакции - то время, которое проходит от момента, когда транзакция возникла до момента, когда началась реально выполняться ее первая элементарная операция.

2. Сумма времен выполнения элементарных операций транзакции.

3. Сумма времен всех элементарных операций других транзакций, вклинившихся между элементарными операциями транзакции.

Оптимальным будет график, дающий минимум стоимостной функции. Очевидно, оптимальность графика запуска зависит от выбора стоимостной функции, т.е. график, оптимальный с точки зрения одних критериев (например, с точки зрения приведенной функции стоимости) не будет оптимальным с точки зрения других критериев (например, с точки зрения достижения максимально быстрого начала выполнения каждой транзакции).

Т.к. транзакции не мешают друг другу, если они обращаются к разным данным или выполняются в разное время, то имеется два способа разрешить конкуренцию между поступающими в произвольные моменты транзакциями:

1. "Притормаживать" некоторые из поступающих транзакций настолько, насколько это необходимо для обеспечения правильности смеси транзакций в каждый момент времени (т.е. обеспечить, чтобы конкурирующие транзакции выполнялись *в разное время*).

2. Предоставить конкурирующим транзакциям "разные" экземпляры данных (т.е. обеспечить, чтобы конкурирующие транзакции работали *с разными версиями данными*).

Первый метод - "притормаживание" транзакций - реализуется путем использованием блокировок различных видов или метода временных меток.

Второй метод - предоставление разных версий данных - реализуется путем использованием данных из журнала транзакций.

Блокировки

Основная идея блокировок заключается в том, что если для выполнения некоторой транзакции необходимо, чтобы некоторый объект не изменялся без ведома этой транзакции, то этот объект должен быть заблокирован, т.е. доступ к этому объекту со стороны других транзакций ограничивается на время выполнения транзакции, вызвавшей блокировку.

Различают два типа блокировок:

Монопольные блокировки (***X-блокировки***, ***X-locks*** - eXclusive locks) - блокировки без взаимного доступа (блокировка записи).

Разделяемые блокировки (***S-блокировки***, ***S-locks*** - Shared locks) - блокировки с взаимным доступом (блокировка чтения).

Если транзакция A блокирует объект при помощи X-блокировки, то всякий доступ к этому объекту со стороны других транзакций отвергается.

Если транзакция A блокирует объект при помощи S-блокировки, то запросы со стороны других транзакций на X-блокировку этого объекта будут отвергнуты, запросы со стороны других транзакций на S-блокировку этого объекта будут приняты.

Правила взаимного доступа к заблокированным объектам можно представить в виде следующей ***матрицы совместимости блокировок***. Если транзакция A наложила блокировку на некоторый объект, а транзакция B после этого пытается наложить блокировку на этот же объект, то успешность блокирования транзакцией B объекта описывается таблицей:

	Транзакция B пытается наложить блокировку:	
Транзакция A наложила блокировку:	S-блокировку	X-блокировку
S-блокировку	**Да**	**НЕТ (Конфликт R-W)**
X-блокировку	**НЕТ (Конфликт W-R)**	**НЕТ (Конфликт W-W)**

Три случая, когда транзакция B не может блокировать объект, соответствуют трем видам конфликтов между транзакциями.

Доступ к объектам базы данных на чтение и запись должен осуществляться в соответствии со следующим ***протоколом доступа к данным***:

1. Прежде чем прочитать объект, транзакция должна наложить на этот объект S-блокировку.

2. Прежде чем обновить объект, транзакция должна наложить на этот объект X-блокировку. Если транзакция уже заблокировала объект S-блокировкой (для чтения), то перед обновлением объекта S-блокировка должна быть заменена X-блокировкой.

3. Если блокировка объекта транзакцией B отвергается оттого, что объект уже заблокирован транзакцией A, то транзакция B переходит в ***состояние ожидания***. Транзакция B будет находиться в состоянии ожидания до тех пор, пока транзакция A не снимет блокировку объекта.

4. X-блокировки, наложенные транзакцией A, сохраняются до конца транзакции A.

Решение проблем параллелизма при помощи блокировок

Проблема потери результатов обновления

Две транзакции по очереди записывают некоторые данные в одну и ту же строку и фиксируют изменения.

Транзакция A	Время	Транзакция B
S-блокировка *P* успешна	t1	
Чтение $P=P_0$	t2	
	t3	S-блокировка *P* успешна
	t4	Чтение $P=P_0$
X-блокировка *P* отвергается	t5	
Ожидание	t6	X-блокировка *P* отвергается
Ожидание	t7	Ожидание
Ожидание		***Ожидание***

Обе транзакции ожидают друг друга и не могут продолжаться. Возникла ситуация *тупика*.

Проблема незафиксированной зависимости

Тр-я В изменяет данные в строке. После этого тр-я А читает измененные данные и работает с ними. Тр-я В откатывается и восстанавливает старые данные.

Транзакция А	Время	Транзакция В
	t1	S-блокировка *P* успешна
	t2	Чтение $P=P_0$
	t3	X-блокировка *P* успешна
	t4	Запись $P_1 \rightarrow P$
S-блокировка *P* отвергается	t5	
Ожидание	t6	Откат транзакции $P_0 \rightarrow P$ (Блокировка снимается)
S-блокировка *P* успешна	t7	
Чтение $P=P_0$	t8	
Работа с прочитанными данными P_0	t9	
Все правильно		

Тр-я А притормозилась до окончания (отката) тр-и В. После этого тр-я А продолжила работу в обычном режиме и работала с правильными данными. Конфликт разрешен за счет некоторого увеличения времени работы тр-и А (потрачено время на ожидание снятия блокировки тр-й В).

Проблема несовместимого анализа

Неповторяемое считывание

Транзакция А дважды читает одну и ту же строку. Между этими чтениями вклинивается транзакция В, которая изменяет значения в строке.

Транзакция А	Время	Транзакция В
S-блокировка *P* успешна	t1	
Чтение $P=P_0$	t2	
	t3	X-блокировка *P* отвергается
Ожидание	t4	Ожидание
Повторное чтение $P=P_1$	t5	Ожидание
Фиксация (Блокировка снимается)	t6	Ожидание
	t7	X-блокировка *P* успешна
	t8	Чтение $P=P_0$
	t9	Запись $P_1 \rightarrow P$
	t10	Фиксация (Блокировка снимается)
Все правильно		

Фиктивные элементы (фантомы)

Транзакция А дважды выполняет выборку строк с одним и тем же условием. Между выборками вклинивается транзакция В, которая добавляет новую строку, удовлетворяющую условию отбора.

Транзакция А	**Время**	**Транзакция В**
S-блокировка строк, удовлетворяющих условию *U* (заблокировано *n* строк)	t1	
Выборка строк , удовлетворяющих условию *U* (отобрано *n* строк)	t2	
	t3	Вставка новой строки удовлетворяющей условию *U*
	t4	Фиксация
S-блокировка строк, удовлетворяющих условию *U* (заблокировано *n+1* строка)	t5	
Выборка строк , удовлетворяющих условию *U* (отобрано *n+1* строка)	t6	
Фиксация	t7	
Появились строки, которых раньше не было		

Собственно несовместимый анализ

Длинная транзакция выполняет некоторый анализ по всей таблице, например, подсчитывает общую сумму денег на счетах клиентов банка для главного бухгалтера. Пусть на всех счетах находятся одинаковые суммы, например, по $100. Короткая транзакция в этот момент выполняет перевод $50 с одного счета на другой так, что общая сумма по всем счетам не меняется.

Транзакция A	**Время**	**Транзакция B**
S-блокировка *P1* успешна	t1	
Чтение счета *P1=100* и *SUM=100*	t2	
	t3	X-блокировка *P3* успешна
	t4	Снятие денег с *P3. P3:100 ->50*
	t5	X-блокировка *P1* отвергается
	t6	Ожидание
S-блокировка *P2* успешна	t7	Ожидание
Чтение счета *P2=100* и *SUM=200*	t8	Ожидание
S-блокировка *P3* отвергается	t9	Ожидание
Ожидание		***Ожидание***

Разрешение тупиковых ситуаций

Итак, при использовании протокола доступа к данным с использованием блокировок часть проблем разрешилось (не все), но возникла новая проблема – тупики/

Общий вид ***тупика*** (***dead locks***) следующий:

Транзакция А	Время	Транзакция В
Блокировка объекта P_1- успешна	t1	
	t2	Блокировка объекта P_2- успешна
Блокировка объекта P_2 - конфликтует с блокировкой, наложенной транзакцией В	t3	
Ожидание	t4	Блокировка объекта P_1 - конфликтует с блокировкой, наложенной транзакцией А
Ожидание	t5	Ожидание
Ожидание		***Ожидание***

Ситуация тупика может возникать при наличии не менее двух транзакций, каждая из которых выполняет не менее двух операций. На самом деле в тупике может участвовать много транзакций, ожидающих друг друга.

Методом разрешения тупиковой ситуации является откат одной из транзакций (транзакции-жертвы) так, чтобы другие транзакции продолжили свою работу. После разрешения тупика, транзакцию, выбранную в качестве жертвы можно повторить заново.

Можно представить два принципиальных подхода к обнаружению тупиковой ситуации и выбору транзакции-жертвы:

1. СУБД не следит за возникновением тупиков. Транзакции сами принимают решение, быть ли им жертвой.
2. За возникновением тупиковой ситуации следит сама СУБД, она же принимает решение, какой транзакцией пожертвовать.

Первый подход характерен для так называемых настольных СУБД (FoxPro и т.п.). Этот метод является более простым и не требует дополнительных ресурсов системы. Для транзакций задается время ожидания (или число попыток), в течение которого транзакция пытается установить нужную блокировку. Если за указанное время (или после указанного числа попыток) блокировка не завершается успешно, то транзакция откатывается (или генерируется ошибочная ситуация). За простоту этого метода приходится платить тем, что транзакции-жертвы выбираются, вообще говоря, случайным образом. В результате из-за одной простой транзакции может откатиться очень дорогая транзакция, на выполнение которой уже потрачено много времени и ресурсов системы.

Второй способ характерен для промышленных СУБД (ORACLE, MS SQL Server и т.п.). В этом случае система сама следит за возникновением ситуации тупика, путем построения (или постоянного поддержания) графа ожидания транзакций.

Граф ожидания транзакций - это ориентированный двудольный граф, в котором существует два типа вершин - вершины, соответствующие транзакциям, и вершины, соответствующие объектам захвата. Ситуация тупика возникает, если в графе ожидания транзакций имеется хотя бы один цикл. Одну из транзакций, попавших в цикл, необходимо откатить, причем, система сама может выбрать эту транзакцию в соответствии с некоторыми стоимостными соображениями (например, самую короткую, или с минимальным приоритетом и т.п.).

Преднамеренные блокировки

Как видно из анализа поведения транзакций, при использовании протокола доступа к данным не решается проблема фантомов. Это происходит оттого, что были рассмотрены только блокировки на уровне строк. Можно рассматривать блокировки и других объектов базы данных:

- Блокировка самой базы данных.
- Блокировка файлов базы данных.
- Блокировка таблиц базы данных.
- Блокировка страниц (Единиц обмена с диском, обычно 2-16 Кб. На одной странице содержится несколько строк одной или нескольких таблиц).
- Блокировка отдельных строк таблиц.
- Блокировка отдельных полей.

Кроме того, можно блокировать индексы, заголовки таблиц или другие объекты.

Чем крупнее объект блокировки, тем меньше возможностей для параллельной работы. С другой стороны уменьшаются накладные расходы системы и решаются проблем, не решаемые с использованием блокировок менее крупных объектов.

При использовании блокировок объектов разной величины возникает проблема обнаружения уже наложенных блокировок. Если транзакция А пытается заблокировать таблицу, то необходимо иметь информацию, не наложены ли уже блокировки на уровне строк этой таблицы, несовместимые с блокировкой таблицы.

Для решения этой проблемы используется ***протокол преднамеренных блокировок***, являющийся расширением протокола доступа к данным.

Суть этого протокола в том, что перед тем, как наложить блокировку на объект (например, на строку таблицы), необходимо наложить специальную ***преднамеренную блокировку*** (***блокировку намерения***) на объекты, в состав которых входит блокируемый объект - на таблицу, содержащую строку, на файл, содержащий таблицу, на базу данных, содержащую файл. Тогда наличие преднамеренной блокировки таблицы будет свидетельствовать о наличии блокировки строк таблицы и для другой транзакции, пытающейся блокировать целую таблицу не нужно проверять наличие блокировок отдельных строк.

- ***Преднамеренная блокировка с возможностью взаимного доступа*** (***IS-блокировка*** - Intent Shared lock). Накладывается на некоторый составной объект T и означает намерение блокировать некоторый входящий в T объект в режиме S-блокировки. Например, при намерении читать строки из таблицы T, эта таблица должна быть заблокирована в режиме IS (до этого в таком же режиме должен быть заблокирован файл).
- ***Преднамеренная блокировка без взаимного доступа*** (***IX-блокировка*** - Intent eXclusive lock). Накладывается на некоторый составной объект T и означает намерение блокировать некоторый входящий в T объект в режиме X-блокировки. Например, при намерении удалять или модифицировать строки из таблицы T эта таблица должна быть заблокирована в режиме IX (до этого в таком же режиме должен быть заблокирован файл).

- ***Преднамеренная блокировка как с возможностью взаимного доступа, так и без него*** (***SIX-блокировка*** - Shared Intent eXclusive lock). Накладывается на некоторый составной объект T и означает разделяемую блокировку всего этого объекта с намерением впоследствии блокировать какие-либо входящие в него объекты в режиме X-блокировок. Например, если выполняется длинная операция просмотра таблицы с возможностью удаления некоторых просматриваемых строк, то можно заблокировать эту таблицу в режиме SIX (до этого захватить файл в режиме IS).

IS, IX и SIX-блокировки должны накладываться на сложные объекты базы данных (таблицы, файлы). Кроме того, на сложные объекты могут накладываться и блокировки типов S и X.

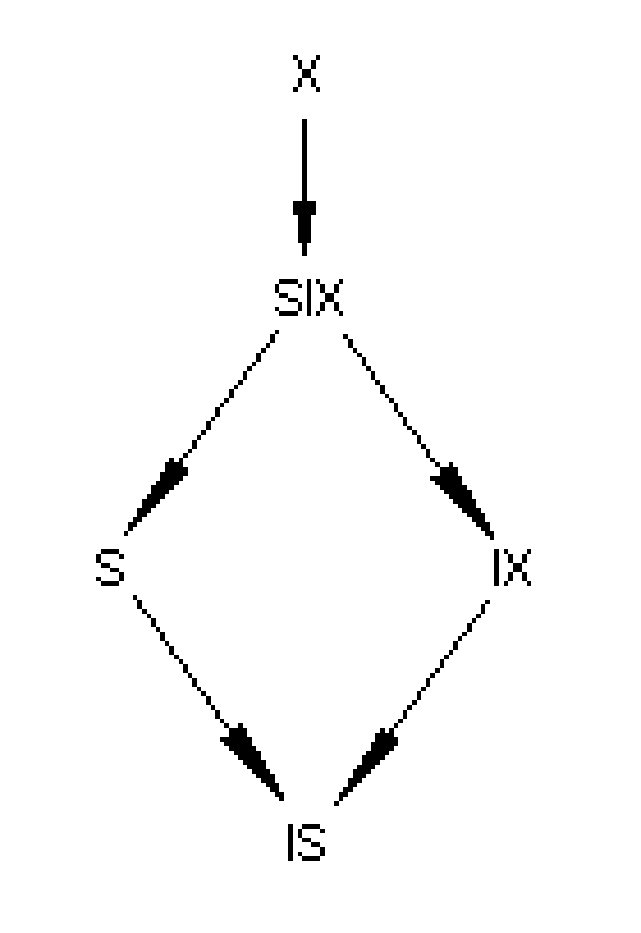
Для сложных объектов (например, для таблицы базы данных) таблица совместимости блокировок имеет следующий вид:

	Транзакция B пытается наложить на таблицу блокировку:				
Транзакция A наложила на таблицу блокировку:	IS	S	IX	SIX	X
IS	Да	Да	Да	Да	Нет
S	Да	Да	Нет	Нет	Нет
IX	Да	Нет	Да	Нет	Нет
SIX	Да	Нет	Нет	Нет	Нет
X	Нет	Нет	Нет	Нет	Нет

Более точная формулировка протокола преднамеренных блокировок для доступа к данным выглядит следующим образом:

1. При задании X-блокировки для сложного объекта *неявным* образом задается X-блокировка для всех дочерних объектов этого объекта.
2. При задании S- или SIX-блокировки для сложного объекта *неявным* образом задается S-блокировка для всех дочерних объектов этого объекта.
3. Прежде чем транзакция наложит S- или IS-блокировку на заданный объект, она должна задать IS-блокировку (или более сильную) по крайней мере для одного родительского объекта этого объекта.
4. Прежде чем транзакция наложит X-, IX- или SIX-блокировку на заданный объект, она должна задать IX-блокировку (или более сильную) для всех родительских объектов этого объекта.
5. Прежде чем для данной транзакции будет отменена блокировка для данного объекта, должны быть отменены все блокировки для дочерних объектов этого объекта.

Понятие относительной силы блокировок можно описать при помощи следующей диаграммы приоритета (сверху - более сильные блокировки, снизу - более слабые):



Протокол преднамеренных блокировок не определяет однозначно, какие блокировки должны быть наложены на родительский объект при блокировании дочернего объекта. Например, при намерении задать S-блокировку строки таблицы, на таблицу, включающую эту строку, можно наложить любую из блокировок типа IS, S, IX, SIX, X

Посмотрим, как разрешается проблема фиктивных элементов (фантомов) с использованием протокола преднамеренных блокировок для доступа к данным.

Транзакция A дважды выполняет выборку строк с одним и тем же условием. Между выборками вклинивается транзакция B, которая добавляет новую строку, удовлетворяющую условию отбора.

Транзакция B перед попыткой вставить новую строку должна наложить на таблицу IX-блокировку, или более сильную (SIX или X). Тогда транзакция A, для предотвращения возможного конфликта, должна наложить такую блокировку на таблицу, которая не позволила бы транзакции B наложить IX-блокировку. По таблице совместимости блокировок определяем, что транзакция A должна наложить на таблицу S, или SIX, или X-блокировку. (Блокировки IS недостаточно, т.к. эта блокировка *позволяет* транзакции B наложить IX-блокировку для последующей вставки строк).

Транзакция A	Время	Транзакция B
S-блокировка таблицы (с целью потом блокировать строки) - успешна	t1	---
S-блокировка строк, удовлетворяющих условию *U*. (Заблокировано n строк)	t2	---
Выборка строк, удовлетворяющих условию *U*. (Отобрано n строк)	t3	---
---	t4	IX-блокировка таблицы (с целью потом вставлять строки) - отвергается из-за конфликта с S-блокировкой, наложенной тр-ей A
---	t5	Ожидание…
S-блокировка строк, удовлетворяющих условию *U*. (Заблокировано n строк)	t6	Ожидание…
Выборка строк, удовлетворяющих условию *U*. (Отобрано n строк)	t7	Ожидание…
Фиксация транзакции - блокировки снимаются	t8	Ожидание…
---	t9	IX-блокировка таблицы (с целью потом вставлять строки) - успешна
---	t10	Вставка новой строки, удовлетворяющей условию *U*.
---	t11	Фиксация транзакции
Транзакция A дважды читает один и тот же набор строк Все правильно		

Проблема фиктивных элементов (фантомов) решается, если транзакция A использует преднамеренную S-блокировку или более сильную.

Т.к. транзакция A собирается только *читать* строки таблицы, то *минимально необходимым* условием в соответствии с протоколом преднамеренных блокировок является преднамеренная IS-блокировка таблицы. Однако этот тип блокировки не предотвращает появление фантомов. Таким образом, транзакцию A можно запускать с *разными уровнями изолированности* - предотвращая или допуская появление фантомов. Причем, оба способа запуска *соответствуют* протоколу преднамеренных блокировок для доступа к данным.

Предикатные блокировки

Другим способом блокирования является блокировка не объектов базы данных, а условий, которым могут удовлетворять объекты. Такие блокировки называются ***предикатными блокировками***.

Поскольку любая операция над БД задается некоторым условием (т.е. в ней указывается не конкретный набор объектов БД, над которыми нужно выполнить операцию, а условие, которому должны удовлетворять объекты этого набора), то удобным способом было бы S или X-блокирование именно этого условия. Однако при попытке использовать этот метод в реальной СУБД возникает трудность определения совместимости различных условий. Действительно, в языке SQL допускаются условия с подзапросами и другими сложными предикатами. Проблема совместимости сравнительно легко решается для случая простых условий, имеющих вид:

{*Имя атрибута* {= | <> | > | >= | < | <=} *Значение*}

[{**OR** | **AND**} {*Имя атрибута* {= | <> | > | >= | < | <=} *Значение*}.,..]

Проблема фиктивных элементов (фантомов) легко решается с использованием предикатных блокировок, т.к. вторая транзакция не может вставить новые строки, удовлетворяющие уже заблокированному условию.

Заметим, что блокировка всей таблицы в каком-либо режиме фактически есть предикатная блокировка, т.к. каждая таблица имеет предикат, определяющий какие строки содержатся в таблице и блокировка таблицы есть блокировка предиката этой таблицы.

Метод временных меток

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций и не требующий построения графа ожидания транзакций основан на использовании ***временных меток***.

Основная идея метода состоит в следующем: если транзакция A началась раньше транзакции B, то система обеспечивает такой режим выполнения, как если бы A была целиком выполнена до начала B.

Для этого каждой транзакции T предписывается временная метка t, соответствующая времени начала T. При выполнении операции над объектом r базы данных транзакция T помечает его своей временной меткой и типом операции (чтение или изменение).

Перед выполнением операции над объектом r транзакция B выполняет следующие действия:

- Проверяет, не закончилась ли транзакция A, пометившая этот объект. Если A закончилась, B помечает объект r своей временной меткой и выполняет операцию.
- Если транзакция A не завершилась, то B проверяет конфликтность операций. Если операции неконфликтны, при объекте r остается или проставляется временная метка с меньшим значением (более ранняя), и транзакция B выполняет свою операцию.
- Если операции B и A конфликтуют, то если t(A) > t(B) (т.е. транзакция A является более "молодой", чем B), то транзакция A откатывается и, получив новую временную метку, начинается заново. Транзакция B продолжает работу.
- Если же t(A) < t(B) (A "старше" B), то транзакция B откатывается и, получив новую временную метку, начинается заново. Транзакция A продолжает работу.

В итоге система обеспечивает такую работу, при которой при возникновении конфликтов всегда откатывается более молодая транзакция (начавшаяся позже).

Очевидным недостатком метода временных меток является то, что может откатиться более дорогая транзакция, начавшаяся позже более дешевой.

К другим недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования блокировок. Это связано с тем, что конфликтность транзакций определяется более грубо.

Механизм выделения версий данных

Использование блокировок гарантирует сериальность планов выполнения смеси транзакций за счет общего замедления работы - конфликтующие транзакции ожидают, когда транзакция, первой заблокировавшая некоторый объект, не освободит его. Без блокировок не обойтись, если все транзакции *изменяют* данные. Но если в смеси транзакций присутствуют как транзакции, изменяющие данные, так и *только читающие* данные, можно применить альтернативный механизм обеспечения сериальности, свободный от недостатков метода блокировок. Этот метод состоит в том, что транзакциям, читающим данные, предоставляется как бы "своя" версия данных, имевшаяся в момент начала читающей транзакции. При этом транзакция не накладывает блокировок на читаемые данные, и, поэтому, не блокирует другие транзакции, изменяющие данные.

Такой механизм называется ***механизм выделения версий*** и заключается в использовании журнала транзакций для генерации разных версий данных. Журнал транзакций предназначен для выполнения операции отката при неуспешном выполнении транзакции или для восстановления данных после сбоя системы.

Кратко суть метода состоит в следующем:

- Для каждой транзакции (или запроса) запоминается текущий системный номер (***SCN*** - ***System Current Number***). Чем позже начата транзакция, тем больше ее SCN.
- При записи страниц данных на диск фиксируется SCN транзакции, производящей эту запись. Этот SCN становится текущим системным номером страницы данных.
- Транзакции, только читающие данные не блокируют ничего в базе данных.
- Если транзакция А читает страницу данных, то SCN транзакции А сравнивается с SCN читаемой страницы данных.
- Если SCN страницы данных меньше или равен SCN транзакции А, то транзакция А читает эту страницу.
- Если SCN страницы данных больше SCN транзакции А, то это означает, что некоторая транзакция В, начавшаяся позже транзакции А, успела изменить или сейчас изменяет данные страницы. В этом случае транзакция А просматривает журнал транзакция назад в поиске первой записи об изменении нужной страницы данных с SCN меньшим, чем SCN транзакции А. Найдя такую запись, транзакция А использует старый вариант данных страницы.

Транзакция A	Время	Транзакция B
Проверка SCN счета P1- SCN транзакции больше SCN счета. Чтение счета P1=100 без наложения блокировки и суммирование. SUM=100		---
---		X-блокировка счета P3- успешна
---		Снятие денег со счета *P3.* *P3:100 ->50*
---		X-блокировка счета - успешна
---		Помещение денег на счет *P1.* *P1:100 ->150*
---		Фиксация транзакции (Снятие блокировок)
Проверка SCN счета P2- SCN транзакции больше SCN счета. Чтение счета P2 без наложения блокировки и суммирование. SUM=200		---
Проверка SCN счета P3 - SCN транзакции МЕНЬШЕ SCN счета. Чтение старого варианта счета P3 и суммирование. SUM=300		---
Фиксация транзакции		---
Сумма на счетах посчитана правильно.		

Теорема Есварана о сериализуемости

Концепция способности к упорядочению была впервые предложена Есвараном. Им был предложен ***протокол двухфазной блокировки***:

1. Перед выполнение каких-либо операций с некоторым объектом, транзакция должна заблокировать этот объект.
2. После снятия блокировки, транзакция не должна накладывать никаких других блокировок.

Транзакции, используемые в этом протоколе, не различаются по типам и считаются монопольными. Описанные выше протоколы доступа к данным с использованием S- и X-блокировок и протокол преднамеренных блокировок являются модификациями протокола двухфазной блокировки для случая, когда блокировки имеют различные типы.

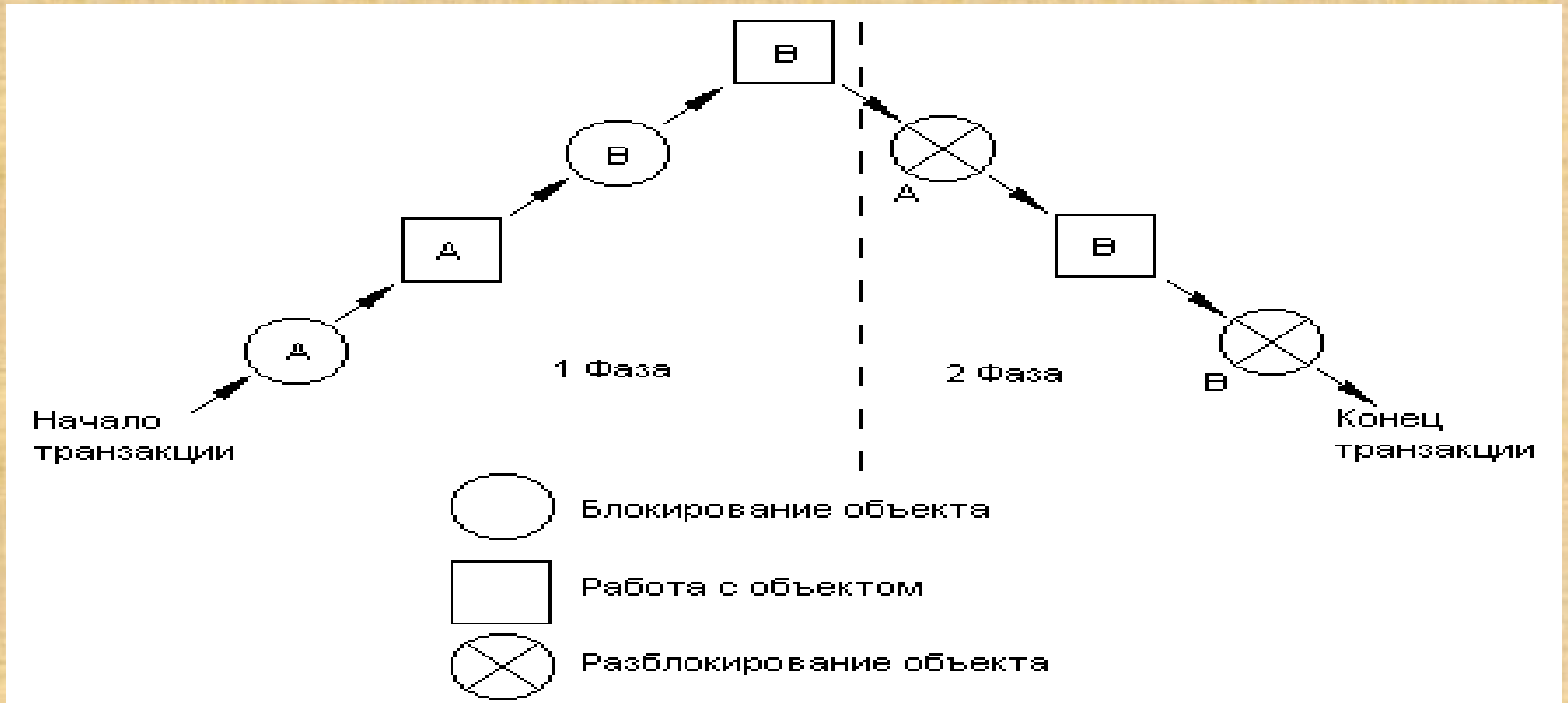
Есвараном сформулирована следующая теорема:

Теорема Есварана. Если все транзакции в смеси подчиняются протоколу двухфазной блокировки, то для всех чередующихся графиков запуска существует возможность упорядочения.

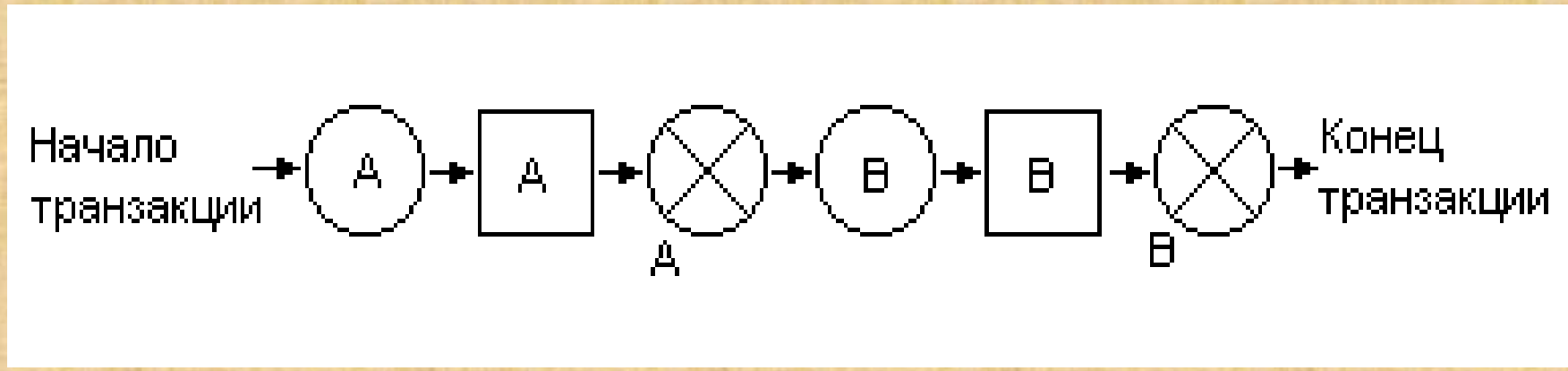
Протокол называется двухфазным, потому что он характеризуется двумя фазами:

1 фаза - нарастание блокировок. Во время этой фазы накладываются блокировки, и производится работа с заблокированными объектами.

2 фаза - снятие блокировок. Во время этой фазы блокировки только снимаются. Работа с ранее заблокированными данными может продолжаться.



На следующем рисунке показан пример транзакции, не подчиняющийся протоколу двухфазной блокировки:



На практике, как правило, вторая фаза сводится к одной операции завершения транзакции (или отката транзакции) с одновременным снятием всех блокировок.

Если некоторая транзакция А не подчиняется протоколу двухфазной блокировки (и, следовательно, состоит не менее чем из двух операция блокирования и разблокирования), то всегда можно построить другую транзакцию В, которая при чередующемся выполнении вместе с А приводит к графику, не подлежащему упорядочению и неверному.

Реализация изолированности транзакций средствами SQL

Уровни изоляции

Стандарт SQL *не предусматривает* понятие блокировок для реализации сериализуемости смеси транзакций. Вместо этого вводится понятие уровней изоляции. Этот подход обеспечивает необходимые требования к изолированности транзакций, оставляя возможность производителям различных СУБД реализовывать эти требования своими способами (в частности, с использованием блокировок или выделением версий данных).

Стандарт SQL предусматривает 4 уровня изоляции:

- ***READ UNCOMMITTED*** - *уровень незавершенного считывания*.
- ***READ COMMITTED*** - *уровень завершенного считывания*.
- ***REPEATABLE READ*** - *уровень повторяемого считывания*.
- ***SERIALIZABLE*** - *уровень способности к упорядочению*.

Если все транзакции выполняются на уровне способности к упорядочению (принятом по умолчанию), то чередующееся выполнение любого множества параллельных транзакций может быть упорядочено. Если некоторые транзакции выполняются на более низких уровнях, то имеется множество способов нарушить способность к упорядочению. В стандарте SQL выделены три особых случая нарушения способности к упорядочению, фактически именно те, которые были описаны выше как проблемы параллелизма:

- ***Неаккуратное считывание*** ("Грязное" чтение, незафиксированная зависимость).
- ***Неповторяемое считывание*** (Частный случай несовместного анализа).
- ***Фантомы*** (Фиктивные элементы - частный случай несовместного анализа).

Потеря результатов обновления стандартом SQL не допускается, т.е. на самом низком уровне изолированности транзакции должны работать так, чтобы не допустить потери результатов обновления.

Различные уровни изоляции определяются по возможности или исключению этих особых случаев нарушения способности к упорядочению. Эти определения описываются следующей таблицей:

Уровень изоляции	Неаккуратное считывание	Неповторяемое считывание	Фантомы
READ UNCOMMITTED	Да	Да	Да
READ COMMITTED	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Да
SERIALIZABLE	Нет	Нет	Нет

Синтаксис операторов SQL, определяющих уровни изоляции

Уровень изоляции транзакции задается следующим оператором:

SET TRANSACTION {ISOLATION LEVEL

{READ UNCOMMITTED

| READ COMMITTED

| REPEATABLE READ

| SERIALIZABLE}

| {READ ONLY | READ WRITE}}.,..

Этот оператор определяет режим выполнения *следующей* транзакции, т.е. этот оператор не влияет на изменение режима той транзакции, в которой он подается. Обычно, выполнение оператора SET TRANSACTION выделяется как отдельная транзакция:

… (предыдущая транзакция выполняется со своим уровнем изоляции)

COMMIT;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

COMMIT;

… (следующая транзакция выполняется с уровнем изоляции REPEATABLE READ)

Если задано предложение ISOLATION LEVEL, то за ним должно следовать один из параметров, определяющих уровень изоляции.

Кроме того, можно задать признаки READ ONLY или READ WRITE. Если указан признак READ ONLY, то предполагается, что транзакция будет только читать данные. При попытке записи для такой транзакции будет сгенерирована ошибка. Признак READ ONLY введен для того, чтобы дать производителям СУБД возможность уменьшать количество блокировок путем использования других методов сериализации (например, метод выделения версий).

Оператор SET TRANSACTION должен удовлетворять следующим условиям:

Если предложение ISOLATION LEVEL отсутствует, то по умолчанию принимается уровень SERIALIZABLE.

Если задан признак READ WRITE, то параметр ISOLATION LEVEL не может принимать значение READ UNCOMMITTED.

Если параметр ISOLATION LEVEL определен как READ UNCOMMITTED, то транзакция становится по умолчанию READ ONLY. В противном случае по умолчанию транзакция считается как READ WRITE.

Транзакции и восстановление данных

Главное требование долговечности данных транзакций состоит в том, что данные зафиксированных транзакций должны сохраняться в системе, даже если в следующий момент произойдет сбой системы. Казалось бы, самый простой способ обеспечить такую гарантию - это во время каждой операции сразу записывать все изменения на дисковые носители. Такой способ не является удовлетворительным, т.к. имеется существенное различие в скорости работы с оперативной и с внешней памятью. Единственный способ достичь приемлемой скорости работы состоит в буферизации страниц базы данных в оперативной памяти. Это означает, что данные попадают во внешнюю долговременную память не сразу после внесения изменений, а через некоторое (достаточно большое) время. Тем не менее, что-что во внешней памяти должно оставаться, т.к. иначе неоткуда получить информацию для восстановления.

Требование атомарности транзакций утверждает, что не законченные или откатившиеся транзакции не должны оставлять следов в базе данных. Это означает, что данные должны храниться в базе данных с избыточностью, позволяющей иметь информацию, по которой восстанавливается состояние базы данных на момент начала неудачной транзакции. Такую избыточность обычно обеспечивает журнал транзакций. Журнал транзакций содержит детали всех операций модификации данных в базе данных, в частности, старое и новое значение модифицированного объекта, системный номер транзакции, модифицировавшей объект и другая информация.

Виды восстановления данных

Восстановление базы данных может производиться в следующих случаях:

Индивидуальный откат транзакции. Откат индивидуальной транзакции может быть инициирован либо самой транзакцией путем подачи команды ROLLBACK, либо системой. СУБД может инициировать откат транзакции в случае возникновения какой-либо ошибки в работе транзакции (например, деление на нуль) или если эта транзакция выбрана в качестве жертвы при разрешении тупика.

Мягкий сбой системы (***аварийный отказ программного обеспечения***). Мягкий сбой характеризуется утратой оперативной памяти системы. При этом поражаются все выполняющиеся в момент сбоя транзакции, теряется содержимое всех буферов БД. Данные, хранящиеся на диске, остаются неповрежденными. Мягкий сбой может произойти, например, в результате аварийного отключения питания или в результате неустранимого сбоя процессора.

Жесткий сбой системы (***аварийный отказ аппаратуры***). Жесткий сбой характеризуется повреждением внешних носителей памяти. Жесткий сбой может произойти, например, в результате поломки головок дисковых накопителей.

Во всех трех случаях основой восстановления является избыточность данных, обеспечиваемая журналом транзакций.

Как и страницы базы данных, данные из журнала транзакций не записываются сразу на диск, а предварительно буферизируются в оперативной памяти. Таким образом, система поддерживает два вида буферов - буферы страниц базы данных и буферы журнала транзакций.

Страницы базы данных, содержимое которых в буфере (в оперативной памяти) отличается от содержимого на диске, называются ***"грязными" (dirty) страницами***. Система постоянно поддерживает список "грязных" страниц - ***dirty-список***. Запись "грязных" страниц из буфера на диск называется ***выталкиванием страниц во внешнюю память***.

Очевидно, необходимо предусмотреть такие правила выталкивания буферов базы данных и буферов журнала транзакций, которые обеспечивали бы два требования:

1. *Максимальную скорость выполнения транзакций*. Для этого необходимо выталкивать страницы как можно реже. В идеале, если оперативная память была бы бесконечной, и сбои никогда бы не происходили, наилучшим выходом была бы загрузка *всей* базы данных в оперативную память, работа с данными *только в оперативной памяти*, и запись измененных страниц на диск *только* в момент завершения работы всей системы.

2. Гарантию, что при возникновении сбоя (любого типа), данные завершенных транзакций можно было бы восстановить, а данные незавершенных транзакций бесследно удалить, т.е. обеспечение восстановления последнего согласованного состояния базы данных. Для этого что-то выталкивать на диск все-таки необходимо, даже если мы обладали бы бесконечной оперативной памятью.

Таким образом, имеется две причины для периодического выталкивания страниц во внешнюю память - недостаток оперативной памяти и возможность сбоев.

Основным принципом согласованной политики выталкивания буфера журнала и буферов страниц базы данных является то, что запись об изменении объекта базы данных должна попадать во внешнюю память журнала раньше, чем измененный объект оказывается во внешней памяти базы данных. Соответствующий протокол журнализации (и управления буферизацией) называется ***Write Ahead Log*** (***WAL***) - "***пиши сначала в журнал***", и состоит в том, что если требуется вытолкнуть во внешнюю память измененный объект базы данных, то перед этим нужно гарантировать выталкивание во внешнюю память журнала записи о его изменении. Это означает, что если во внешней памяти базы данных содержится объект, к которому применена некоторая команда модификации, то во внешней памяти журнала транзакций содержится запись об этой операции. Обратное неверно - если во внешней памяти журнала содержится запись о некотором изменении объекта, то во внешней памяти базы данных может и не быть самого измененного объекта.

Дополнительное условие на выталкивание буферов - каждая успешно завершившаяся транзакция должна быть реально зафиксирована во внешней памяти. Какой бы сбой не произошел, система должна быть в состоянии восстановить состояние базы данных, содержащее результаты всех *зафиксированных* к моменту сбоя транзакций.

3-м условием выталкивания буферов является ограниченность объемов буферов БД и журнала транзакций. Периодически или при наступлении определенного события (например, количество страниц в dirty-списке превысило определенный порог, или количество свободных страниц в буфере уменьшилось и достигло критического значения) система принимает так называемую ***контрольную точку***. Принятие контрольной точки включает выталкивание во внешнюю память содержимого буферов БД и специальную физическую ***запись контрольной точки***, которая представляет собой список всех осуществляемых в данный момент транзакций.

Оказывается, что *минимальным требованием*, гарантирующим возможность восстановления последнего согласованного состояния БД, является *выталкивание при фиксации транзакции во внешнюю память журнала всех записей об изменении базы данных этой транзакцией*. При этом последней записью в журнал, производимой от имени данной транзакции, является специальная запись о конце этой транзакции.

Индивидуальный откат транзакции

Для того чтобы можно было выполнить по журналу транзакций индивидуальный откат транзакции, все записи в журнале от данной транзакции связываются в обратный список. Началом списка для не закончившихся транзакций является запись о последнем изменении базы данных, произведенном данной транзакцией. Для закончившихся транзакций (индивидуальные откаты которых уже невозможны) началом списка является запись о конце транзакции, которая обязательно вытолкнута во внешнюю память журнала. Концом списка всегда служит первая запись об изменении базы данных, произведенном данной транзакцией. В каждой записи имеется уникальный системный номер транзакции, чтобы можно было восстановить прямой список записей об изменениях базы данных данной транзакцией.

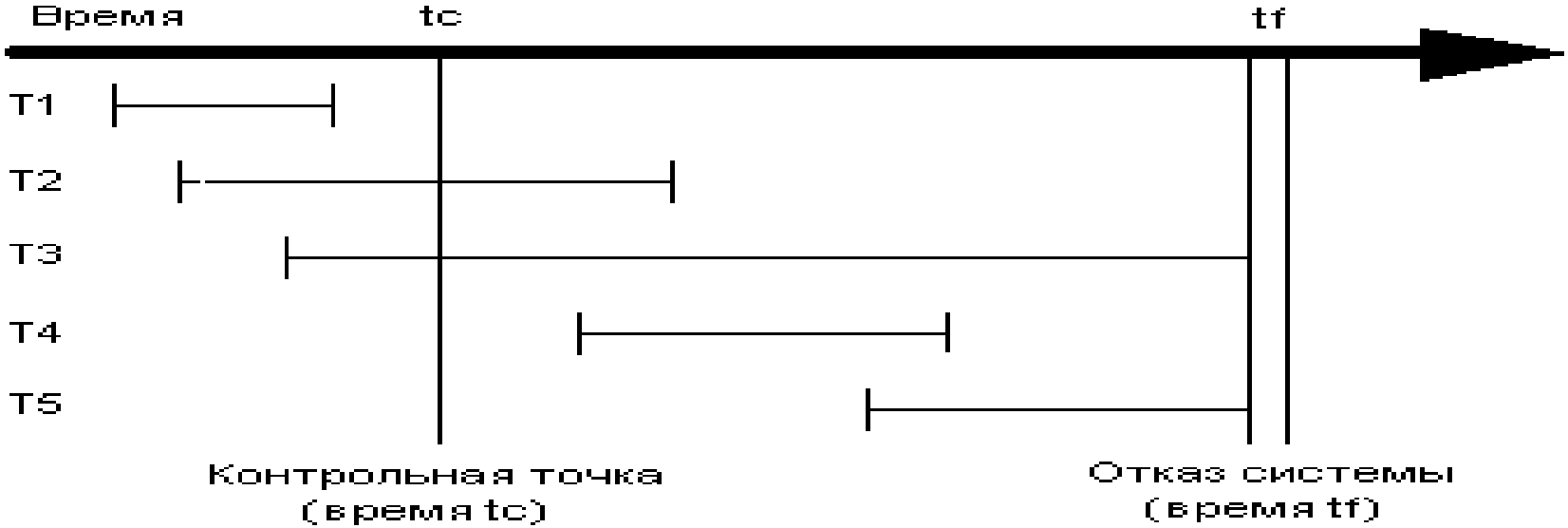
Индивидуальный откат транзакции выполняется следующим образом:

- Просматривается список записей, сделанных данной транзакцией в журнале транзакций (от последнего изменения к первому изменению).
- Выбирается очередная запись из списка данной транзакции.
- Выполняется противоположная по смыслу операция: вместо операции INSERT выполняется соответствующая операция DELETE, вместо операции DELETE выполняется INSERT, и вместо прямой операции UPDATE обратная операция UPDATE, восстанавливающая предыдущее состояние объекта базы данных.
- Любая из этих обратных операций также журнализируются. Это необходимо делать, потому что во время выполнения индивидуального отката может произойти мягкий сбой, при восстановлении после которого потребуется откатить такую транзакцию, для которой не полностью выполнен индивидуальный откат.
- При успешном завершении отката в журнал заносится запись о конце транзакции.

Восстановление после мягкого сбоя

Несмотря на протокол WAL, после мягкого сбоя не все физические страницы базы данных содержат измененные данные, т.к. не все "грязные" страницы базы данных были вытолкнуты во внешнюю память.

Последний момент, когда гарантированно были вытолкнуты "грязные" страницы - это момент принятия последней контрольной точки. Имеется 5 вариантов состояния транзакций по отношению к моменту последней контрольной точки и к моменту сбоя:



Последняя контрольная точка принималась в момент tc. Мягкий сбой системы произошел в момент tf. Транзакции T1-T5 характеризуются следующими свойствами:

- ***T1*** - транзакция успешно завершена до принятия контрольной точки. Все данные этой транзакции сохранены в долговременной памяти - как записи журнала, так и страницы данных, измененные этой транзакцией. Для транзакции T1 никаких операций по восстановлению не требуется.
- ***T2*** - транзакция начата до принятия контрольной точки и успешно завершена после контрольной точки, но до наступления сбоя. Записи журнала транзакций, относящиеся к этой транзакции вытолкнуты во внешнюю память. Страницы данных, измененные этой транзакцией, только частично вытолкнуты во внешнюю память. Для данной транзакции необходимо повторить заново те операции, которые были выполнены после принятия контрольной точки.

- ***T3*** - транзакция начата до принятия контрольной точки и не завершена в результате сбоя. Такую транзакцию необходимо откатить. Проблема, однако, в том, что часть страниц данных, измененных этой транзакцией, уже содержится во внешней памяти - это те страницы, которые были обновлены до принятия контрольной точки. Следов изменений, внесенных после контрольной точки в БД нет. Записи журнала транзакций, сделанные до принятия контрольной точки, вытолкнуты во внешнюю память, те записи журнала, которые были сделаны после контрольной точки, отсутствуют во внешней памяти журнала.
- ***T4*** - транзакция начата после принятия контрольной точки и успешно завершена до сбоя системы. Записи журнала транзакций, относящиеся к этой транзакции вытолкнуты во внешнюю память журнала. Изменения в БД, внесенные этой транзакцией, полностью отсутствуют во внешней памяти БД. Такую транзакцию необходимо повторить целиком.
- ***T5*** - транзакция начата после принятия контрольной точки и не завершена в результате сбоя. Никаких следов этой транзакции нет ни во внешней памяти журнала транзакций, ни во внешней памяти БД. Для такой транзакции никаких действий предпринимать не нужно, ее как бы и не было вовсе.

Восстановление системы после мягкого сбоя осуществляется как *часть процедуры перезагрузки* системы. При перезагрузке системы транзакции T2 и T4 необходимо частично или полностью повторить, транзакцию T3 - частично откатить, для транзакций T1 и T5 никаких действий предпринимать не нужно. При перезагрузке система выполняет следующие действия:

- Создается два списка транзакций UNDO (отменить) и REDO (повторить). В список UNDO заносятся все транзакции из последней записи контрольной точки (т.е. все транзакции, выполнявшиеся в момент принятия контрольной точки). Список REDO остается пустым. В нашем случае будет: UNDO = {T2, T3}, REDO = { }.
- Начиная с записи контрольной точки просматривается вперед журнал транзакций.
- Если в журнале транзакций обнаруживается запись о начале транзакции, то эта транзакция добавляется в список UNDO. В нашем случае будет: UNDO = {T2, T3, T4}, REDO = { }. Заметим, что следов транзакции T5 в журнале транзакций нет.

- Если в файле регистрации обнаруживается запись COMMIT об окончании транзакции, то эта транзакция добавляется в список REDO. В нашем случае будет: UNDO = {T2, T3, T4}, REDO = {T2, T4}. Заметим, что записи о конце этих транзакций *имеются* во внешней памяти журнала транзакций в соответствии с минимальным требованием выталкивания записей журнала при фиксации транзакции.
- Когда достигается конец журнала транзакций, оба списка анализируются. При этом из списка UNDO удаляются те транзакции, которые попали в список REDO. В нашем случае будет: UNDO = {T3}, REDO = {T2, T4}.
- После этого система просматривает журнал транзакций назад, начиная с момента контрольной точки и откатывая все транзакции из списка UNDO. В нашем случае будут откатываться те операции транзакции T3, которые были выполнены до принятия контрольной точки.

Окончательно, система просматривает журнал транзакций вперед, начиная с момента контрольной точки, и повторно выполняет все операции транзакций из списка REDO. В нашем случае, система выполнит повторно все операции транзакции Т4 и те операции транзакции Т2, которые были выполнены после принятия контрольной точки.

Восстановление после жесткого сбоя

При жестком сбое база данных на диске нарушается физически. Основой восстановления в этом случае является журнал транзакций и ***архивная копия базы данных***. Архивная копия базы данных должна создаваться периодически, а именно с учетом скорости наполнения журнала транзакций.

Восстановление начинается с обратного копирования базы данных из архивной копии. Затем выполняется просмотр журнала транзакций для выявления всех транзакций, которые закончились *успешно* до наступления сбоя. (Транзакции, закончившиеся откатом до наступления сбоя, можно не рассматривать). После этого по журналу транзакций в прямом направлении повторяются все успешно законченные транзакции. При этом нет необходимости отката транзакций, прерванных в результате сбоя, т.к. изменения, внесенными этими транзакциями, отсутствуют после восстановления базы данных из резервной копии.

Наиболее плохим случаем является ситуация, когда разрушены физически и база данных, и журнал транзакций. В этом случае единственное, что можно сделать - это восстановить состояние базы данных на момент последнего резервного копирования. Для того чтобы не допустить возникновение такой ситуации, базу данных и журнал транзакций обычно располагают на *физически* разных дисках, управляемых физически разными контроллерами.

Восстановление данных и стандарт SQL

Стандарт языка SQL не содержит требований к восстановимости данных, оставляя эти вопросы на усмотрение разработчиков СУБД.

Оптимизация запросов для SQL Server

Планы запросов

Когда сервер SQL Server выполняет запрос, сначала требуется определить наилучший способ выполнения. Для этого нужно рассчитать, как и в каком порядке обращаться к данным и соединять их, как и когда выполнять вычисления и агрегации и т. д.

За это отвечает подсистема, которая называется Query Optimizer (Оптимизатор запроса). Оптимизатор запроса использует статистические данные о распределении данных, метаданные, относящиеся к объектам в базе данных, информацию индекса и другие факторы для вычисления нескольких возможных планов выполнения запроса. Для каждого из этих планов Оптимизатор запроса предполагает его стоимость на основе статистики по этим данным и выбирает план с минимальными затратами ресурсов на выполнение. Конечно, SQL Server не вычисляет всех возможных планов для каждого запроса, поскольку для некоторых запросов сами эти вычисления могут отнять больше времени, чем выполнение наименее эффективного из всех планов.

Следовательно, SQL Server использует сложные алгоритмы, чтобы найти план выполнения с разумной стоимостью, близкой к минимально возможной. После того, как план выполнения сгенерирован, он хранится в буферном кэше (на что SQL Server выделяет большую часть своей виртуальной памяти). Затем план выполняется тем способом, который Оптимизатор запроса сообщает ядру базы данных (компоненту database engine).

- Сможет ли Query Optimazer (Оптимизатор запросов) сгенерировать эффективный план для конкретного запроса, зависит от следующих аспектов:
- **Индексы**. Подобно оглавлению в книге, индекс базы данных позволяет быстро найти определенные строки в таблице. В таблице может быть не один индекс. Благодаря наличию в таблице индексов, Оптимизатор запросов SQL Server может оптимизировать доступ к данным, выбрав для использования подходящий индекс. Если индексы отсутствуют, у Оптимизатора запросов остается только один вариант, который заключается в сканировании всех данных, имеющихся в таблице, в поиске нужных строк.

- **Статистика распределения данных:** SQL Server хранит статистику о распределении данных. Если эта статистика отсутствует или устарела, Оптимизатор запросов не сможет вычислить эффективный план выполнения запроса. В большинстве случаев, статистические данные генерируются и обновляются автоматически. Далее в этой лекции рассказывается о том, как генерируются статистические данные и как можно управлять статистикой.

Как видите, генерирование плана выполнения запросов - это функция, немаловажная для производительности SQL Server, поскольку эффективность плана выполнения запроса определяет, будет ли время его выполнения измеряться в миллисекундах, секундах или даже минутах. Планы выполнения запросов, которые показали низкую скорость выполнения, можно проанализировать, чтобы определить, имеется ли индекс, устарели ли данные статистики или просто SQL Server выбрал не самый эффективный план (такое случается не очень часто).

Индексы

Индекс является структурой на диске, которая связана с таблицей или представлением и ускоряет получение строк из таблицы или представления. Индекс содержит ключи, построенные из одного или нескольких столбцов в таблице или представлении. Эти ключи хранятся в виде структуры сбалансированного дерева, которая поддерживает быстрый поиск строк по их ключевым значениям.

Кластеризованный

Кластеризованные индексы сортируют и хранят строки данных в таблицах или представлениях на основе их ключевых значений. Этими значениями являются столбцы, включенные в определение индекса. Существует только один кластеризованный индекс для каждой таблицы, потому что строки данных могут быть отсортированы только в единственном порядке.

Строки данных в таблице хранятся в порядке сортировки только в том случае, если таблица содержит кластеризованный индекс. Если у таблицы есть кластеризованный индекс, то таблица называется кластеризованной. Если у таблицы нет кластеризованного индекса, то строки данных хранятся в неупорядоченной структуре, которая называется кучей.

Некластеризованный

Некластеризованные индексы имеют структуру, отдельную от строк данных. В некластеризованном индексе содержатся значения ключа некластеризованного индекса, и каждая запись значения ключа содержит указатель на строку данных, содержащую значение ключа.

Указатель из строки индекса в некластеризованном индексе, который указывает на строку данных, называется указателем строки. Структура указателя строки зависит от того, хранятся ли страницы данных в куче или в кластеризованной таблице. Для кучи указатель строки является указателем на строку. Для кластеризованной таблицы указатель строки данных является ключом кластеризованного индекса.

Можно добавить неключевые столбцы на конечный уровень некластеризованного индекса и обойти существующее ограничение на ключи индексов (900 байт и 16 ключевых столбцов) и выполнять полностью индексированные запросы.

Тип индекса	Описание
Кластеризованный	Кластеризованный индекс сортирует и хранит строки данных таблицы или представления в порядке, определяемом ключом кластеризованного индекса. Кластеризованный индекс реализуется в виде сбалансированного дерева, которое поддерживает быстрое получение строк по значениям ключа кластеризованного индекса.
Некластеризованный	Некластеризованный индекс можно определить в таблице или представлении вместе с кластеризованным индексом или в куче. Каждая строка некластеризованного индекса содержит некластеризованное ключевое значение и указатель на строку. Этот указатель определяет строку данных кластеризованного индекса или кучи, содержащую ключевое значение. Строки в индексе хранятся в порядке, определяемом значениями ключа индекса, но до создания кластеризованного индекса в таблице нет никакой гарантии того, что строки данных будут расположены в каком-либо определенном порядке.

Уникальный	Уникальный индекс обеспечивает отсутствие повторяющихся значений ключа индекса, что, в свою очередь, приводит к тому, что каждая строка в таблице или представлении является в каком-то смысле уникальной. Как кластеризованные, так и некластеризованные индексы могут быть уникальными.
Индекс с включенными столбцами	Некластеризованный индекс, дополнительно содержащий кроме ключевых столбцов еще и неключевые.
Индексированные представления	Индекс представления материализуется, при этом представление и результирующий набор постоянно хранятся в уникальном кластеризованном индексе тем же образом, что и таблица с кластеризованным индексом. Некластеризованные индексы для представления можно добавить после создания кластеризованного индекса.

Полнотекстовый	Специальный тип функционального индекса, основанный на маркере, построенный и поддерживаемый средством полнотекстового поиска для SQL Server. Он обеспечивает эффективную поддержку сложных операций поиска слов в символьных строковых данных.
Пространственный	Пространственный индекс обеспечивает возможность более эффективного использования конкретных операций на пространственных объектах (*пространственных данных*) в столбце типа данных geometry. Пространственные индексы снижают количество объектов, к которым должны применяться пространственные операции, требующие больших затрат.
Фильтруемый	Оптимизированный некластеризованный индекс, в особенности подходящий для покрытия запросов из хорошо определенного набора данных. Использует предикат фильтра для индексирования части строк в таблице. Хорошо спроектированный фильтруемый индекс позволяет повысить производительность запросов, снизить затраты на обслуживание и хранение индексов по сравнению с полнотабличными индексами.

XML	Вырезанное материализованное представление больших двоичных XML-объектов (BLOB) в столбце с типом данных xml.

Основы проектирования индексов

Использование индекса не всегда означает высокую производительность, а высокая производительность не всегда означает эффективное использование индекса. Если бы использование индекса всегда способствовало производительности, то работа оптимизатора запросов была бы очень простой. На самом деле, неверный выбор индекса может привести к неоптимальной производительности. Следовательно, задача оптимизатора запросов состоит в том, чтобы выбрать индекс или комбинацию индексов, если это улучшит производительность, и избежать индексированного поиска, если это ее понизит.

Рекомендуемая стратегия проектирования индексов включает в себя следующие задачи:

- Прежде всего следует понять характеристики самой базы данных. Например, будет ли это база данных оперативной обработки транзакций (OLTP) с часто изменяющимися данными, или система поддержки решений (DDS), или хранилище данных (OLAP), предназначенное в основном для чтения?
- Определите наиболее часто используемые запросы. Например, если известно, что часто используется запрос на соединение двух и более таблиц, это поможет определить наилучший тип индексов.
- Выясните характеристики столбцов, используемых в запросах. Например, идеальным будет индекс для целочисленных столбцов, которые к тому же имеют уникальные или обязательно определяемые значения. Фильтруемый индекс подходит для столбцов, имеющих точно определенные подмножества данных.
- Определите, какие параметры индексов могут повысить производительность при создании индекса или при его поддержке. Например, при создании кластеризованного индекса для существующей большой таблицы очень выгодно будет использовать параметр ONLINE. Параметр ONLINE позволяет продолжать параллельную обработку базовых данных во время создания или повторного построения индекса.

- Определите оптимальное расположение для хранения индекса. Некластеризованный индекс может храниться в той же файловой группе, что и базовая таблица, или в другой группе. Правильный выбор расположения для хранения индексов может повысить производительность запросов за счет повышения скорости дискового ввода-вывода. Например, если некластеризованный индекс хранится в файловой группе не на том диске, на котором расположены файловые группы таблицы, то производительность может повыситься, поскольку это позволяет одновременно обращаться к нескольким дискам.
- Кластеризованные и некластеризованные индексы могут использовать схему секционирования, которая охватывает несколько файловых групп. Секционирование делает большие таблицы и индексы более управляемыми, позволяет быстро и эффективно получать доступ к наборам данных и управлять ими, при этом сохраняя целостность всей коллекции. При выборе секционирования определите, требуется ли выравнивание индекса, то есть должен ли индекс быть секционирован точно так же, как и таблицы, или он может быть секционирован иным образом.

Опытный администратор базы данных может спроектировать хороший набор индексов, но эта задача очень сложна, требует много времени и сопряжена с ошибками даже для рабочих нагрузок и баз данных средней сложности. В разработке оптимальных индексов может помочь понимание характеристик базы данных, запросов и столбцов данных.

Соображения, связанные с базами данных

При проектировании индекса следует учитывать следующие рекомендации:

- Большое количество индексов в таблице снижает производительность инструкций INSERT, UPDATE, DELETE и MERGE, потому что при изменении данных в таблице все индексы должны быть изменены соответствующим образом.
  - Избегайте использования чрезмерного количества индексов для интенсивно обновляемых таблиц и следите, чтобы индексы были узкими, то есть содержали как можно меньше столбцов.
  - Используйте большое количество индексов, чтобы улучшить производительность запросов для таблиц с низкими требованиями к обновлениям, но большими объемами данных.

- Индексирование маленьких таблиц может оказаться не лучшим выбором, так как поиск данных в индексе может потребовать у оптимизатора запросов больше времени, чем простой просмотр таблицы.
- Индексы представлений могут дать значительное улучшение производительности, если представление содержит агрегаты, объединения таблиц или сочетание того и другого. *Соображения, связанные с запросами*
- При проектировании индекса следует принимать во внимание следующие рекомендации, связанные с обработкой запросов.
- Следует создавать некластеризованные индексы для всех столбцов, которые часто используются в предикатах и условиях соединения в запросах.
- Покрывающие индексы могут повысить производительность запросов, так как данные, необходимые для удовлетворения требований запроса, присутствуют в самом индексе. Т.о., для получения запрашиваемых данных требуются только страницы индекса, а не страницы данных таблицы или кластеризованного индекса; следовательно, уменьшается общий объем операций дискового ввода-вывода.

Соображения, связанные со столбцами

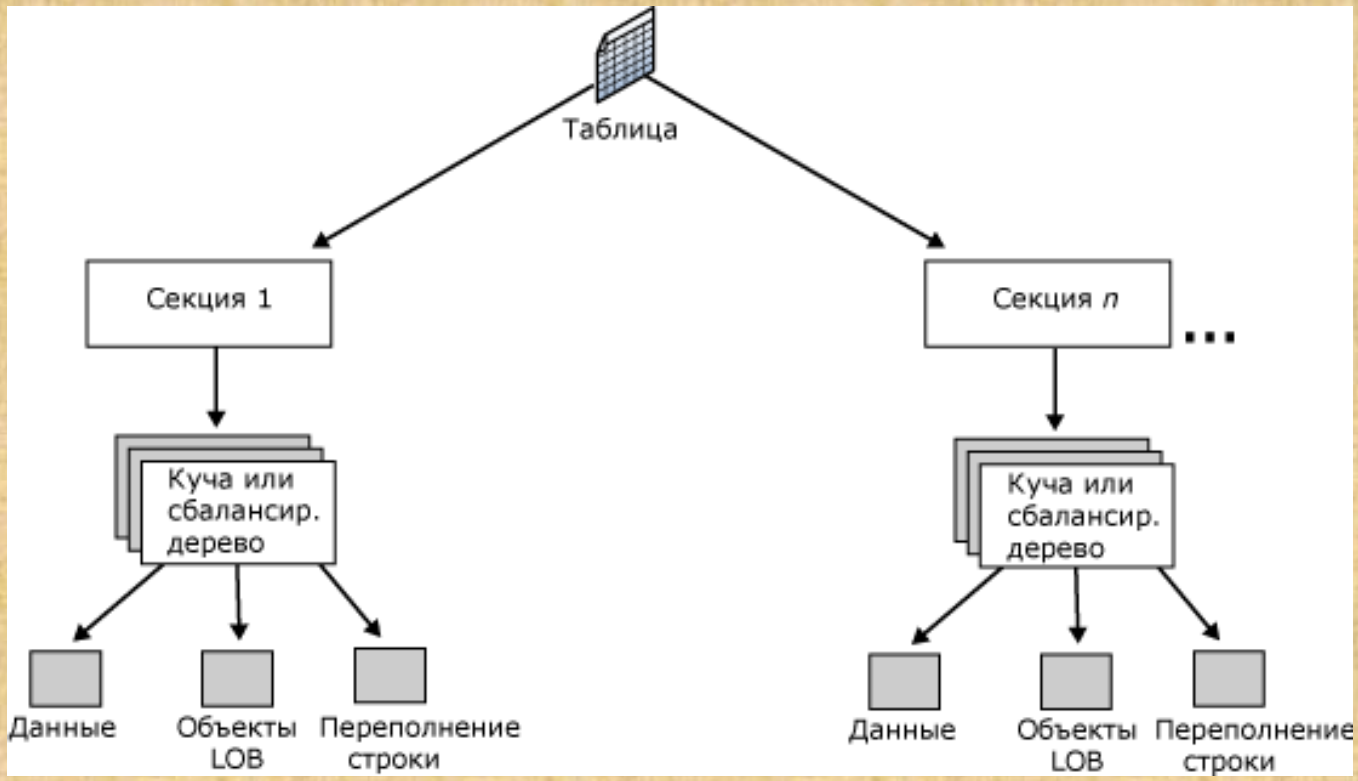
При проектировании индекса, следует принимать во внимание следующие рекомендации, относящиеся к столбцам.

- Нужно следить, чтобы длина ключа для кластеризованных индексов была небольшой.
- Проверьте распределение данных в столбце. Часто длительное выполнение запроса обусловлено индексированием столбца, в котором мало уникальных значений, или присоединением такого столбца. Например: физический телефонный справочник, отсортированный в алфавитном порядке по фамилии, не сможет быстро найти человека, если всех жителей города зовут Смит или Джонс.
- Следует учитывать порядок столбцов, если индекс будет включать их несколько. Столбец, использованный в предложении WHERE в условии поиска =, >, < или BETWEEN значений или участвующий в соединении, должен стоять первым.

Например, если индекс определен как LastName, FirstName, индекс будет полезным, если критерий поиска — WHERE LastName = 'Smith' или WHERE LastName = Smith AND FirstName LIKE 'J%'. Однако оптимизатор запросов не станет использовать этот индекс для запроса только по критерию FirstName (WHERE FirstName = 'Jane').

Организация таблиц и индексов

Таблицы и индексы хранятся в виде коллекции страниц размером 8 КБ. Таблица содержится в одной или нескольких секциях, а каждая секция содержит строки данных либо в куче, либо в структуре кластеризованного индекса. Страницы кучи или кластеризованного индекса объединяются в одну или несколько единиц распределения в зависимости от типов столбцов в строках данных.



Страницы таблиц и индексов содержатся в одной или нескольких секциях. Секция — это пользовательская единица организации данных. По умолчанию таблица или индекс имеет единственную секцию, которая содержит все страницы таблицы или индекса. Секция располагается в одной файловой группе. Таблица или индекс, имеющие одну секцию, эквивалентны организационной структуре таблиц и индексов предыдущих версий SQL Server.

Если таблица или индекс используют несколько секций, данные секционируются горизонтально, так что группы строк сопоставляются отдельным секциям, основываясь на указанном столбце. Секции могут храниться в одной или нескольких файловых группах в базе данных. Таблица или индекс рассматриваются как единая логическая сущность при выполнении над данными запросов или обновлений.

Таблицы SQL Server используют один из двух методов организации страниц данных внутри секции.

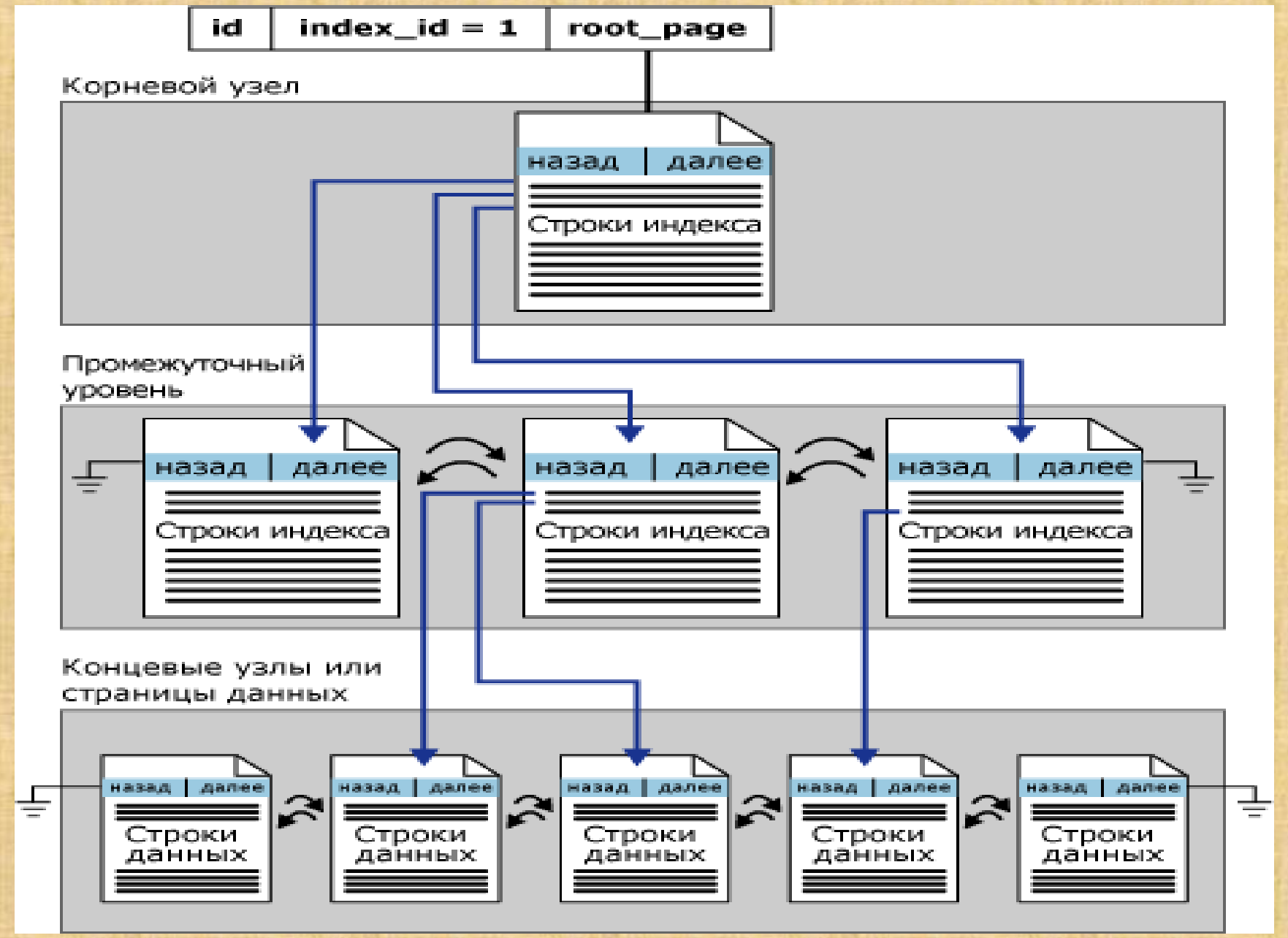
- Кластеризованные таблицы — это таблицы, имеющие кластеризованный индекс. Строки данных хранятся по порядку ключа кластеризованного индекса.
- Кучи — это таблицы, которые не имеют кластеризованного индекса. Строки данных хранятся без определенного порядка, и какой-либо порядок в последовательности страниц данных отсутствует. Страницы данных не связаны в связный список.

Структуры кластеризованного индекса (КИ)

В SQL Server индексы организованы в виде сбалансированных деревьев. Каждая страница в сбалансированном дереве индекса называется узлом индекса. Верхний узел сбалансированного дерева называется корневым. Узлы нижнего уровня индекса называются конечными. Все уровни индекса между корневыми и конечными узлами называются промежуточными. В КИ конечные узлы содержат страницы данных базовой таблицы. На страницах индекса корневого и промежуточного узлов находятся строки индекса. Каждая строка индекса содержит ключевое значение и указатель либо на страницу промежуточного уровня сбалансированного дерева, либо на строку данных на конечном уровне индекса. Страницы на каждом уровне связаны в двунаправленный список.

Для каждого КИ таблица sys.partitions содержит одну строку со значением index_id равным 1 для каждой секции индекса. По умолчанию КИ занимает одну секцию. Если КИ занимает несколько секций, каждая секция содержит сбалансированное дерево, содержащее данные этой секции. Например, если КИ занимает четыре секции, существует четыре сбалансированных дерева: по одному в каждой секции.

- В зависимости от типов данных, каждая структура кластеризованного индекса состоит из одной или более единиц распределения, которые применяются для хранения и управления данными секции.
- Столбец root_page в таблице sys.system_internals_allocation_units содержит указатели на корневые узлы кластеризованного индекса для каждой секции. SQL Server движется вниз по индексу, чтобы найти строку, соответствующую ключу кластеризованного индекса. Чтобы найти диапазон ключей, SQL Server сначала находит начальное значение ключа в диапазоне, а затем сканирует страницы данных, используя указатели на следующую и предыдущую страницу. Чтобы найти первую страницу в цепочке страниц данных, SQL Server движется по самым левым указателям от корня индекса.
- На следующем рисунке изображена структура кластеризованного индекса для одной секции.


id
index_id = 1
root_page
Корневой узел
назад
далее
Строки индекса
Промежуточный уровень
назад
далее
Строки индекса
назад
далее
Строки индекса
назад
далее
Строки индекса
Концевые узлы или страницы данных
назад
далее
Строки данных
назад
далее
Строки данных
назад
далее
Строки данных
назад
далее
Строки данных
назад
далее
Строки данных

Структуры некластеризованных индексов

Некластеризованные индексы имеют ту же структуру сбалансированного дерева, что и кластеризованные индексы; существуют только следующие различия:

- строки данных в базовой таблице не сортируются и хранятся в порядке, который основан на их некластеризованных ключах;
- конечный уровень некластеризованного индекса состоит из страниц индекса вместо страниц данных.

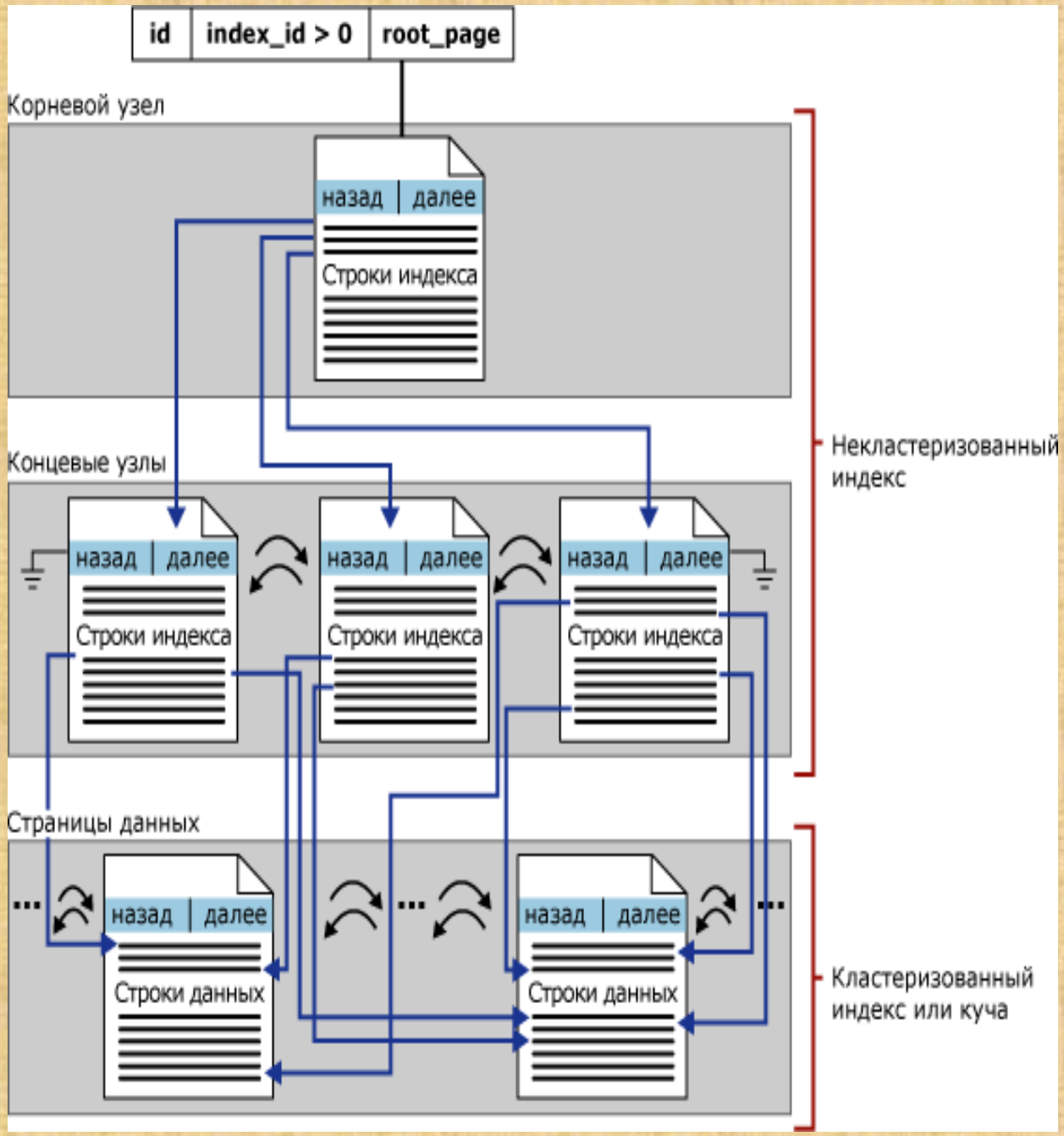
Некластеризованные индексы могут определяться на таблице или представлении с кластеризованным индексом, или на куче. Каждая строка некластеризованного индекса содержит некластеризованное ключевое значение и указатель на строку. Этот указатель определяет строку данных кластеризованного индекса или кучи, содержащую ключевое значение.

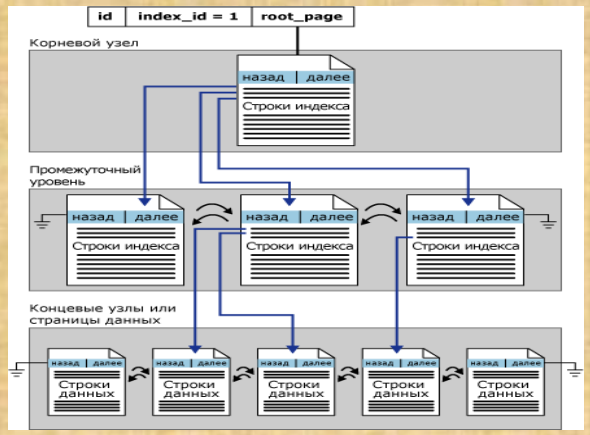
Указатели строк на строках некластеризованных индексов являются либо указателем на строку, либо ключом кластеризованного индекса для строки, как описано ниже.

- Если таблица является кучей, что означает, что она не содержит КИ, то указатель строки является указателем на строку. Указатель строится на основе идентификатора файла (ID), номера страницы и номера строки на странице. Весь указатель целиком называется идентификатором строки (RID).
- Если для таблицы имеется КИ или индекс построен на индексированном представлении, то указатель строки — это ключ КИ для строки. Если КИ не является уникальным индексом, то SQL Server делает все имеющиеся повторяющиеся ключи уникальными путем добавления внутри созданного значения, называемого uniqueifier. Это четырехбайтовое значение невидимо для пользователей. Оно используется тогда, когда необходимо сделать кластеризованный ключ уникальным, чтобы использовать в некластеризованных индексах. SQL Server получает строку данных путем поиска по КИ, используя ключ КИ, который хранится в конечной строке некластеризованного индекса.

Для некластеризованных индексов есть одна строка в таблице sys.partitions со значением столбца index_id >0 для каждой секции, используемой индексом. По умолчанию некластеризованный индекс включает одну секцию. Если некластеризованный индекс состоит из нескольких секций, то каждая секция имеет структуру сбалансированного дерева, в которой содержатся индексные строки для данной конкретной секции. Например, если некластеризованный индекс состоит из четырех секций, то существуют четыре структуры сбалансированного дерева, по одной на каждую секцию.

На следующей иллюстрации показана структура некластеризованного индекса, состоящего из одной секции.


id
index_id > 0
root_page
Корневой узел
назад
далее
Строки индекса
Некластеризованный индекс
Концевые узлы
Страницы данных
Строки данных
Кластеризованный индекс или куча


id
index_id = 1
root_page
Корневой узел
Промежуточный уровень
Концевые узлы или страницы данных

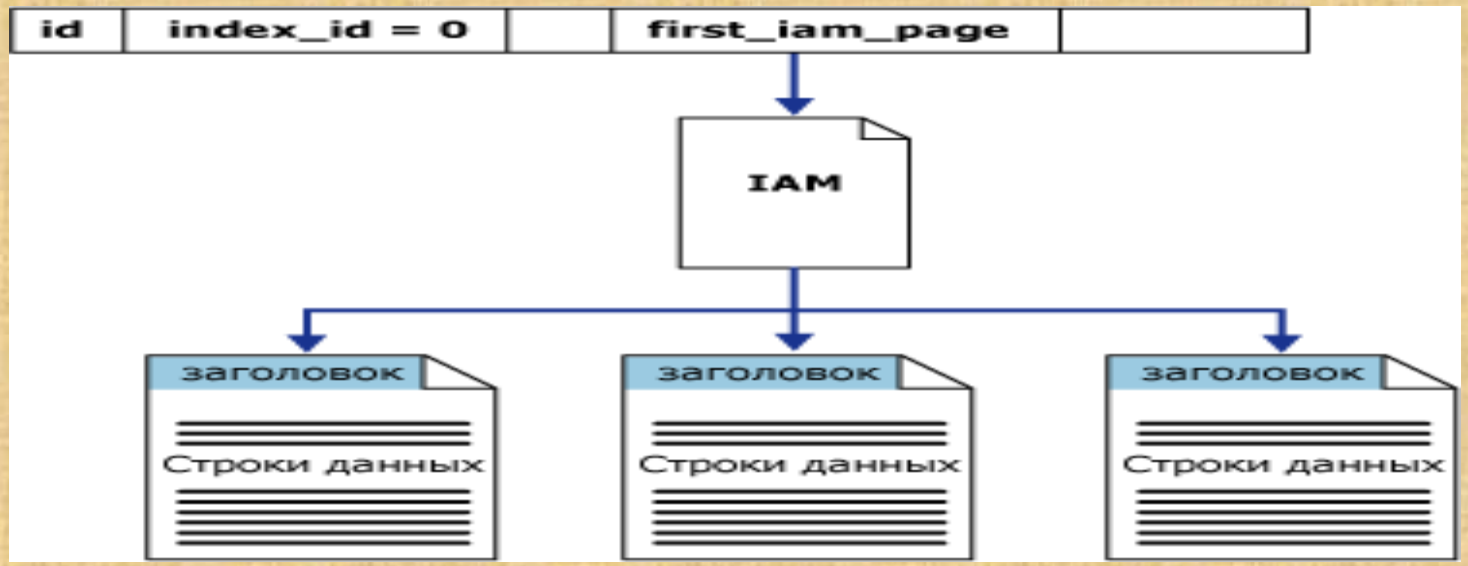
Структуры кучи

Кучей является таблица без кластеризованного индекса. Для каждой кучи существует одна строка в представлении sys.partitions с index_id = 0 для каждой секции, используемой кучей. По умолчанию у кучи есть одна секция. Если куча имеет несколько секций, каждая из них имеет структуру кучи, содержащую данные для этой определенной секции. Например, если у кучи четыре секции, имеются четыре структуры кучи, по одной на каждую секцию.

Столбец first_iam_page в системном представлении sys.system_internals_allocation_units указывает на первую IAM-страницу в цепи IAM-страниц, управляющей выделением пространства куче в определенной секции. SQL Server использует IAM-страницы для перемещения по куче. Страницы данных и строки в этих страницах не расположены в каком-либо порядке и не связаны. Единственным логическим соединением страниц данных являются данные, записанные в IAM-страницы.

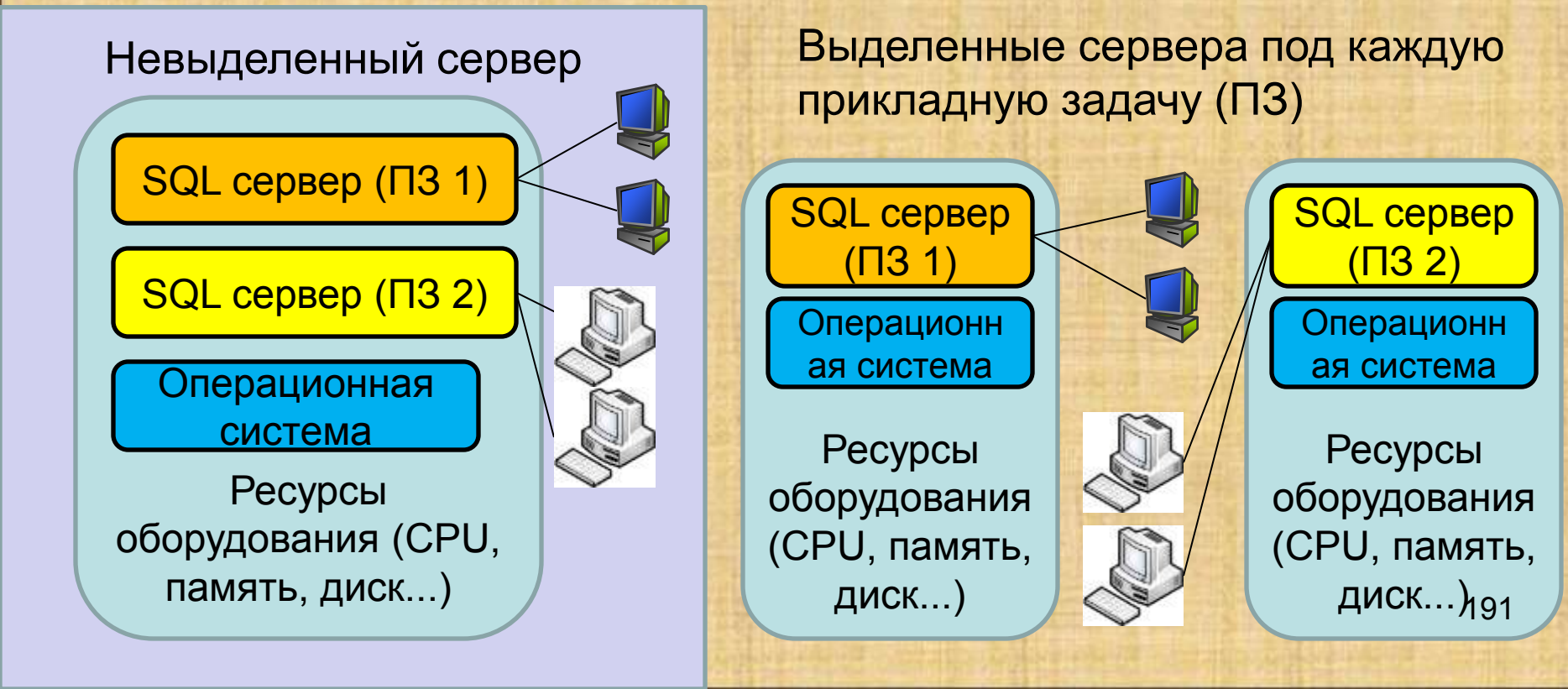
Просмотр таблиц или последовательное считывание в куче может выполняться просмотром IAM-страниц для нахождения экстентов, хранящих страницы кучи. Так как карта IAM представляет экстенты в том же порядке, в котором они существуют в файлах данных, это означает, что последовательный просмотр кучи выполняется последовательно в каждом файле. Использование IAM-страниц для определения последовательности просмотра означает также, что строки из кучи обычно возвращаются не в том порядке, в котором они вставлялись.

На следующей иллюстрации демонстрируется, как компонент SQL Server Database Engine использует IAM-страницы для получения строк данных из кучи с одной секцией.

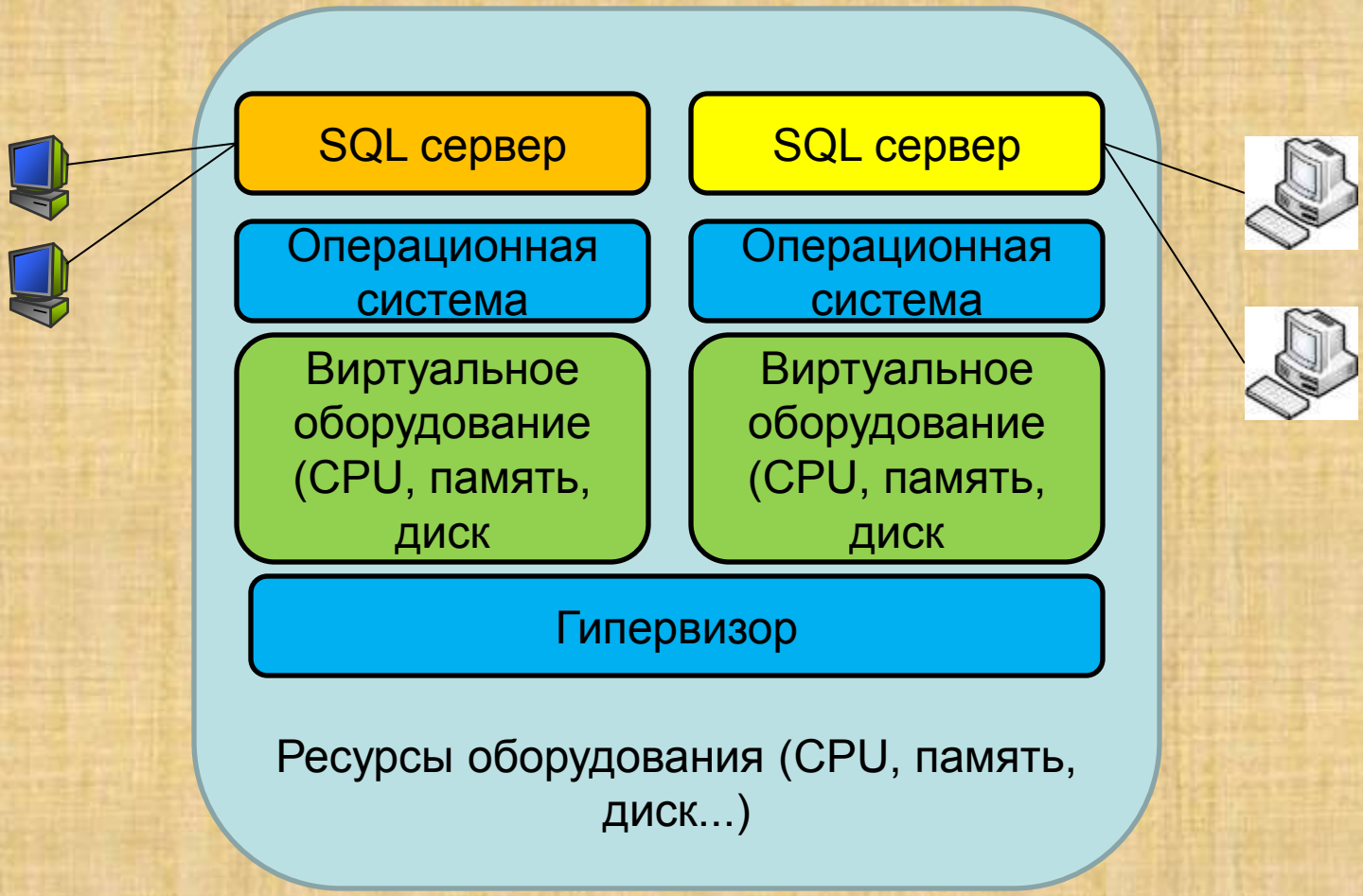


Проектирование инфраструктуры серверных решений

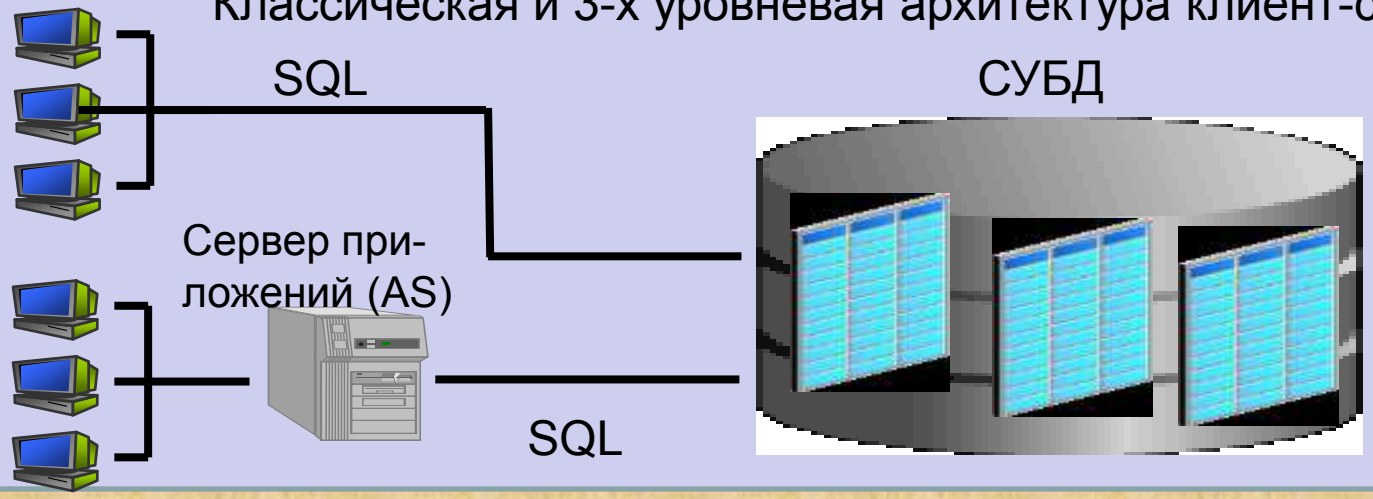
Инфраструкту́ра (лат. *infra — ниже, под* и лат. *structura — строение, расположение*) — комплекс взаимосвязанных обслуживающих структур, составляющих и/или обеспечивающих основу для решения проблемы (задачи).



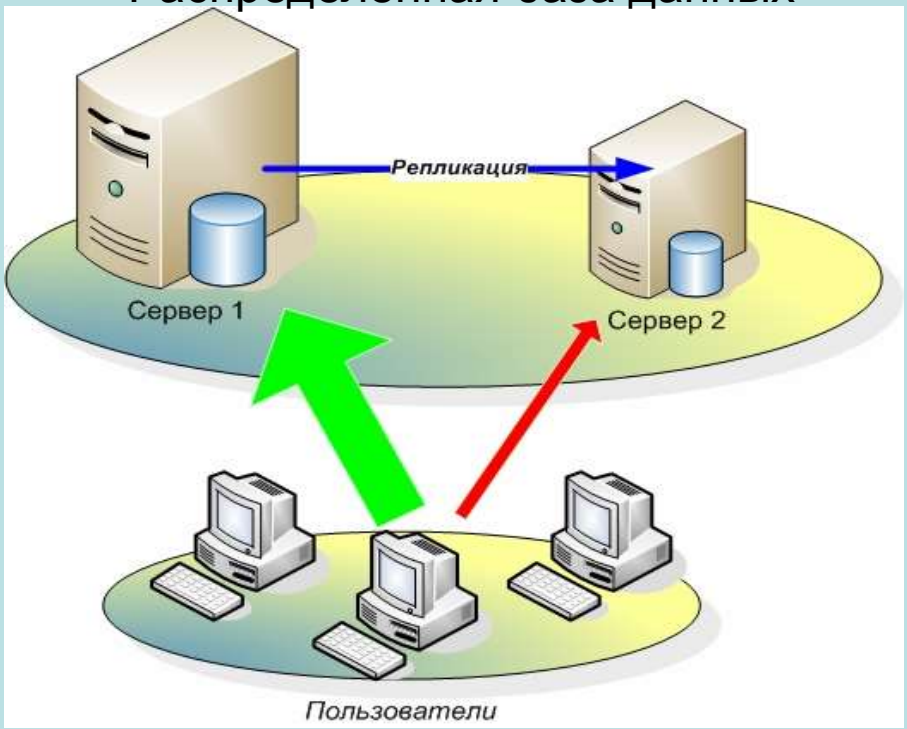
Виртуализация серверов



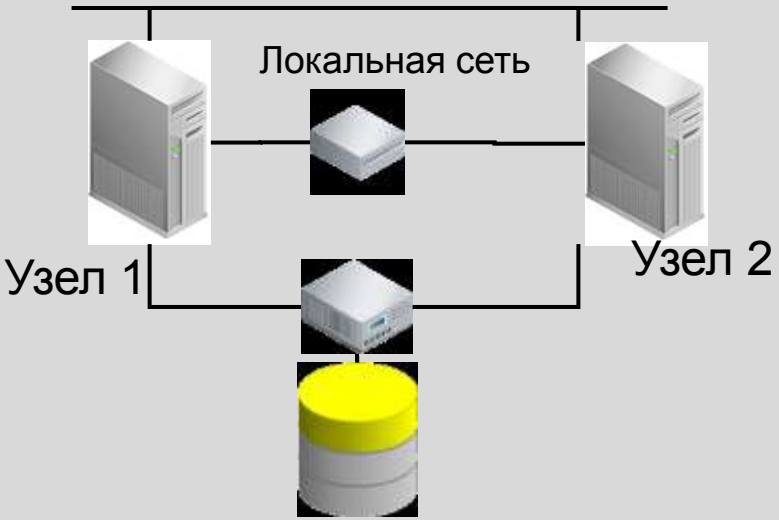
Классическая и 3-х уровневая архитектура клиент-сервер



Распределенная база данных



Кластеризация серверов



Анализ требований к системе (по мощности, памяти, сетевому трафику)

При разработке инфраструктуры серверов баз данных в первую очередь необходимо проанализировать их нагрузку, которая определяется решаемыми в организации задачами.

Такой анализ включает в себя как изучение текущей загруженности системы, так и определение перспектив её роста.

В основном работа идет по следующим 4-м направлениям:

- Оценка требований к внешней памяти
- Оценка требований к центральному процессору
- Оценка требований к оперативной памяти
- Оценка требований к сети

Внешняя память (хранилища данных)

Необходимое дисковое пространство
=
дисковое пространство для файлов БД
+
дисковое пространство для транзакционных log-файлов
+
дисковое пространство для tempdb
+
дисковое пространство для полнотекстовых индексов
+
дополнительное пространство для реорганизации индексов
(необходимо и для файлов БД и для журнала транзакций)

Если оценка произведена верно, и для всех файлов места достаточно, то MS SQL Server для работы не потребуется дополнительная динамическая память, которая может привести к фрагментации файлов базы.

Производительность работы диска

Необходимо определить какая скорость I/O диска требуется для БД. Для определения количества выполняемых операций I/O диска обычно используют два счетчика в System Monitor: Physical Disk:Disk Read Bytes/sec и Physical Disk:Disk Write Bytes/sec.

Если возможно установить соответствие между этими операциями и операциями чтения/записи на диск для транзакций, то можно повысить производительность работы диска.

Когда из БД в основном производится чтение, то значение счетчика Avg. Disk Read Queue Length уменьшают, а если в БД в основном ведется запись, то уменьшают значение счетчика Avg. Disk Write Queue Length.

Размещение и роль серверов БД

Необходимо определить какие данные являются необходимыми для распределенных БД. Например, сервера в филиалах организации могут хранить только некоторое подмножество из данных, которые находятся на главном сервере. Основываясь на роли серверов, можно определить каким базам может потребоваться дополнительное пространство для работы, а каким хватит небольших ресурсов.

Выполнение нормативных требований

В любой организации может существовать свой набор требований по хранению и защите данных. Эти ограничения также должны быть учтены при определении необходимой внешней памяти. Например, какие-то данные должны храниться только в зашифрованном виде (это увеличивает занимаемый ими объем), какие-то должны быть доступны всегда и с высокой скоростью с любых рабочих станций, а другие только в локальной сети и т.д.

Определение требований к внешней памяти в будущем

1. Определить период оценки – на сколько лет (месяцев или дней) необходимо создать «запас» памяти для развития системы. Это работа производится совместно со специалистами в прикладных областях, работающими с данными.

2. Определить предполагаемый рост занятости для дискового пространства. Например, зная среднее количество заключаемых в день договоров, можно определить, сколько места на диске дополнительно потребуется в следующем месяце.

3. Определить формулу для расчета необходимого пространства.

a) Линейный рост Linear growth

Если в течение всего периода оценки занятое дисковое пространство будет увеличиваться на постоянную величину, то используется следующая формула

```
Future disk space required=Current disk space +
   Disk  space growth amount * Number of periods
```

Например, если 500 Гб БД растет на 100 Гб/год, то через три года ее размер будет 800Гб.

b) Сложный рост Compound growth

Если увеличение дискового пространства происходит с постоянной скоростью, то расчет идет по формуле

```
Future disk space required=Current disk space *
(1 + Growth percent rate) ^ Number of periods
```

Например, если 500 Гб БД растет на 2% в месяц в течение 2-х лет, то через два года ее размер будет 500*(1+0.02)^24=804 Гб

c) Геометрическая прогрессия Geometric growth

Если увеличение дискового пространства происходит на некоторую величину (increment), но эта величина сама растет с постоянной скоростью, то объем необходимого пространства увеличивается в геометрической прогрессии и рассчитывается по формуле

```
Future disk space required = Current disk
space +(Initial increment * (1 – increment
growth rate ^(Number of periods +1)))/( 1 –
increment growth rate)
```

Например, если 500 Гб БД увеличивается в начале на 2 Гб в месяц, а затем эта приращение само увеличивается на 2% в месяц, то через 12 месяцев БД займет 500+(2*(1-1.02:12))/(1-1.02) = 529 Гб.

Центральный процессор

Анализ текущей производительности

При анализе текущей производительности CPU сервера БД рассматриваются следующие факторы:

Affinity mask settings. По умолчанию, каждый поток (thread) выполняющийся в экземпляре SQL Server использует следующий доступный процессор. Однако, можно установить affinity mask (маска предпочтения), чтобы использовать для экземпляра определенное подмножество процессоров. Дополнительно, установка affinity mask гарантирует, что каждый поток использует один и тот же процессор. Это ограничивает переключение (*swapping*) потоков между несколькими процессорами и улучшает использование кэша 2-го уровня.

Текущая загруженность CPU. Для определения производительности следует определить базовую загруженность CPU в текущем окружении. Вначале найти количество подключенных пользователей и количество данных приложений. Затем следует определить текущую загруженность CPU, используя инструменты мониторинга (например, System Monitor). Наконец, установить связь между этими параметрами, и тогда возможно будет предсказать изменение загруженности CPU.

Распознавание проблем (trouble spots). Проблемы с железом (hardware bottleneck), перекомпиляция хранимых процедур, использование курсоров являются одними из главных причин снижения производительности CPU. Для идентификации проблем следует использовать счетчики, включенные в SQL Server Plan Cache и SQL Server: SQL Statistics из состава System Monitor.

Выбор типа процессора

SQL Server поддерживает как 32-разрядные, так и 64-разрядные CPU, также поддерживается многоядерность CPU и технология hyperthreading. При оценке требований к CPU следует учитывать преимущества, предоставляемые различными типами процессоров.

Многоядерность процессора позволяет использовать каждое ядро как отдельный процессор, что повышает производительность системы.

Поддержка hyperthreading позволяет использовать логические CPU по отдельности, но так как они обращаются к одним и тем же ресурсам, то нет эта ситуация не дает преимущества перед использованием различных физических процессоров.

Преимущества 64-разрядных процессоров перед 32-разрядными:

- *б*ольший объем напрямую адресуемой памяти. Сервер, выполняющийся под Windows Server 2003 на 64-разрядной архитектуре может поддерживать 1.024 Тб физической памяти и 512Гб адресуемой памяти. А 32-разрядный сервер напрямую может адресовать только 3Гб физической памяти. Чтобы сервер мог работать с б*о*льшей памятью необходимо задействовать переключатель Address Windowing Extensions (AWE).

- Лучшее управление кэшем. 64-разрядный процессор размещает разделы памяти (memory structures) SQL Server (кэш запросов, пул подключений, менеджер блокировок) используя всю доступную память. 32-разрядный CPU не позволит разместить эти разделы в дополнительной памяти, которая стала доступной после включения AWE.

- Расширенные возможности параллельной работы процессоров. 64-разрядная архитектура поддерживает 64 процессора, а 32-разрядная только 32.

Оперативная память

Анализ текущей загруженности памяти

Определение доступной и используемой физической памяти. Необходимо помнить, что ОП используется не только самим сервером, но и ОС, а также процессами, выполняющимися на сервере. Счетчики System Monitor, такие как Memory: Available Mbytes, Memory: Pages/sec, Memory: Committed Bytes, Memory: Commit Limit позволяют определить доступную и используемую память.

Анализ загрузки памяти сервером. Определить количество памяти используемой каждым экземпляром сервера, можно используя счетчик SQL Server: Memory Manager:Total Server Memory из System Monitor (и некоторых др. счетчиков).

Определение размера БД.

Определение памяти занятой подключениями к серверу.

Идентификация возможных проблем (trouble spots). Определить максимальную и минимальную загрузки, базовый уровень загрузки памяти. Сравнить текущую загрузку с базовой, и установить, в каких случаях может возникнуть нехватка памяти.

Влияние типа процессора на использование памяти SQL Server'ом

Тип CPU определяет как один экземпляр SQL Server использует память. Рассмотрим 2-разрядный сервер БД с 8Гб физической памяти.

Если сервер использует стандартную конфигурацию памяти, то максимальный объем напрямую адресуемой памяти будет 4Гб. Из этой памяти 2 Гб резервируются для Windows.

Оставшиеся 2Гб будут доступны для таких приложений как SQL Server. Для увеличения памяти доступной для SQL Server, можно активизировать /3G, Physical Address Extension (PAE) и Address Windowing Extensions (AWE) переключатели памяти.

Установка опции /3G ограничивает память, требующуюся для Windows до 1Гб, увеличивая до 3Гб память для SQL Server.

С включенным PAE 1Гб памяти используется Windows, и 2Гб для SQL Server.

Однако, сервер может использовать больше 4Гб памяти (они свободны!). В результате сервер может выполнять несколько экземпляров SQL Server без использования дисковой памяти. Если включить оба переключателя /3G, PAE, сервер увеличит память для SQL Server до 3Гб. Также можно использовать переключатели /3G, PAE вместе с AWE. Эта комбинация позволит использовать SQL Server больше чем 4Гб памяти. В дополнение, более 3Гб памяти может быть использовано для размещения экземпляров SQL Server.

Теперь рассмотрим 64-разрядный сервер с 8Гб физической памяти. Сервер может напрямую адресовать всю доступную память. В результате, сервер может предоставить 7Гб для SQL Server без требования конфигурировать переключатели памяти.

Так 64-разрядные процессоры имеют больший объем напрямую адресуемой памяти, то они используются для серверов БД с большим требованиями к объему памяти.

Стратегии архивирования, распределения, восстановления данных

Требования к объему архивируемых данных:

1) бизнес требования. Объем и состав online-данных, доступных для пользователей зависит от вида деятельности организации. Это обычно определяется в соответствии с рекомендациями тех, кто работает с данными.

2) нормативные требования. Бизнес требования могут регулировать время, в течение которого данные должны быть в online-доступе (например, информация о банковских операциях). Однако, нормативными требованиями может быть установлено, что данные должны быть сохранены в архиве, но доступ к ним должен быть открыт как только это потребуется.

3) степень детализации архивных данных. Возможна ситуация, когда пользователям не нужна детализация данных за некоторый период. В таких случаях, можно сохранить обобщенных таблицы в online, и сохранить данные в архиве.

При определении устройства для архивации необходимо определить **временные требования к доступности данных**. Например:

1) доступ через 24 ч. Если доступ к архиву должен быть получен не позже чем через сутки, то можно использовать в качестве устройства хранения магнитные ленты (медленно, дешево, испытанно).

2) доступ через 2 ч. В этом случае можно использовать архивные сервера. Такие сервера требуют не много ресурсов, и имеют более низкую производительность в сравнении с главным сервером базы. Объем диска архивного сервера обычно такой же, или больше чем у главного.

3) немедленный доступ. Для мгновенного доступа к архивным данным их необходимо хранить на отдельных серверах с достаточной производительностью для поддержки запросов к архивам.

Определение структуры архивных данных.

Можно структурировать архивные данные, используя следующие типы таблиц.

Разделенные (Partitioned). Разделенные таблицы введены в SQL Server 2005 и более удобны при управлении большими таблицами и индексами, чем объединение разделенных представлений (partitioned view) . Можно размещать разделенные таблицы и их индексы в разных файловых группах. Также можно автоматически перераспределять данные между разными разделами таблицы.

Нормализованные. Совместное архивирование связанных данных сохраняет исторический контекст данных. Для сохранения исторического контекста можно использовать нормализованные таблицы в структуре архивных данных.При использовании нормализованных таблиц необходимо учитывать, что таблицы согласовывают изменения в lookup-значениях или связанных таблицах. Одним из путей выполнения этого является добавление даты изменений в таблицы, также надо определять даты изменений для lookup-значений.

Денормализованные. Если архив всех связанных данных целиком не доступен, можно использовать денормализованные таблицы (ДТ) для сохранения исторического контекста данных. Такие таблицы сохраняют актуальные данные быстрее, чем ссылки на текущие данные. Поэтому эти таблицы более удобны и полезны для оптимизированных запросов, чем сложные join. Например, мы хотим заархировать таблицу Orders содержащую первичный ключ названый CustomerID. Если существующее имя покупателя изменяется в текущий момент и изменения не касаются архивных данных, то начальное имя покупателя теряется. Для сохранения теряемой информации можно архивировать с дополнительным столбцом, таким как CustomerName, который содержит то значение, которое было в архиве. В дополнение к ДТ можно использовать индексированные представления денормализованных данных. Так как ДТ содержат данные на физическом уровне, мы можем получать данные из них быстрее чем из индексированных представлений. Однако, ДТ требуют дополнительного пространства, кроме того их периодически надо перестраивать, и их нельзя автоматически обновлять подобно индексированным представлениям.

Сводные (summary). Возможно, что нет необходимости в детализированных данных за определенный период. В таких случаях можно использовать сводные таблицы для online-доступа и архивировать детальные данные и сохранять их offline. Например, рассмотрим базу, которая сохраняет доход от месячных продаж некоторой продукции. После нескольких лет администратор базы данных архивировать детальные данные и сохранять только результаты за месяц.

Стратегия консолидации серверов БД

Организации используют все больше приложений для управления бизнес-процессами, предоставления новых услуг и анализа эффективности работы предприятий, что вызывает значительный рост числа серверов приложений и серверов, хранящих данные. Зачастую расходы на приобретение оборудования, развертывание и поддержку множества серверов ложатся на организацию тяжким бременем.

В то же время мощность оборудования и возможности ПО позволяют современным ИТ-системам справляться с куда большей нагрузкой, чем было возможно раньше. Этот процесс стимулировал консолидацию программных сервисов на небольшом числе физических серверов, позволяя обойтись меньшим количеством оборудования, полнее задействовав имеющиеся серверы. В результате удается снизить затраты на приобретение и поддержку серверов и повысить эффективность управления ИТ-инфраструктурой.

Преимущества

1. Уменьшение стоимости

Сокращение избыточности данных. Например, в организации в нескольких отделах есть сервера баз данных, на которых хранятся копии размещенных на центральном сервере данных. Консолидация таких серверов в один исключает необходимость хранения локальных копий пересекающихся данных.

Лицензирование ПО. Например, если заменить два сервера баз данных на один сервер, на котором выполняются два экземпляра SQL Server, можно сэкономить стоимость одной лицензии. Однако, для заменяющего сервера может требоваться больше ресурсов для поддержки двух экземпляров. Дополнительно может понадобиться лицензия на Enterprise SQL Server вместо лицензий на Standard SQL Server. Поэтому необходимо сравнить расходы в этих случаях.

Железо. Консолидация серверов снижает необходимость в железе, таком как устройства хранения и бесперебойного питания. Дополнительно консолидация может уменьшить расходы на обслуживание такого железа.

2. Упрощение администрирования.

Для администратора базы уменьшается время на выполнение таких задач, как мониторинг производительности, создание архивов, оптимизация запросов. Но с другой стороны для обслуживания консолидированных серверов может понадобиться персонал более высокой квалификации.

3. Снижение сложности управления безопасностью.

Обычно пользователи имеют различные права доступа и ограничения для каждой базы. Консолидация может уменьшить сложность управления и правами и ограничениями для серверов. Тем не менее надо понимать, что консолидация не уменьшает (compromise) защиту сервера базы и баз размещенных на нем.

Риски

Кроме преимуществ от консолидации серверов, необходимо учитывать и следующие риски:

Единая точка отказа. Когда много баз консолидированы на одном сервере, необходимо учитывать, что все базы будут недоступны в случае отказа сервера. Для устранения всех возможных единых точек отказа все аппаратные компоненты должны быть избыточными (иметь 100%-ное резервирование); все аппаратные компоненты должны удовлетворять спецификациям горячей установки и горячей замены; программно-аппаратные средства (firmware) должны иметь возможность обновления без нарушения работоспособности системы (без необходимости ее перезагрузки или прерывания работы); данные в буферной памяти должны быть полностью защищены от любой вероятной катастрофы. Кроме того, все данные должны быть защищены от искажения, а сам центр - от неожиданных катастроф.

Падение производительности сервера. После консолидации серверов, производительность консолидированного сервера может уменьшиться из-за увеличения нагрузки. Необходимо оценить для сервера загруженность диска, памяти, процессора, сети, чтобы определить как увеличение загруженности сервера повлияло на его производительность.

Жесткий диск. Рассмотрим увеличение загруженности дисковой подсистемы по отношению к использованному пространству и скорости I/O на консолидированном сервере. Для оценки активности I/O дисков каждого из серверов участвовавших в консолидации можно использовать инструмент SQLIO.EXE.

Память. Требования к консолидированной памяти обычно являются суммой памяти использованной отдельными серверами. Однако, такое допущение может оказаться неверным, если некоторые базы разделяют таблицы. В этом случае, требования к консолидированной памяти могут быть меньше, чем суммарная память отдельных серверов. Аналогичным образом, количество пользовательских подсоединений, которое консолидированный сервер должен поддерживать может быть меньше , чем количество подключений к отдельным серверам.

Процессор. Определение требований к процессору подобно определению требований к памяти. Требования к CPU могут не быть суммой загруженности CPU отдельных серверов, в частности, в случае, если сервера используются в разное время.

Сеть. Клиентский трафик из/к консолидированному серверу может возрасти. Это увеличение трафика может уменьшить производительность routers и, в некоторых случаях firewall. Хотя клиентский трафик может возрасти, трафик между серверами уменьшиться.

Обычно говорят о следующих трех типах консолидации:

- серверов - перемещение децентрализованных, но конгруэнтных приложений, распределенных на различных серверах компании, в один кластер централизованных гомогенных серверов;
- систем хранения - совместное использование централизованной системы хранения данных несколькими гетерогенными узлами (хостами);
- приложений - размещение нескольких приложений на одном хосте.

Методы консолидации серверов можно разделить на три категории: географическая, физическая, логическая.

Рассмотрим организацию, которая имеет несколько серверов баз данных в своих подразделениях. Базы размещенные на этих серверах могут быть перемещены на один сервер, на котором выполняются один и или несколько экземпляров SQL Server. Это называется физической консолидацией, и уменьшает количество серверов нуждающихся в управлении. Если использовать многопроцессорный секционированный сервер как сервер консолидации, можно переместить базы департаментов в различные секции сервера. Можно выполнять SQL Server в каждой секции и назначить соответствующие ресурсы дисков, процессоров и памяти. Поэтому, каждая секция может быть активирована как отдельный физический сервер.

Предположим, что организация имеет офисы размещенные в различных частях мира. Каждый офис имеет свой собственный сервер баз данных. Если организация установит централизованное хранилище данных, все сервера из офисов могут быть перемещены на хранилище данных. Это называется географической консолидацией и содействует централизованному управлению.

Предположим, что организация имеет две базы - Orders и Customers, для обработки заказов на продажу (sales orders). Тем не менее, эти базы размещены на двух различных серверах, на каждом из которых также размещены другие базы. Можно консолидировать базы Orders и Customers выполняя два экземпляра SQL Server на одном сервере. Это называется логической консолидацией, и обычно используется когда базы данных со сходной функциональностью, что предпочтительнее чем сервера целиком (rather than complete servers), должны быть консолидированы.

Для логической консолидации также можно использовать виртуальные машины. Они создаются с использованием ПО подобного Microsoft Virtual PC. Мы можем выполнять множество экземпляров SQL Server на одном сервере баз данных инсталлируя их в различные виртуальные машины. Каждая виртуальная машина ведет себя подобно независимому серверу баз данных со своим системой регистрации и операционной системой. Так как каждый экземпляр связан с виртуальной машиной, которая работает rather than физический сервер, виртуальная машина может быть легко перенесена с одного сервера баз данных на другой. Другая форма логической консолидации использует Windows-on-Windows 64-bit, или WOW64, операционную систему. Можно консолидировать множество 32-bit SQL Server экземпляров сервера баз данных на 64-bit CPU, выполняя каждый экземпляр как независимую WOW64 сессию. Создавая план консолидации для организации необходимо учитывать их бизнес-требования и выбрать подходящие пути консолидации.

Примерный план консолидации

Оценка текущей инфраструктуры:

- определение емкости и степени использования серверов;
- документация процессов поддержки и управления;
- инвентаризация инфраструктуры серверов;
- определение стоимости управления и поддержки инфраструктуры.

Определение целей консолидации серверов:

- определение высших деловых и технологических приоритетов проекта;
- указание среди приоритетов возможности определенных уступок, включая большую доступность, уменьшение стоимости и гибкость инфраструктуры;
- определение целей емкости новой среды;
- определение целей стоимости для управления и поддержки консолидированной инфраструктуры;
- составление графика и бюджета проекта консолидации.

Создание нового окружения:

- поиск возможностей программного и аппаратного обеспечения;
- выбор инфраструктуры на основании требований емкости и роста;
- ориентировка на отказоустойчивость и избыточность.

Разработка плана миграции:

- оценка вариантов консолидации с точки зрения бизнеса;
- определение организационных функций и ответственности в ходе процесса консолидации и после него;
- полная оценка плана, рисков, бюджета и требуемых результатов перед внедрением.

Внедрение нового пилотного окружения:

- определение потребностей в программном и аппаратном обеспечении;
- определение особенностей сети и инфраструктуры;
- допуск технических ограничений и рисков;
- построение и тестирование консолидированной среды.

Окончательное оформление плана миграции пользователей и данных:

- запись процедур для перемещения пользователей и данных для нового окружения;
- создание детального графика развертывания, включая планирование непредвиденных обстоятельств;
- выбор критерия для продолжения.

Внедрение новой производственной среды:

- установка приложений, служебных программ и инструментов в новой консолидированной среде;
- разработка и документирование процедур управления и обслуживания после консолидации.

Миграция пользователей и данных в новое окружение:

- проверка (до начала миграции) того, что существуют планы реагирования на непредвиденные обстоятельства, а также планы соответствующего резервирования;
- выполнение миграции в соответствии с детальным расписанием;
- тестирование консолидированной среды, включая пользователей и данные;
- переход к новой производственной среде.

Оценка и обзор проекта:

- оценка результатов процесса консолидации, включая стоимость и процедуры обслуживания;
- повторная регулярная оценка консолидации;
- оптимизация среды.

Стратегии распределения данных

Распределение данных включает в себя пересылку данных между серверами БД. Обычно организации нуждаются в передаче данных между главным сервером в центре и несколькими географически удаленными серверами или наоборот. Для создания плана распределения данных необходимо определить требования к данным для каждого места размещения. Также можно использовать инструменты СУБД для распределения данных между точками размещения.

Предпосылки к использованию распределения данных

Распределение данных помогает поддерживать мобильность пользователей, уменьшает нагрузку при составлении отчетов на главный сервер, упрощает управление хранилищами данных, синхронизирует географически разделенные данные, и гарантирует высокую доступность критически важной информации.

Уменьшение отчетной нагрузки. Выполнение запросов на отчеты может существенно снизить скорость выполнения OLTP задач на главном сервере БД, в результате блокировок или других связанных с производительностью аспектов. Можно минимизировать нагрузку на главный сервер, распределяя read-only копии данных по различным серверам отчетов. Сервер отчетов может следовать другой политике индексации данных, чем главный сервер, для повышения производительности генерации отчетов. Также возможно увеличивать производительность сервера отчетов используя денормализованные или обобщенные таблицы для архивных отчетных данных.

Упрощение управления хранилищами данных. Хранилища данных используются при консолидации и организации данных для отчетов и анализа. Для эффективного составления отчетов и аналитических запросов хранилища данных должны иметь доступ ко всем данным которые хранятся на OLTP сервере баз данных.

Хотя хранилища данных построены по реляционной технологии, их архитектура отличается от архитектуры OLTP БД. Различия заключаются в следующем:

- в отличие от OLTP БД, некоторые таблицы в хранилище могут быть не полностью нормализованными.
- критерии производительности и бизнес-требования, которые управляют архитектурой хранилища отличаются от таких же требований для OLTP. Обычно хранилища проектируются для быстрой загрузки и запросов, но не для обновления., в отличие от OLTP.

Поэтому данные должны быть преобразованы, прежде чем они будут переданы с производственного сервера в хранилище. Такие инструменты распределения данных как SSIS (SQL Server Integration Services), делают данные, хранящиеся на производственном сервере пригодными для хранения в хранилище.

Синхронизация географически удаленных данных. Организации часто имеют несколько филиалов, которые географически удалены. Централизовано хранящиеся данные должны быть распределены между этими офисами, так чтобы каждый офис мог выполнять необходимые операции. При этом обновление данных должно быть синхронизировано со всеми офисами имеющими доступ к текущим данным.

Высокая доступность критически важных данных. Мы можем распределить копии критически важных данных между разными серверами для того, чтобы данные были доступны даже при отказе главного сервера.

Поддержка мобильных пользователей. Некоторые сотрудники организации, такие как торговые представители, много ездят. Во время поездок такой персонал нуждается в доступе к данным хранящимся в центральной БД, например, к данным о продажах, ценах и т.д. Эти служащие должны иметь локальные копии таких данных. В свою очередь торговые представители должны сообщать информацию о своих продажах в центр, т.е. загружать данные на главный сервер.

Инструменты для распределения данных

SQL Server имеет четыре инструмента для распределения данных:

- репликация
- SSIS (SQL Server Integration Services)
- Service Broker
- RDA (Remote-Data-Access)

Репликация

Репликация SQL Server представляет собой набор технологий, с помощью которых данные или объекты баз данных можно скопировать и перенести из одной базы данных в другую, а затем синхронизировать эти базы данных для обеспечения согласованности.

Модель публикации репликации

Репликация использует терминологию издательской отрасли для представления компонентов в топологии репликации.

- Publisher (Издатель) - сервер или база данных, которая посылает данные на другой сервер или в другую базу данных.
- Subscriber (Подписчик) - сервер или база данных, которая получает данные от другого сервера или другой базы данных.
- Distributor (Распространитель) - сервер, который управляет потоком данных через систему репликации. Этот сервер содержит специализированную базу данных: Distribution database.

Publisher содержит публикацию/публикации. Публикация - это совокупность одной или более статей, которые посылаются серверу подписчику (subscriber) или базе данных. Статья (Article) - основной модуль репликации и это может быть таблица или подмножество таблицы. Подписка (subscriptions) - это группа данных, которые сервер или база данных получает.

Последовательность действий можно описать следующим образом.

- Издатель журнала производит одну или несколько публикаций
- Публикация содержит статьи
- Издатель или распространяет журнал напрямую, или использует распространитель
- Подписчики получают публикации, на которые они подписались

Хотя сравнение с журналом полезно для понимания репликации, следует отметить, что репликация SQL Server включает возможности, которые не представлены в данной метафоре, в частности возможность подписчика выполнять обновления и возможность издателя посылать дополнительные изменения в статьи публикации.

Топология репликации определяет отношения между серверами и копиями данных, и проясняет логику, определяющую порядок обмена данными между серверами. Существует несколько процессов репликации (называемых *агентами*), которые отвечают за копирование и перемещение данных между издателем и подписчиками. Следующая иллюстрация представляет собой обзор компонентов входящих в репликацию.

Существуют три типа репликации:

- snapshot (репликация моментальных снимков),
- transactional (репликация транзакций)
- merge (репликация слиянием)

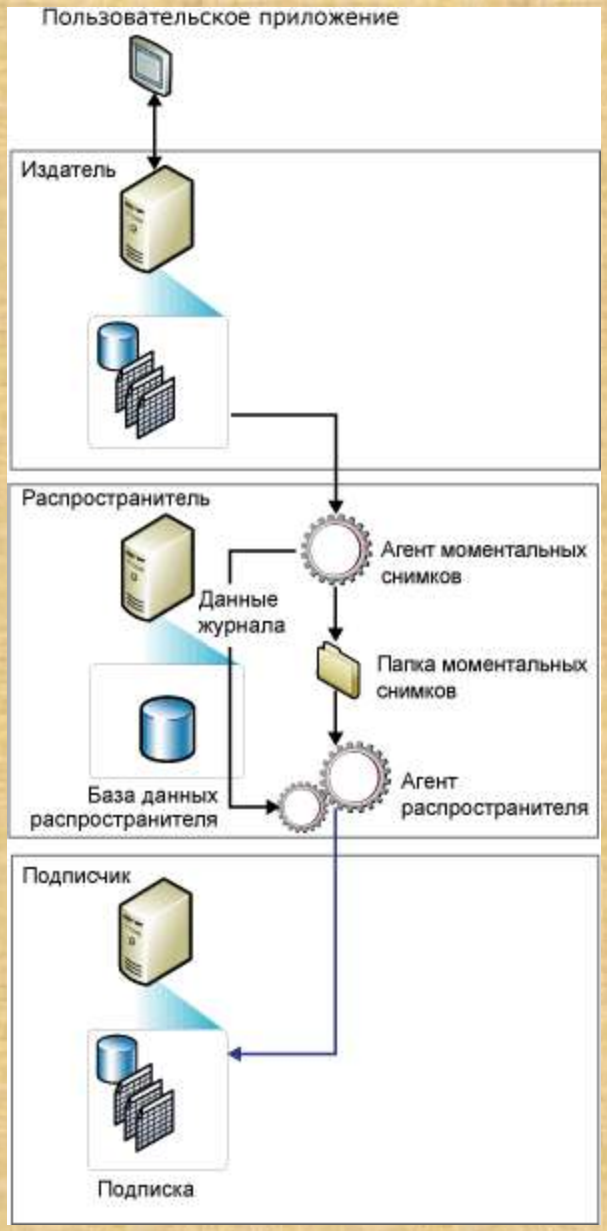
Репликация моментальных снимков распространяет данные точно в том виде, в котором они были представлены в определенный момент времени, и не контролирует обновления этих данных. Во время синхронизации формируется моментальный снимок и отсылается подписчикам целиком.

Использование репликации моментальных снимков самой по себе наиболее приемлемо, когда выполняется одно или несколько следующих условий:

- Данные изменяются редко.
- Допустимо на определенный период времени иметь копии данных, устаревших по отношению к издателю.
- Репликация небольших объемов данных.
- Большой объем изменений производится за короткий период времени.

Например, если торговая организация ведет прайс-лист на товары, и цены обновляются в одно и то же время раз или два раза в год, рекомендуется производить репликацию всего моментального снимка данных после их изменений. При наличии определенных типов данных наиболее подходящими могут быть более частые моментальные снимки. Например, если относительно небольшая таблица обновляется на издателе в течение дня, но допустима некоторая задержка, то изменения могут доставляться ночью в виде моментального снимка.

По умолчанию все три типа репликации для инициализации подписчиков используют моментальный снимок. Агент моментальных снимков SQL Server всегда создает файлы моментальных снимков, но агент, доставляющий файлы, меняется в зависимости от используемого типа репликации. Репликация моментальных снимков и репликация транзакций используют для доставки этих файлов агент распространителя, в то время как репликация слиянием использует агент слияния SQL Server. Агент моментальных снимков выполняется на распространителе. Агент распространителя и агент слияния выполняются на распространителе для принудительных подписок и на подписчиках для подписок по запросу.



Моментальные снимки могут создаваться и применяться или непосредственно после создания подписки, или в соответствии с расписанием, установленным при создании публикации.

Агент моментальных снимков подготавливает файлы моментального снимка, содержащие схему и данные опубликованных таблиц и объектов базы данных, сохраняет эти файлы в папке моментальных снимков для издателя и записывает данные слежения в базу данных распространителя на распространителе.

При настройке распространителя указывается папка моментальных снимков по умолчанию, но можно указать и другое расположение публикации вместо или дополнительно к расположению по умолчанию.

Основные компоненты репликации моментальных снимков.

Репликация транзакций обычно используется в серверных средах и пригодна в следующих случаях:

- Необходимо, чтобы добавочные изменения распространялись подписчикам без задержек по мере появления.
- Для приложения необходимы малые задержки между моментом внесения изменений на издателе и моментом прибытия изменений на подписчик.
- Для приложения необходим доступ к промежуточным состояниям данных. Например, если строка изменяется пять раз, репликация транзакций позволяет приложению реагировать на каждое изменение (например, срабатывание триггера), а не просто на итоговое изменение строки.
- На издателе выполняется очень большой объем вставок, обновлений и удалений.
- Издатель и подписчик являются базами данных, отличными от баз данных SQL Server (например, Oracle).

По умолчанию подписчики на публикации транзакций должны быть доступны только для чтения, так как изменения не распространяются обратно на издатель. Однако репликация транзакций предоставляет возможности, позволяющие выполнять обновления у подписчика.

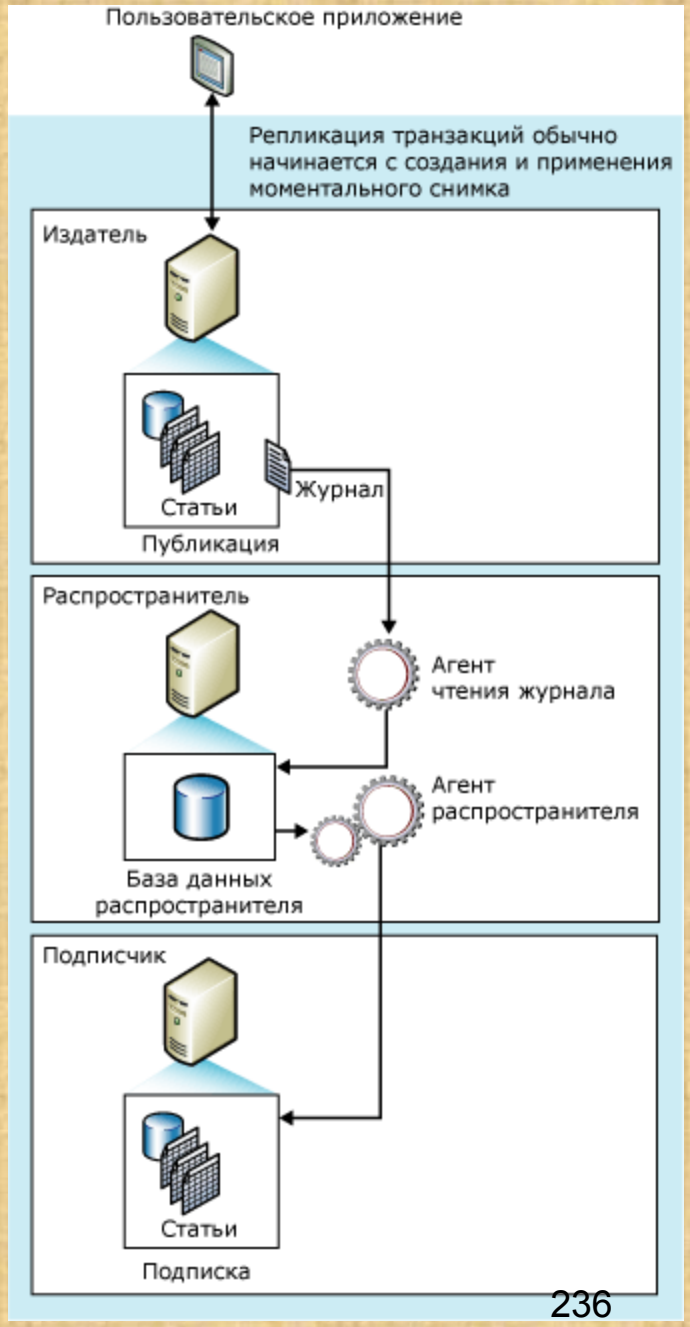
Репликация транзакций реализуется агентом моментальных снимков, агентом чтения журналов и агентом распространителя SQL Server. Агент моментальных снимков готовит файлы моментальных снимков, содержащие схему, данные публикуемых таблиц и объекты базы данных, хранит файлы в папке моментальных снимков и записывает задания синхронизации в базу данных распространителя на распространителе.

Агент чтения журнала контролирует журнал транзакций всех баз данных, настроенных для репликации транзакций, и копирует транзакции, отмеченные для репликации, из журнала транзакций в базу данных распространителя, которая действует как надежная очередь с функциями хранения и переадресации данных. Агент распространителя копирует файлы исходного моментального снимка из папки моментальных снимков и транзакции, хранимые в таблицах базы данных распространителя, на подписчики.

Добавочные изменения, вносимые на издателе, поступают подписчикам согласно расписанию агента распространителя, который может выполняться непрерывно для достижения минимальной задержки либо в запланированные интервалы времени.

Поскольку изменения данных должны вноситься на издателе (когда репликация транзакций используется без немедленного обновления и отложенного обновления), возникновение конфликтов обновления исключается.

В конечном счете, для всех подписчиков устанавливаются такие же значения, как и на издателе. Если с репликацией транзакций используется немедленное или отложенное обновление, в этом случае обновления могут вноситься на подписчике, и для отложенного обновления возможно возникновение конфликтов.



Репликация слиянием, как и репликация транзакций, как правило, начинается с моментального снимка объектов и данных базы данных публикации. Последующие изменения данных и схемы, произведенные на стороне издателя и подписчиков, отслеживаются при помощи триггеров. Подписчик синхронизируется с издателем при подключении к сети и обменивается с ним всеми строками, которые изменились со времени последней синхронизации издателя и подписчика.

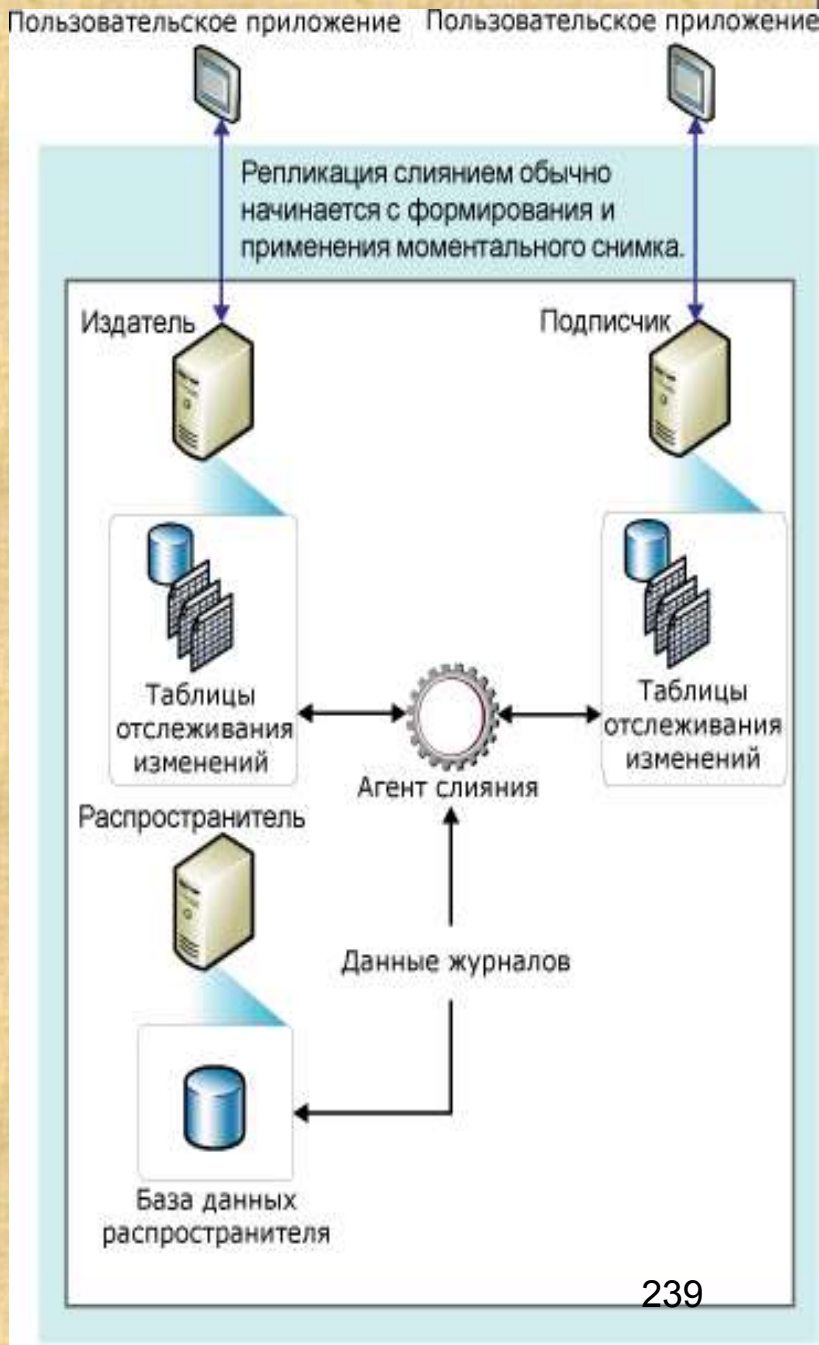
Как правило, репликация слиянием применяется в средах «сервер-клиент». Репликация слиянием подходит для любой из следующих ситуаций.

- Несколько подписчиков могут обновлять одни и те же данные в разное время и передавать эти изменения на издатель и на другие подписчики.
- Подписчикам нужно получить данные, внести изменения в автономном режиме и позднее синхронизировать изменения с издателем и другими подписчиками.

- Каждому подписчику нужна индивидуальная секция данных.
- Поскольку возможно возникновение конфликтов, необходимы средства по распознаванию и разрешению конфликтов.
- Приложению требуется конечное изменение данных, а не доступ к промежуточным состояниям данных. Например, если строка меняется пять раз на подписчике до его синхронизации с издателем, на издателе строка изменится только один раз для отображения конечного изменения данных (то есть, пятого значения).

Репликация слиянием позволяет различным узлам работать автономно, и позднее выполнить слияние обновлений в единый результат. Поскольку обновления выполняются на нескольких узлах, одни и те же данные могут быть обновлены издателем и несколькими подписчиками. Поэтому при слиянии обновлений могут возникать конфликты, и репликация слиянием предоставляет несколько способов обработки конфликтов.

Репликация слиянием реализуется агентом моментальных снимков SQL Server и агентом слияния. Если публикация не отфильтрована или использует статические фильтры, агент слияния создает один моментальный снимок. Если публикация использует параметризованные фильтры, то агент слияния создает моментальный снимок для каждой секции данных. Агент слияния применяет все исходные моментальные снимки на подписчиках. Он также объединяет добавочные изменения данных, которые возникли на издателе или подписчиках после создания исходного моментального снимка, выявляет и разрешает любые конфликты в соответствии с заданными вами правилами.



Компоненты, используемые при репликации слиянием.

Повышение производительности репликации

Перед настройкой репликации рекомендуется ознакомиться с факторами, влияющими на производительность репликации:

- серверное и сетевое аппаратное обеспечение;
- структура базы данных;
- конфигурация распространителя;
- структура и параметры публикации;
- структура фильтра и его использование;
- параметры подписки;
- параметры моментального снимка;
- параметры агентов.
- обслуживание

После настройки репликации рекомендуется разработать базовый уровень производительности, позволяющий определить работу репликации при рабочей нагрузке, типичной для используемых приложений и топологии. Следует применить монитор репликации и системный монитор для определения типичных значений следующих пяти измерений производительности репликации:

• Задержка: время распространения изменений данных между узлами в топологии репликации.
• Пропускная способность: величина репликационной активности (измеряемая в командах, доставленных за период времени), поддерживаемой системой.
• Параллелизм: число процессов репликации, которые могут выполняться системой одновременно.
• Длительность синхронизации: время, затрачиваемое на выполнение заданной синхронизации.
• Потребление ресурсов: аппаратные и сетевые ресурсы, используемые для обработки репликации.

Задержка и пропускная способность наиболее полно характеризуют репликацию транзакций, поскольку для систем, основанных на репликации транзакций, обычно требуется малая задержка и высокая пропускная способность.

Параллелизм и длительность синхронизации наиболее точно характеризуют репликацию слиянием, поскольку системы, основанные на репликации слиянием, часто имеют большое число подписчиков, а издатель может иметь значительное число параллельно выполняющихся синхронизаций с этими подписчиками.

Обеспечение безопасности СУБД

Обеспечение безопасности ПО должно строится на четырех основных принципах.

- **Безопасная архитектура (secure by design).** ПО должно иметь безопасную архитектуру, являющуюся основой для борьбы со злоумышленниками и защиты данных.
- **Безопасная стандартная конфигурация (secure by default).** Системные администраторы не должны тратить силы на то, чтобы сделать только что установленную систему безопасной; это должно обеспечиваться по умолчанию.
- **Безопасное развертывание (secure in deployment).** ПО должно помогать администратору себя защищать, самостоятельно устанавливая последние защитные «заплатки» и обеспечивая удобство поддержки.
- **Обмен информацией (communications).** Обмен передовыми методиками и информацией о постоянно появляющихся новых угрозах позволяет администраторам заблаговременно защищать свои системы.

Безопасность платформы и сети

Платформа для SQL Server включает в себя физическое оборудование и сетевые компьютеры, с помощью которых клиенты соединяются с серверами базы данных, а также двоичные файлы, применяемые для обработки запросов базы данных.

Рекомендуется строго ограничивать доступ к физическим серверам и компонентам оборудования. Например, оборудование сервера базы данных и сетевые устройства должны находиться в закрытых охраняемых помещениях. Доступ к резервным носителям также следует ограничить.

Реализация физической сетевой безопасности начинается с запрета доступа неавторизованных пользователей к сети.

Безопасность операционной системы

В состав пакетов обновления и отдельных обновлений для операционной системы входят важные дополнения, позволяющие усилить безопасность. Все обновления для операционной системы необходимо устанавливать только после их тестирования с приложениями базы данных.

Кроме того, эффективную безопасность можно реализовать с помощью брандмауэров. Брандмауэр, распределяющий или ограничивающий сетевой трафик, можно настроить в соответствии с корпоративной политикой информационной безопасности. Использование брандмауэра повышает безопасность на уровне операционной системы, обеспечивая узкую область, на которой можно сосредоточить меры безопасности.

Безопасность файлов операционной системы SQL Server

SQL Server использует файлы операционной системы для работы и хранения данных. Оптимальным решением для обеспечения безопасности файлов будет ограничение доступа к ним.

Безопасность участников и объектов базы данных

Участники — это отдельные пользователи, группы и процессы, которым предоставлен доступ к ресурсам SQL Server. Защищаемые объекты — это сервер, база данных и объекты, которые содержит база данных. У каждого из них существует набор разрешений, с помощью которых можно уменьшить контактную зону SQL Server.

Шифрование и сертификаты

Шифрование не решает проблемы управления доступом. Однако оно повышает безопасность, ограничивая потерю данных даже в тех редких случаях, когда средства управления доступом удается обойти. Например, если компьютер, на котором установлена база данных, был настроен неправильно, и злонамеренный пользователь смог получить конфиденциальные данные (например, номера кредитных карточек), то украденная информация будет бесполезна, если она была предварительно зашифрована.

Безопасность приложений

Для SQL Server рекомендуется разрабатывать защищенные клиентские приложения.

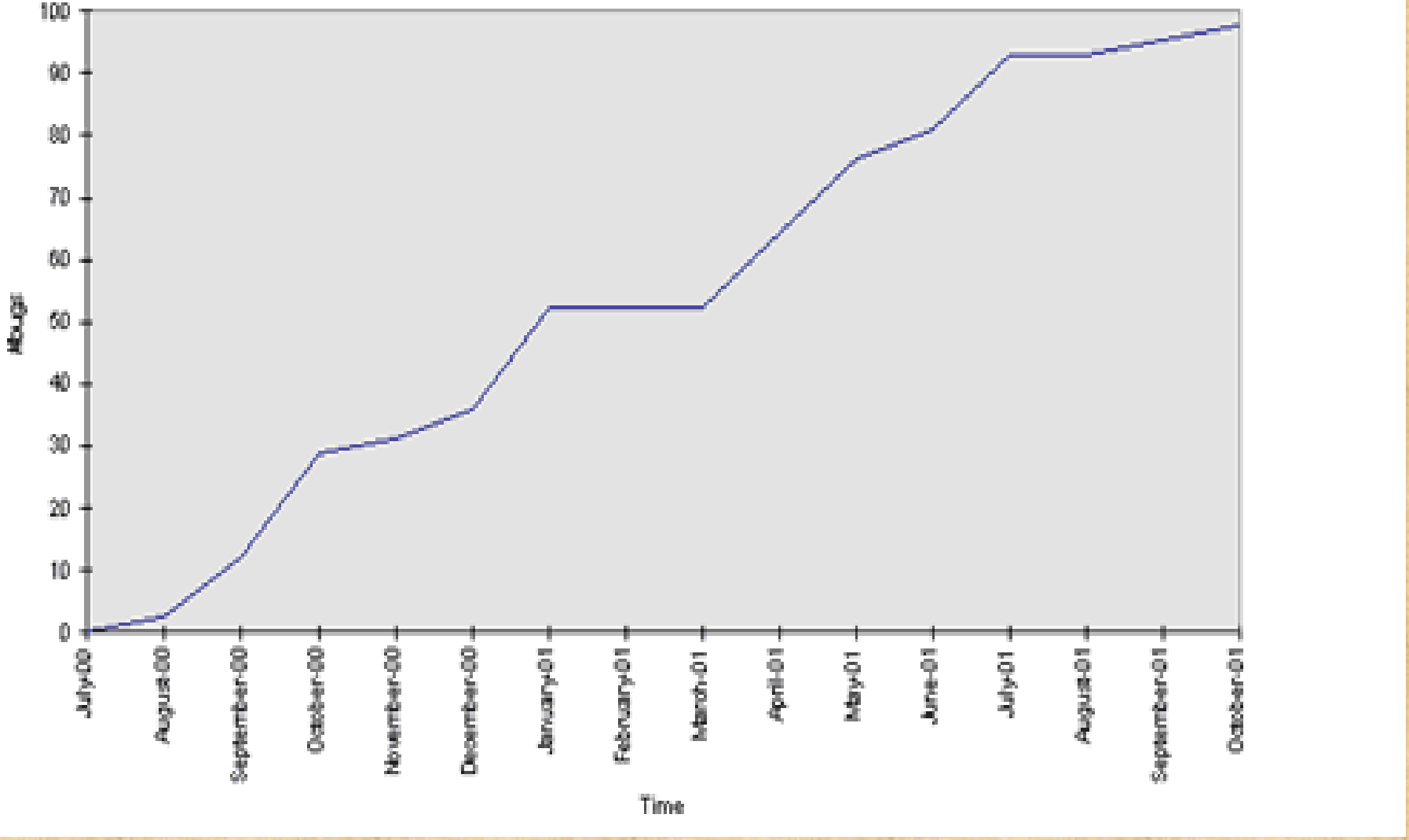
Предотвращение угроз и снижение уязвимости

Несмотря на то, что современные СУБД оснащены разнообразными механизмами защиты, в каждой системе есть функции, уязвимостями которых может воспользоваться злоумышленник. Каждый компонент, который открывает доступ к любым данным, может быть источником опасности при неверной реализации.

В компании Oracle есть специальная служба secalert_us, которая занимается вопросами защиты СУБД.

Половина сообщений, полученных secalert_us, связана с компонентами сетевого доступа; эти сообщения в основном отправляются хакерами и сообществом обеспечения безопасности сетей. Другая половина сообщений в основном связана с доступом к серверам или приложениям. Из сообщений, связанных с компонентами сетевого доступа, половина (25% от общего количества) посвящена листенеру баз данных (компонент сетевого доступа к системам Oracle, принимает клиентские запросы на соединение и направляет их для обработки в соответствующий серверный процесс).

На следующей диаграмме показано распределение количества сообщений об ошибках, связанных с безопасностью, которые ежемесячно поступают в secalert_us (данные нормализованы с целью скрыть фактическое количество сообщений!).



Каждый компонент может представлять риск, но не все угрозы равнозначны. Для некоторых требуется изменить методы применения, для других параметры, для третьих программный код.

Microsoft предлагает использовать следующую таблицу для определения возможных угроз и уязвимостей и способов их устранения.

Угрозы и уязвимости процесса

Угроза или уязвимость	Определение	Меры по снижению риска
Политики безопасности	Политика безопасности представляет собой запись процессов и процедур, которые применяются в организации, чтобы предотвращать, отслеживать и реагировать на угрозы безопасности. Содержит политики для правильного доступа к системам, обновлений, брандмауэров и превентивных антивирусных механизмов.	Создайте, пересматривайте, распространяйте и поддерживайте эффективную политику безопасности.
Принцип «наименьших прав доступа»	В соответствии с принципом «наименьших прав доступа», система должна предоставлять лишь необходимый уровень доступа к защищенному объекту. Кроме того, доступ должен предоставляться только лицам, непосредственно нуждающимся в нем, и только на определенное время.	Пересмотрите и реализуйте безопасность в соответствии с принципом наименьших прав доступа.
Бюллетени безопасности (MS SQL) Оповещения о проблемах безопасности (Oracle)	И Microsoft и Oracle регулярно выпускают оповещения о проблемах безопасности обнаруженных в их СУБД.	Регулярная ознакомление с оповещениями

Угрозы и уязвимости платформы

Угроза или уязвимость	Определение	Меры по снижению риска
Система не обновлена	При отсутствии обновлений, система оказывается более уязвимой для атаки.	Просматривайте и применяйте все пакеты обновления и исправления по мере их выпуска.
Атаки на сетевые порты	Сеть — основной канал доступа при атаках на SQL Server. Открытые стандартные порты в Интернете могут стать причиной атаки.	Используйте брандмауэр на сервере, если он подключен к Интернету, и средство диспетчера конфигурации SQL Server, чтобы настроить конфигурацию сети. Кроме того, возможно использовать протокол SSL для повышения безопасности.
Недопустимые параметры учетной записи службы	Учетным записям служб для SQL Server часто предоставляется больше прав доступа к платформе или сети, чем необходимо.	Работать учетные записи служб для SQL Server должны в соответствии с принципом наименьших прав доступа; они также должны иметь надежные пароли.
Слишком обширная контактная зона	Функции и возможности SQL Server могут быть доступны, когда в них нет необходимости.	Средства «Диспетчер конфигурации SQL Server» и «Управление на основе политики» используются для управления функциями и другими компонентами.
Включены ненужные хранимые процедуры	Некоторые расширенные хранимые процедуры предоставляют доступ к ОС или реестру.	Включайте хранимые процедуры, предоставляющие доступ к операционной системе или реестру, только в случае крайней необходимости.

Угрозы и уязвимости проверки подлинности

Угроза или уязвимость	Определение	Меры по снижению риска
Простые пароли	Простые пароли уязвимы для атак, использующих простой перебор или перебор по словарю.	Всегда используйте надежные, сложные пароли.
Не проводится аудит учетных записей пользователя	Пользователи часто меняют должности или уходят из организации. Если права доступа для учетной записи не изменены, доступ к системе можно получить с прежним уровнем разрешений.	Необходимо регулярно проводить аудит учетных записей пользователя, чтобы убедиться в наличии у них соответствующих прав доступа к серверам БД и объектам.

Программные угрозы и уязвимости

Угроза или уязвимость	Определение	Меры по снижению риска
атака SQL Injection	Внедрение вредоносного запроса в правильный запрос.	
Встроенные пароли	Некоторые приложения сохраняют строки соединения в файлах программы или конфигурации.	Не храните пароли или конфиденциальные сведения о соединении в программе, реестре или файле конфиг-ции.

Угрозы и уязвимости доступа к данным

Угроза или уязвимость	Определение	Меры по снижению риска
Неверно примененный алгоритм шифрования	Шифрование запутывает данные или сведения о соединении в SQL Server. Отсутствие шифрования, когда оно необходимо, или использование шифрования, когда не требуется, приводит к излишнему риску и усложнению.	Необходимо понимать и грамотно применять шифрование SQL Server.
Неверно примененные сертификаты	Сертификаты представляют собой механизмы для проверки подлинности. В SQL Server сертификаты могут использоваться для многих различных целей, от соединений до данных. Неверное использование самостоятельной сертификации и неоправданно увеличенные периоды проверки снижают общую надежность защиты.	Необходимо понимать и грамотно применять сертификаты SQL Server.
Не созданы резервные копии ключей SQL Server	Экземпляр SQL Server и содержащиеся в нем базы данных могут иметь ключи, которые используются в различных защитных целях. В частности, для шифрования.	Необходимо создать резервные копии ключей сервера (главные ключи служб) и ключи БД и хранить их в безопасном месте. Кроме того, их необходимо периодически менять.

Атака SQL Injection

Атака типа SQL Injection — это атака, при которой производится вставка вредоносного кода в строки, передающиеся затем в экземпляр SQL Server для синтаксического анализа и выполнения. Любая процедура, создающая инструкции SQL, должна рассматриваться на предмет уязвимости к вставке небезопасного кода, поскольку SQL Server выполняет все получаемые синтаксически правильные запросы. Даже параметризованные данные могут стать предметом манипуляций опытного злоумышленника.

Основная форма атаки SQL Injection состоит в прямой вставке кода в пользовательские входные переменные, которые объединяются с командами SQL и выполняются. Менее явная атака внедряет небезопасный код в строки, предназначенные для хранения в таблице или в виде метаданных. Когда впоследствии сохраненные строки объединяются с динамической командой SQL, происходит выполнение небезопасного кода.

Атака осуществляется посредством преждевременного завершения текстовой строки и присоединения к ней новой команды. Поскольку к вставленной команде перед выполнением могут быть добавлены дополнительные строки, злоумышленник заканчивает внедряемую строку меткой комментария «--». Весь последующий текст во время выполнения не учитывается.

Следующий сценарий показывает простую атаку SQL Injection. Сценарий формирует SQL-запрос, выполняя объединение жестко запрограммированных строк со строкой, введенной пользователем:

```
var Shipcity;ShipCity = Request.form ("ShipCity");
var sql = "select * from OrdersTable where ShipCity =
'" + ShipCity + "'";
```

Пользователю выводится запрос на ввод названия города. Если пользователь вводит Redmond, то запрос, построенный с помощью сценария, выглядит приблизительно так:

```
SELECT * FROM OrdersTable WHERE ShipCity = 'Redmond'
```

Предположим, однако, что пользователь вводит следующее:

```
Redmond'; drop table OrdersTable--
```

В этом случае запрос, построенный сценарием, будет следующим:

```
SELECT * FROM OrdersTable WHERE ShipCity =
'Redmond';drop table OrdersTable--'
```

Точка с запятой «;» обозначает конец одного запроса и начало другого. А последовательность двух дефисов (--) означает, что остальная часть текущей строки является комментарием и не должна обрабатываться. Если измененный код будет синтаксически правилен, то он будет выполнен сервером. Когда SQL Server будет обрабатывать эту инструкцию, SQL Server прежде всего отберет все записи в OrdersTable, где ShipCity является Redmond. Затем SQL Server удалит OrdersTable.

Если вставленный код SQL синтаксически верен, искаженные данные нельзя выявить программно. Поэтому необходимо проверять правильность всех вводимых пользователями данных, а также внимательно просматривать код, выполняющий созданные с помощью SQL команды на сервере.

Проверка достоверности всех вводимых данных

Всегда проверяйте все данные, вводимые пользователем, выполняя проверку типа, длины, формата и диапазона данных. При реализации мер предосторожности, направленных против злонамеренного ввода данных, учитывайте архитектуру и сценарии развертывания приложения. Помните, что программы, созданные для работы в безопасной среде, могут быть скопированы в небезопасную среду. Рекомендуется следующая стратегия:

1. Не делайте никаких предположений о размере, типе или содержимом данных, получаемых приложением. Например, рекомендуется оценить следующее.

– Как приложение будет вести себя, если пользователь по ошибке или по злому умыслу вставит MPEG-файл размером 10 МБ там, где приложение ожидает ввод почтового индекса?

– Как приложение будет вести себя, если в текстовое поле будет внедрена инструкция DROP TABLE?

2. Проверьте размер и тип вводимых данных и установите соответствующие ограничения. Это поможет предотвратить преднамеренное переполнение буфера.

3. Проверяйте содержимое строковых переменных и допускайте только ожидаемые значения. Отклоняйте записи, содержащие двоичные данные, управляющие последовательности и символы комментария. Это поможет предотвратить внедрение сценария и защитит от некоторых приемов атаки, использующих переполнение буфера.

4. При работе с XML-документами проверяйте все вводимые данные на соответствие схеме.

5. Никогда не создавайте инструкции Transact-SQL непосредственно из данных, вводимых пользователем.

6. Для проверки вводимых пользователем данных используйте хранимые процедуры.

7. В многоуровневых средах перед передачей в доверенную зону должны проверяться все данные. Данные, не прошедшие процесс проверки, следует отклонять и возвращать ошибку на предыдущий уровень.

8. Внедрите многоэтапную проверку достоверности. Меры предосторожности, предпринятые против случайных пользователей-злоумышленников, могут оказаться неэффективными против организаторов преднамеренных атак. Рекомендуется проверять данные, вводимые через пользовательский интерфейс, и далее во всех последующих точках пересечения границ доверенной зоны.
Например, проверка данных в клиентском приложении может предотвратить простое внедрение сценария. Однако если следующий уровень предполагает, что вводимые данные уже были проверены, то любой злоумышленник, которому удастся обойти клиентскую систему, сможет получить неограниченный доступ к системе.

9. Никогда не объединяйте введенные пользователем данные без проверки. Объединение строк является основной точкой входа для внедрения сценария.

10. Не допускайте использование в полях следующих строк, из которых могут быть созданы имена файлов: AUX, CLOCK$, COM1–COM8, CON, CONFIG$, LPT1–LPT8, NUL и PRN.

По возможности отклоняйте вводимые данные, содержащие следующие символы:

Входной символ	Значение в языке Transact-SQL
;	Разделитель запросов.
'	Разделитель строк символьных данных.
--	Разделитель комментариев.
/* ... */	Разделители комментариев. Текст, заключенный между символами /* и */, не обрабатывается сервером.
xp_	Используется в начале имен расширенных хранимых процедур каталога, например **xp_cmdshell**.

Использование SQL-параметров безопасных типов

Коллекция **Parameters** в SQL Server обеспечивает проверку длины и контроль соответствия типов. Если используется коллекция **Parameters**, то вводимые данные обрабатываются как буквенное значение, а не исполняемый код. Дополнительное преимущество использования коллекции **Parameters** состоит в том, что можно использовать принудительные проверки типа и длины данных. Если значение выходит за рамки диапазона, будет вызвано исключение.

```
SqlDataAdapter myCommand = new
SqlDataAdapter("AuthorLogin",
conn);myCommand.SelectCommand.CommandType =
CommandType.StoredProcedure;SqlParameter parm =
myCommand.SelectCommand.Parameters.Add("@au_id",
SqlDbType.VarChar, 11);parm.Value = Login.Text;
```

В этом примере параметр @au_id обрабатывается как буквенное значение, а не исполняемый код. Это значение проверяется по типу и длине. Если значение @au_id не соответствует указанным ограничениям типа и длины, то будет вызвано исключение.

Просмотр кода на предмет возможности атаки SQL Injection

Необходимо просматривать все фрагменты кода, вызывающие инструкции EXECUTE, EXEC или процедуру **sp_executesql**. Чтобы выявить процедуры, содержащие эти инструкции, можно использовать запросы, подобные следующему. Этот запрос проверяет наличие 1, 2, 3 или 4 пробелов после слов EXECUTE и EXEC.

```
SELECT object_Name(id) FROM syscomments
WHERE UPPER(text) LIKE '%EXECUTE  (%'
OR UPPER(text) LIKE '%EXECUTE  (%'
OR UPPER(text) LIKE '%EXECUTE   (%'
OR UPPER(text) LIKE '%EXECUTE    (%'
OR UPPER(text) LIKE '%EXEC (%'
OR UPPER(text) LIKE '%EXEC  (%'
OR UPPER(text) LIKE '%EXEC   (%'
OR UPPER(text) LIKE '%EXEC    (%'
OR UPPER(text) LIKE '%SP_EXECUTESQL%'
```

Атака Injection, проводимая с помощью усечения данных

Любое присваиваемое переменной динамическое значение Transact-SQL усекается, если оно не вмещается в буфер, назначенный для этой переменной. Если организатор атаки способен обеспечить усечение инструкции, передавая хранимой процедуре непредвиденно длинные строки, он получает возможность манипулировать результатом. Так, хранимая процедура, создаваемая с помощью следующего сценария, уязвима для атаки Injection, проводимой методом усечения.

```
CREATE PROCEDURE sp_MySetPassword
@loginname sysname, @old sysname, @new sysname
AS
-- Объявление переменной, и буфер имеет длину 200 символов
DECLARE @command varchar(200)
-- Конструируется динамическая команда T-SQL. Для пароля пользователя ‘sa’
-- необходимо 154 символа.26 для UPDATE, 16 для условия WHERE, 4 - 'sa', и
-- 2 для “” в QUOTENAME (@loginname): 200 – 26 – 16 – 4 – 2 = 154. Но так как -
-- @new объявлена как sysname, эта переменная может иметь 128 символов
SET @command= 'update Users set password=' + QUOTENAME(@new, "") + '
where username=' + QUOTENAME(@loginname, "") + ' AND password = ' +
QUOTENAME(@old, "")
EXEC (@command)
GO
```

Передавая 154 символа в буфер, рассчитанный на 128 символов, злоумышленник может установить новый пароль для sa, не зная старого пароля.

EXEC sp_MySetPassword 'sa', 'dummy',
'12345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012'''
'''

Поэтому для переменной команды следует использовать большой буфер или непосредственно выполнять динамический Transact-SQL внутри инструкции EXECUTE.

Рекомендации по безопасности при установке SQL Server

Повышение физической безопасности

- Физическая и логическая изоляции составляют основу безопасности SQL Server. Для повышения физической безопасности установки SQL Server выполните следующие действия.
- Разместите сервер в помещении, недоступном для посторонних.
- Установите компьютеры, на которых размещены базы данных, в физически защищенных местах, идеальным вариантом является запертая компьютерная комната с системой электромагнитного и противопожарного контроля или системой подавления помех.
- Установите базы данных в безопасной зоне интрасети и не подключайте экземпляры SQL Server к Интернету напрямую.
- Регулярно создавайте резервные копии и храните их в безопасном месте за пределами расположения компьютера.

Использование брандмауэров

Брандмауэры играют важную роль в обеспечении безопасности установки SQL Server. Брандмауэры будут более эффективны, если следовать приведенным ниже правилам.

Установите брандмауэр между сервером и Интернетом. Разрешите работу брандмауэра. Если он отключен, включите его. Если он включен, не отключайте.

Разделите сеть на зоны безопасности, разделенные брандмауэрами. Заблокируйте весь поток данных, после чего разрешите только необходимый.

В многоуровневой архитектуре используйте несколько брандмауэров для создания изолированных подсетей.

При установке сервера внутри домена Windows настройте внутренние брандмауэры на разрешение проверки подлинности Windows.

Если приложение работает с распределенными транзакциями, настройте брандмауэр на обмен данными между отдельными экземплярами координатора распределенных транзакций (MS DTC). Кроме того, нужно настроить брандмауэр на разрешение обмена данными между MS DTC и диспетчерами ресурсов (например, SQL Server).

Изолирование служб

Изолирование служб уменьшает риск того, что подвергнувшаяся опасности служба подвергнет опасности другие службы. Чтобы изолировать службы, следуйте приведенным ниже правилам.

Запускайте разные службы SQL Server под разными учетными записями Windows. Если возможно, пользуйтесь для каждой из служб SQL Server отдельными учетными записями Windows или локальных пользователей, обладающих наименьшими правами.

Настройка безопасной файловой системы

Рекомендуется устанавливать SQL Server на NTFS, так как она обеспечивает более высокую стабильность и восстанавливаемость, чем файловые системы FAT. Кроме того, файловая система NTFS реализует параметры безопасности, например списки управления доступом к файлам и каталогам (ACL), шифрование файловой системы (EFS). Во время установки SQL Server установит необходимые списки ACL на разделы реестра и файлы, если программа установки обнаружит NTFS. Эти разрешения не должны меняться. Возможно, в будущих версиях SQL Server установка на компьютеры с файловыми системами FAT поддерживаться не будет.

Используйте дисковый массив (RAID) для наиболее критичных файлов данных.

Отключение протоколов NetBIOS и SMB

На внешних серверах сети должны быть отключены все ненужные протоколы, включая NetBIOS и SMB.

NetBIOS использует следующие порты:

- UDP/137 (служба имен NetBIOS);
- UDP/138 (служба дейтаграмм NetBIOS);
- UDP/139 (служба сеанса NetBIOS).

SMB использует следующие порты:

- TCP/139
- TCP/445

Веб-серверы и DNS-серверы не требуют наличия NetBIOS или SMB. Отключите на них оба протокола, чтобы снизить угрозу раскрытия списка пользователей.

Безопасное развертывание SQL Server

- Выбор режима проверки подлинности
- Настройка контактной зоны
- Соответствие стандартам безопасности

Выбор режима проверки подлинности

Во время процесса установки следует выбрать режим проверки подлинности. Существует два возможных режима: режим проверки подлинности Windows и смешанный режим. Режим проверки подлинности Windows включает проверку подлинности Windows и отключает проверку подлинности SQL Server. В смешанном режиме включены как проверка подлинности Windows, так и проверка подлинности SQL Server. Проверка подлинности Windows доступна всегда, и отключить ее нельзя.

Настройка режима проверки подлинности

Если во время процесса установки был выбран смешанный режим проверки подлинности, необходимо задать и подтвердить надежный пароль для встроенной учетной записи системного администратора SQL Server с именем **sa**. Учетная запись **sa** устанавливает соединения с помощью проверки подлинности SQL Server.

Если во время процесса установки была выбрана проверка подлинности Windows, программа установки создаст учетную запись **sa** для проверки подлинности SQL Server, но она будет отключена. Если позже переключиться на смешанный режим проверки подлинности и потребуется учетная запись **sa**, ее будет нужно включить. Любая учетная запись Windows или SQL Server может быть настроена в качестве системного администратора. Поскольку учетная запись **sa** широко известна и часто является целью злонамеренных пользователей, ее не рекомендуется включать, за исключением тех случаев, когда приложению это необходимо. Никогда не указывайте пустой или простой пароль для учетной записи **sa**.

Соединение с использованием проверки подлинности Windows

Когда пользователь подключается с помощью пользовательской учетной записи Windows, SQL Server проверяет имя учетной записи и пароль с помощью маркера участника Windows в операционной системе. Это означает, что удостоверение пользователя было подтверждено Windows. SQL Server не запрашивает пароль и не выполняет проверку удостоверения. Проверка подлинности Windows является проверкой подлинности по умолчанию; она обеспечивает более высокий уровень безопасности, чем проверка подлинности SQL Server. Режим проверки подлинности Windows использует протокол безопасности Kerberos, реализует политику паролей в отношении проверки сложности надежных паролей, поддерживает блокировку учетных записей и истечение срока пароля. Соединение, установленное с помощью проверки подлинности Windows, иногда называется доверительным соединением, поскольку SQL Server доверяет учетным данным, предоставляемым Windows.

Соединение с использованием проверки подлинности SQL Server

Если используется проверка подлинности SQL Server, в SQL Server создаются имена входа, которые не основаны на учетных записях пользователей Windows. И имя пользователя, и пароль создаются с помощью SQL Server и хранятся в SQL Server. Пользователи, подключающиеся с помощью проверки подлинности SQL Server, должны предоставлять свои учетные данные (имя входа и пароль) каждый раз при установке соединения. При использовании проверки подлинности SQL Server необходимо задавать надежные пароли для всех учетных записей SQL Server.

Для имен входа SQL Server доступны три дополнительные политики паролей:

- Пользователь должен сменить пароль при следующем входе
- Задать срок окончания действия пароля
- Требовать использование политики паролей Windows

Недостатки проверки подлинности SQL Server

Если пользователь является пользователем домена Windows, имеющим имя входа и пароль Windows, то для подключения он все равно должен предоставить другое имя входа и пароль (SQL Server). Многим пользователям сложно помнить несколько имен входа и паролей.

В проверке подлинности SQL Server не может использоваться протокол безопасности Kerberos.

ОС Windows предоставляет дополнительные политики паролей, недоступные для имен входа SQL Server.

Преимущества проверки подлинности SQL Server

Позволяет SQL Server поддерживать более старые приложения и приложения, поставляемые сторонними производителями, для которых необходима проверка подлинности SQL Server.

Позволяет SQL Server поддерживать среды с несколькими операционными системами, в которых пользователи не проходят проверку подлинности домена Windows.

Позволяет пользователям устанавливать соединения из неизвестных или ненадежных доменов. Например, в приложении, в котором клиенты подключаются с выделенными именами входа SQL Server, чтобы получить состояние их заказов.

Позволяет SQL Server поддерживать веб-приложения, в которых пользователи сами создают собственные удостоверения.

Позволяет разработчикам программного обеспечения распространять свои приложения с помощью сложной иерархии разрешений, основанной на известных, заранее установленных именах входа SQL Server.

Настройка контактной зоны

В конфигурации по умолчанию новых установок SQL Server многие компоненты не включены. SQL Server выборочно устанавливает и запускает только ключевые службы и компоненты для минимизации количества функций, которые могут быть атакованы злонамеренным пользователем. Системный администратор может изменить эти значения по умолчанию в ходе установки, а также включать или отключать функции работающего экземпляра SQL Server по своему выбору. Кроме того, некоторые компоненты могут оказаться недоступными для соединения из других компьютеров, если не выполнена настройка протоколов.

Соответствие стандартам безопасности

Сертификация по стандарту Common Criteria

Во время выпуска SQL Server 2008 официально соответствовал гарантированному уровню соответствия 4 (EAL1) по стандарту Common Criteria

Обзор стандарта Common Criteria

Принятый в качестве международного стандарта в 1999 г., Common Criteria замещает несколько более ранних схем оценки, в том числе «Критерии оценки пригодности компьютерных систем» (TCSEC) (в которой был определен широко известный стандарт безопасности Class C2) в США, «Критерии оценки безопасности информационных технологий» (ITSEC) в Европе и «Критерии оценки пригодности программного обеспечения» (CTCPEC) в Канаде. Стандарт Common Criteria был разработан несколькими государствами с целью расширить доступность продуктов информационных технологий, предлагающих более высокую степень безопасности, помочь пользователям в оценке приобретаемых продуктов и повысить уверенность потребителей в их безопасности. Стандарт Common Criteria поддерживается международной организацией, участниками которой являются более 20 государств, и признается Международной организацией по стандартизации (ISO) как стандарт ISO 15408.

Соответствие стандарту FIPS 140-2

SQL Server 2005 с пакетом обновления 1 (SP1) может быть настроен для соответствия стандарту FIPS 140-2. Стандарт FIPS 140-2 содержит требования безопасности для криптографических модулей. Этот стандарт определяет то, какие алгоритмы шифрования и хэширования могут использоваться и как должны формироваться и управляться ключи шифрования. Чтобы настроить SQL Server 2005 с пакетом обновления 1 (SP1) на соответствие стандарту FIPS 140-2, этот сервер должен работать под управлением ОС Microsoft Windows, которая сертифицирована для работы по стандарту FIPS 140-2 или обеспечивает сертифицированный криптографический модуль.

Безопасная работа с SQL Server

- Политика паролей
- Разрешения и защита объектов базы
- Шифрование SQL Server
- Сертификаты SQL Server и асимметричные ключи
- Аудит (компонент Database Engine)

Политика паролей

При работе в Windows Server 2003 или более поздних версий SQL Server может использовать механизмы политики паролей Windows.Эту возможность обеспечивает API-интерфейс NetValidatePasswordPolicy, доступный начиная с Windows Server 2003.

Сложность пароля

Политика сложности паролей позволяет отражать атаки, использующие простой перебор, путем увеличения числа возможных паролей.

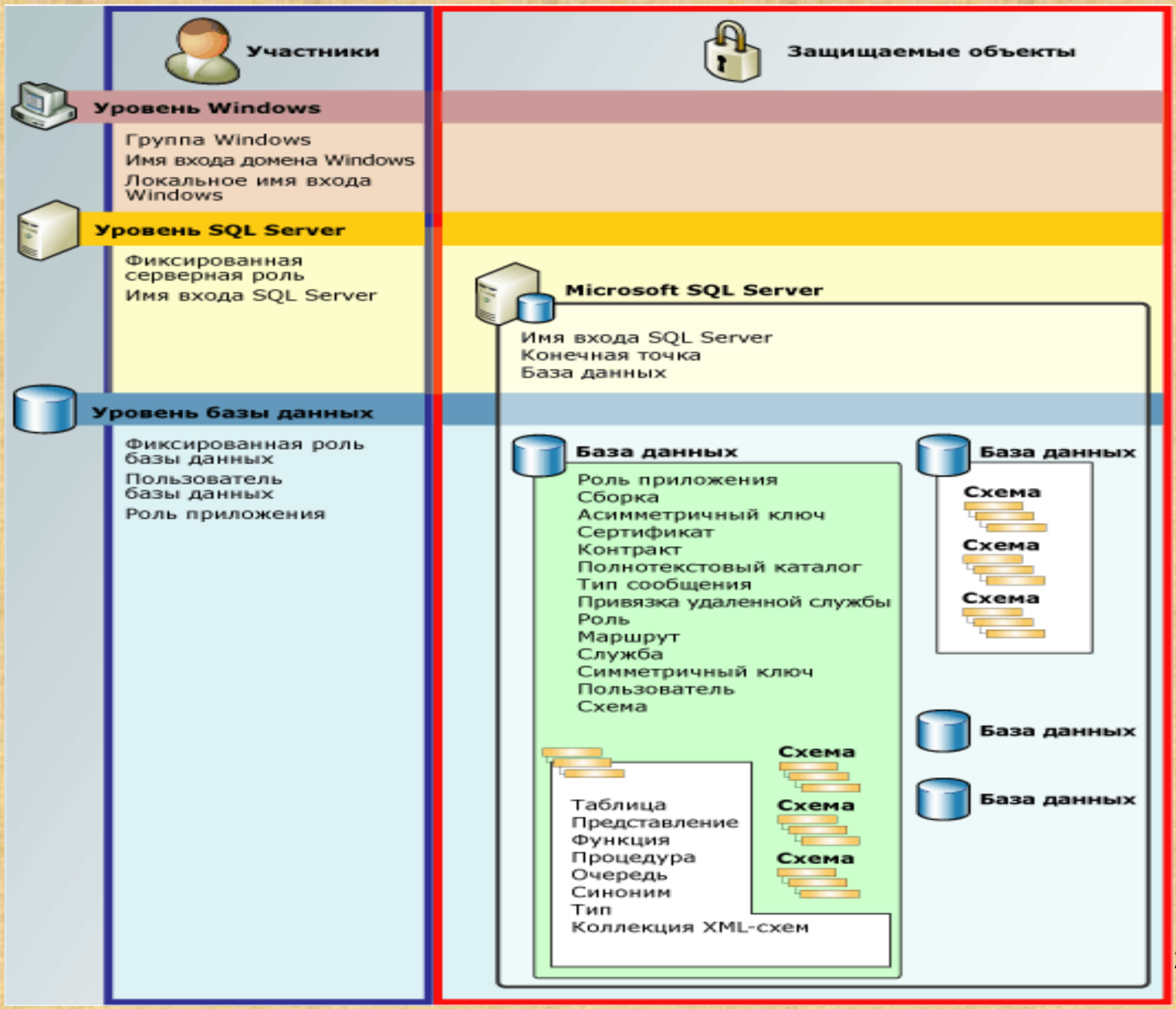
Применение политики

Применение политики паролей можно настроить отдельно для каждого имени входа SQL Server.

Иерархия разрешений

Компонент Database Engine в MS SQL Server управляет иерархической коллекцией сущностей, защита которых производится при помощи разрешений. Эти сущности называются *защищаемыми объектами*. Наиболее общими защищаемыми объектами являются серверы и базы данных, но отдельные разрешения могут устанавливаться на гораздо более глубоком уровне детализации. SQL Server регулирует выполнение участниками действий с защищаемыми объектами, проверяя, обладают ли они соответствующими разрешениями.

На следующей иллюстрации показана связь между иерархиями разрешений компонента Database Engine.


Участники
Защищаемые объекты
Уровень Windows
Группа Windows
Имя входа домена Windows
Локальное имя входа Windows
Уровень SQL Server
Фиксированная серверная роль
Имя входа SQL Server
Уровень базы данных
Фиксированная роль базы данных
Пользователь базы данных
Роль приложения
Microsoft SQL Server
Имя входа SQL Server
Конечная точка
База данных
База данных
Роль приложения
Сборка
Асимметричный ключ
Сертификат
Контракт
Полнотекстовый каталог
Тип сообщения
Привязка удаленной службы
Роль
Маршрут
Служба
Симметричный ключ
Пользователь
Схема
Схема
Схема
Схема
Таблица
Представление
Функция
Процедура
Очередь
Синоним
Тип
Коллекция XML-схем
Схема
Схема
Схема
База данных
База данных
База данных

Работа с разрешениями

Выполнение различных действий с разрешениями осуществляется при помощи обычных запросов Transact-SQL: GRANT, DENY и REVOKE. Сведения о разрешениях доступны через представления каталога sys.server_permissions и sys.database_permissions. Существует также поддержка запроса сведений о разрешениях при помощи встроенных функций.

Цепочки владения

Если несколько объектов БД 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.

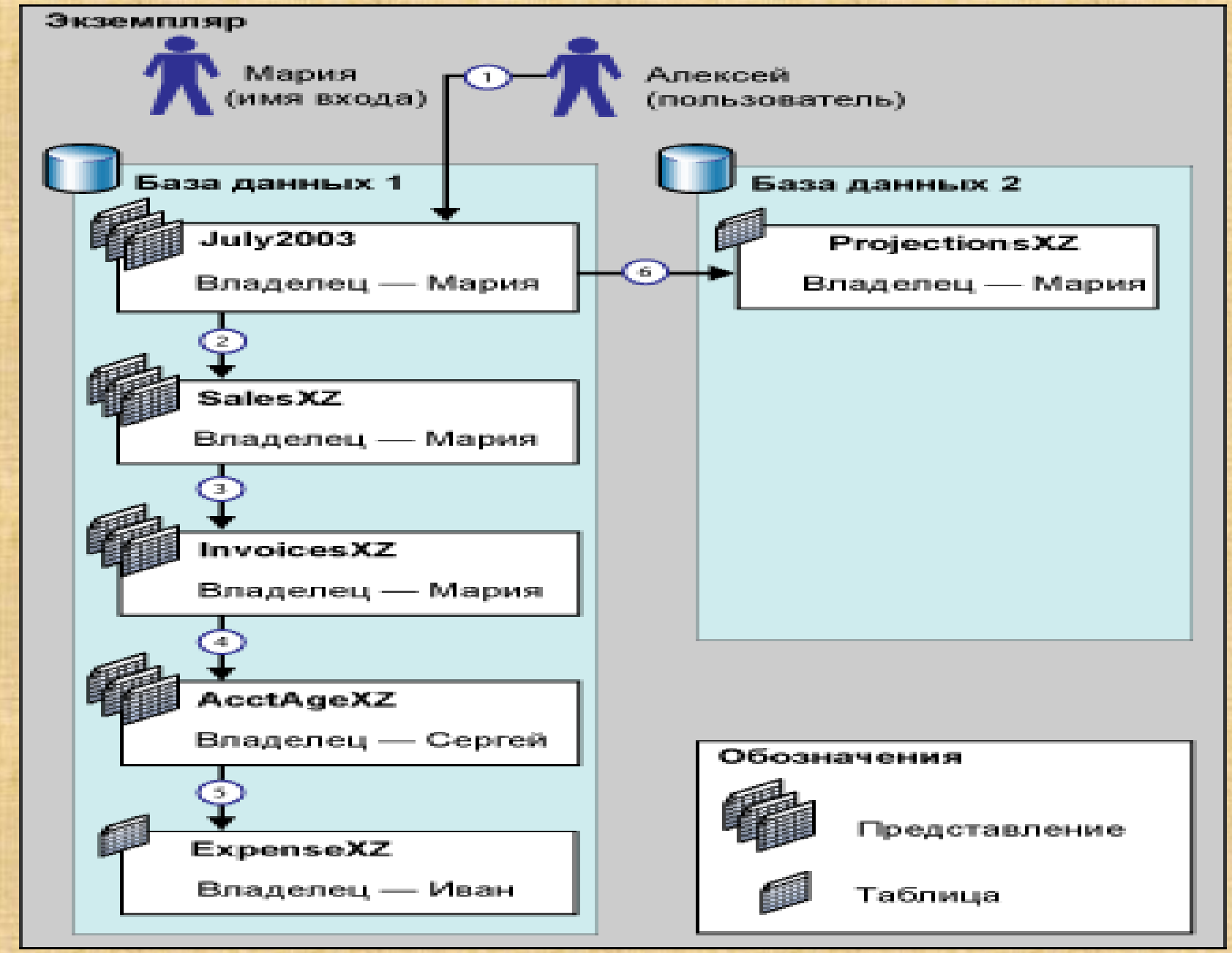
Цепочки владения позволяют управлять доступом к нескольким объектам, таким как таблицы, назначая разрешения одному объекту, например представлению. Цепочки владения также обеспечивают небольшое повышение производительности в случаях, когда позволено пропускать проверку наличия разрешений.

Как выполняется проверка разрешений в цепи

Когда доступ к объекту производится через цепь, SQL Server сначала сравнивает владельца объекта с владельцем вызывающего объекта. Это предшествующее звено в цепи. Если оба объекта имеют одного владельца, то разрешения для ссылаемого объекта не проверяются.

Пример цепочки владения

В следующей иллюстрации владелец представления **July2003** — Мэри. Она предоставила Алексу разрешения для доступа к представлению. Он не имеет других разрешений для объектов базы данных в этом экземпляре. Что произойдет, если Алекс выберет представление?


Экземпляр
Мария
(имя входа)
1
Алексей
(пользователь)
База данных 1
July2003
Владелец — Мария
6
База данных 2
ProjectionsXZ
Владелец — Мария
2
SalesXZ
Владелец — Мария
3
InvoicesXZ
Владелец — Мария
4
AcctAgeXZ
Владелец — Сергей
5
ExpenseXZ
Владелец — Иван
Обозначения
Представление
Таблица

1. Алекс выполняет инструкцию SELECT * в представлении **July2003**. SQL Server проверяет разрешения в представлении и подтверждает, что Алекс имеет разрешение выбирать.
2. Представление **July2003** требует данные из представления **SalesXZ**. SQL Server проверяет владение **SalesXZ**. Владелец этого представления (**Мэри**) 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 (**Мэри**). 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 (**Сэм**, а не **Мэри**), поэтому должны быть получены полные сведения о разрешениях на это представление. Если представление **AcctAgeXZ** имеет разрешения, которые обеспечивают доступ со стороны пользователя **Алекс**, 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 (**Джо**, а не **Сэм**), поэтому должны быть получены полные сведения о разрешениях на эту таблицу. Если таблица **ExpenseXZ** имеет разрешения, которые обеспечивают доступ со стороны пользователя **Алекс**, сведения возвращаются.
6. Если представление **July2003** пытается получить данные из таблицы **ProjectionsXZ**, то сервер сначала проверяет наличие цепочечных связей между БД **Database 1** и **Database 2**. Если цепочечные связи между БД активны, то сервер проверяет владение для таблицы **ProjectionsXZ**. Владелец этой таблицы такой же, как у вызывающего представления (**Мэри**), поэтому разрешения для этой таблицы не проверяются. Возвращаются необходимые данные.

Межбазовые цепочки владения

SQL Server можно настроить так, чтобы разрешить цепочку владения между конкретными базами данных или между всеми базами данных внутри одного экземпляра SQL Server. Межбазовые цепочки владения по умолчанию отключены и они не должны активизироваться без специального запроса.

Потенциальные опасности

Цепочка владения очень полезна при управлении разрешениями в базе данных, но при ее использовании предполагается, что владельцы объектов предвидят все последствия каждого решения о предоставлении разрешений для защищаемого объекта. В предыдущем примере Мэри владеет большинством базовых объектов в представлении **July 2003**. Мэри имеет право сделать принадлежащие ей объекты доступными для любого другого пользователя, поэтому SQL Server ведет себя так, будто каждый раз, предоставляя доступ к первому представлению в цепочке, Мэри принимает сознательное решение о совместном доступе к представлениям и таблицам, на которые оно ссылается.

На практике такое предположение не всегда верно. Производственные базы данных гораздо сложнее, чем база данных-образец, и разрешения, которые определяют доступ к ним, редко в точности соответствуют административным структурам организации.

Следует отдавать себе отчет, что члены ролей базы данных с большими правами доступа могут использовать межбазовые цепочки владения для доступа к объектам в базах данных, внешних по отношению к их собственной БД. Например, если включена межбазовая цепочки владения между базой данных **A** и базой данных **B**, то член предопределенной роли базы данных **db_owner** любой из этих баз данных может незаконно проникнуть в другую базу данных. Процедура проста: Диана (член предопределенной роли базы данных **db_owner** в базе данных **A**) создает пользователя **Стюарт** в базе данных **A**. **Стюарт** уже существует как пользователь в базе данных **B**. Затем Диана создает объект (владелец которого **Стюарт**) в базе данных **A**, который вызывает любой объект, принадлежащий пользователю **Стюарт** в базе данных **B**. Вызывающий и вызываемый объекты имеют общего владельца, поэтому разрешения для объекта в базе данных **B** не будут проверяться, когда Диана обратится к ним через созданный ею объект.

Шифрование SQL Server

Шифрование представляет собой способ скрытия данных с помощью ключа или пароля. Это делает данные бесполезными без соответствующего ключа или пароля для дешифрования. Шифрование не решает проблемы управления доступом. Однако оно повышает защиту за счет ограничения потери данных даже при обходе системы управления доступом. Например, если компьютер, на котором установлена база данных, был настроен неправильно и злоумышленник смог получить конфиденциальные данные, то украденная информация будет бесполезна, если она была предварительно зашифрована.

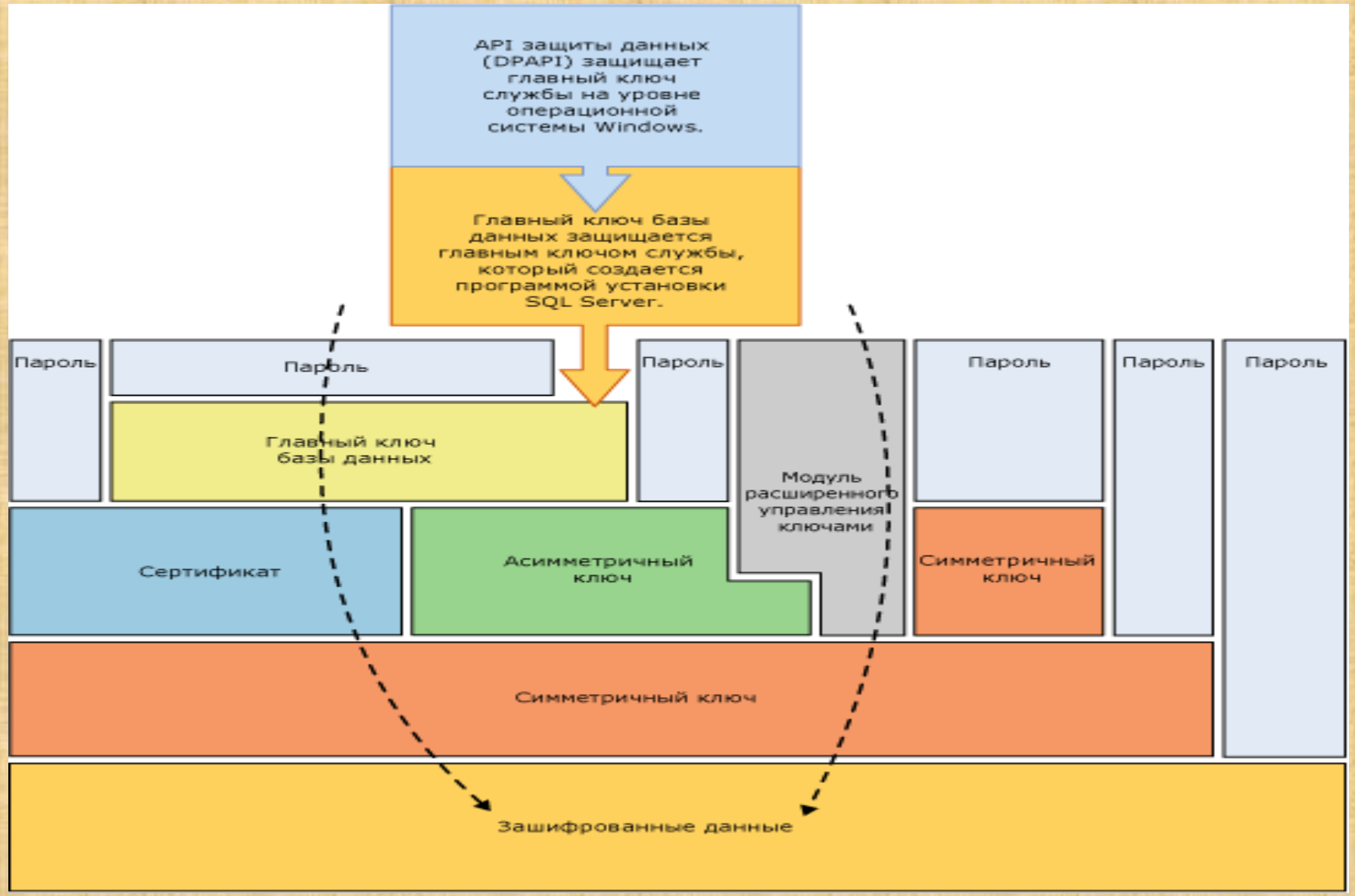
В SQL Server можно шифровать соединения, данные и хранимые процедуры.

При определении необходимости и способов шифрования следует рассмотреть следующие разделы:

- Иерархия средств шифрования
- Средства проверки подлинности
- Выбор алгоритма шифрования
- Основные сведения о прозрачном шифровании данных (TDE)
- Ключи шифрования базы данных и SQL Server

SQL Server шифрует данные, используя иерархическую структуру средств шифрования и управления ключами. На каждом уровне данные низшего уровня шифруются на основе комбинации сертификатов, асимметричных ключей и симметричных ключей. Асимметричные и симметричные ключи можно хранить вне модуля расширенного управления ключами SQL Server.

На следующем рисунке показано, что на каждом уровне иерархии средств шифрования шифруются данные более нижнего уровня, и отображаются наиболее распространенные конфигурации шифрования. Доступ к началу иерархии, как правило, защищается паролем.


API защиты данных (DPAPI) защищает главный ключ службы на уровне операционной системы Windows.
Главный ключ базы данных защищается главным ключом службы, который создается программой установки SQL Server.
Пароль
Пароль
Пароль
Пароль
Пароль
Пароль
Главный ключ базы данных
Модуль расширенного управления ключами
Сертификат
Асимметричный ключ
Симметричный ключ
Симметричный ключ
Зашифрованные данные

Следует учитывать следующие основные понятия.

- Для лучшей производительности данные следует шифровать с помощью симметричных ключей, а не с помощью сертификатов и асимметричных ключей.
- Главные ключи базы данных защищены главным ключом службы. Главный ключ службы создается при установке SQL Server и шифруется API-интерфейсом защиты данных Windows (DPAPI).
- Возможны другие иерархии шифрования с дополнительными уровнями.
- Модуль расширенного управления ключами хранит симметричные или асимметричные ключи вне SQL Server.
- Прозрачное шифрование данных (TDE) должно использовать симметричный ключ, который называется ключом шифрования базы данных, защищенный сертификатом, который, в свою очередь защищается главным ключом базы данных master или асимметричным ключом, хранящимся в модуле расширенного управления ключами.
- Главный ключ службы и все главные ключи базы данных являются симметричными ключами.

Механизмы шифрования

SQL Server поддерживает следующие механизмы шифрования:

- Transact-SQL функции
- асимметричные ключи;
- симметричные ключи.
- сертификаты;
- Прозрачное шифрование данных

Функции Transact-SQL

Отдельные элементы можно шифровать по мере того, как они вставляются или обновляются, с помощью функций Transact-SQL (ENCRYPTBYPASSPHRASE и DECRYPTBYPASSPHRASE).

Асимметричные ключи

Асимметричный ключ состоит из закрытого ключа и соответствующего открытого ключа. Каждый из этих ключей позволяет дешифровать данные, зашифрованные другим ключом. На выполнение асимметричных операций шифрования и дешифрования требуется сравнительно много ресурсов, но они обеспечивают более надежную защиту, чем симметричное шифрование. Асимметричный ключ можно использовать для шифрования симметричного ключа перед его сохранением в базе данных.

Симметричные ключи

Симметричный ключ — это ключ, используемый и для шифрования, и для дешифрования данных. Данные при использовании симметричного ключа шифруются и дешифруются быстро, и он вполне подходит для повседневной защиты конфиденциальных данных, хранящихся в базе данных.

Прозрачное шифрование данных

Прозрачное шифрование данных (TDE) является особым случаем шифрования с использованием симметричного ключа. TDE шифрует всю базу данных, используя симметричный ключ, который называется ключом шифрования базы данных. Ключ шифрования базы данных защищен другими ключами или сертификатами, которые, в свою очередь защищаются главным ключом базы данных или асимметричным ключом, хранящимся в модуле расширенного управления ключами.

Сертификаты

Сертификат открытого ключа, или просто сертификат, представляет собой подписанную цифровой подписью инструкцию, которая связывает значение открытого ключа с идентификатором пользователя, устройства или службы, имеющей соответствующий закрытый ключ. Сертификаты поставляются и подписываются центром сертификации (certification authority, CA). Сущность, получающая сертификат от центра сертификации, является субъектом этого сертификата. Как правило, сертификаты содержат следующие сведения.

- Открытый ключ субъекта.
- Идентификационные данные субъекта, например имя и адрес электронной почты.
- Срок действия, то есть интервал времени, на протяжении которого сертификат будет считаться действительным.

Сертификат действителен только в течение указанного в нем периода, который задается в каждом сертификате при помощи дат (**Valid From** и **Valid To**), определяющих начало и окончание срока действия.

- Идентификационные данные поставщика сертификата.
- Цифровая подпись поставщика.

Эта подпись подтверждает действительность связи между открытым ключом и идентификационными данными субъекта

Главное преимущество сертификатов в том, что они позволяют не хранить на узлах совокупность паролей отдельных субъектов. Вместо этого узел просто устанавливает доверительные отношения с поставщиком сертификата; после этого поставщик может подписать неограниченное количество сертификатов.

Когда узел (например, защищенный веб-сервер) указывает, что конкретный поставщик сертификата является доверенным корневым центром сертификации, он неявно выражает доверие к политикам, которые поставщик использовал при определении связей для изданных им сертификатов. Таким образом, узел выражает уверенность в том, что поставщик проверил идентификационные данные субъекта сертификата. Узел делает поставщика доверенным корневым центром сертификации, помещая самостоятельно подписанный сертификат поставщика, содержащий открытый ключ поставщика, в хранилище сертификатов доверенного корневого центра сертификации на компьютере сервера. Промежуточные или подчиненные центры сертификации являются доверенными только в том случае, если к ним ведет правильный путь от доверенного корневого центра сертификации.

Поставщик может отозвать сертификат до истечения срока его действия. При этом отменяется связь открытого ключа с идентификационными данными, указанными в сертификате. Каждый поставщик ведет список отозванных сертификатов, который можно использовать для проверки конкретного сертификата.

Самостоятельно подписанные сертификаты, созданные SQL Server, соответствуют стандарту X.509 и поддерживают поля X.509 v1.

Основные сведения о прозрачном шифровании данных (TDE)

Чтобы защитить базу данных, можно принять ряд мер предосторожности, например, спроектировать систему безопасности, проводить шифрование конфиденциальных ресурсов и поместить серверы базы данных под защиту брандмауэра. Однако в случае с похищением физического носителя (например, диска или ленты) злоумышленник может просто восстановить или подключить базу данных и получить доступ к данным. Одним из решений может стать шифрование конфиденциальных данных в базе данных и защита ключей, используемых при шифровании, с помощью сертификата. Это не позволит ни одному человеку, не обладающему ключами, использовать данные, однако такой тип защиты следует планировать заранее.

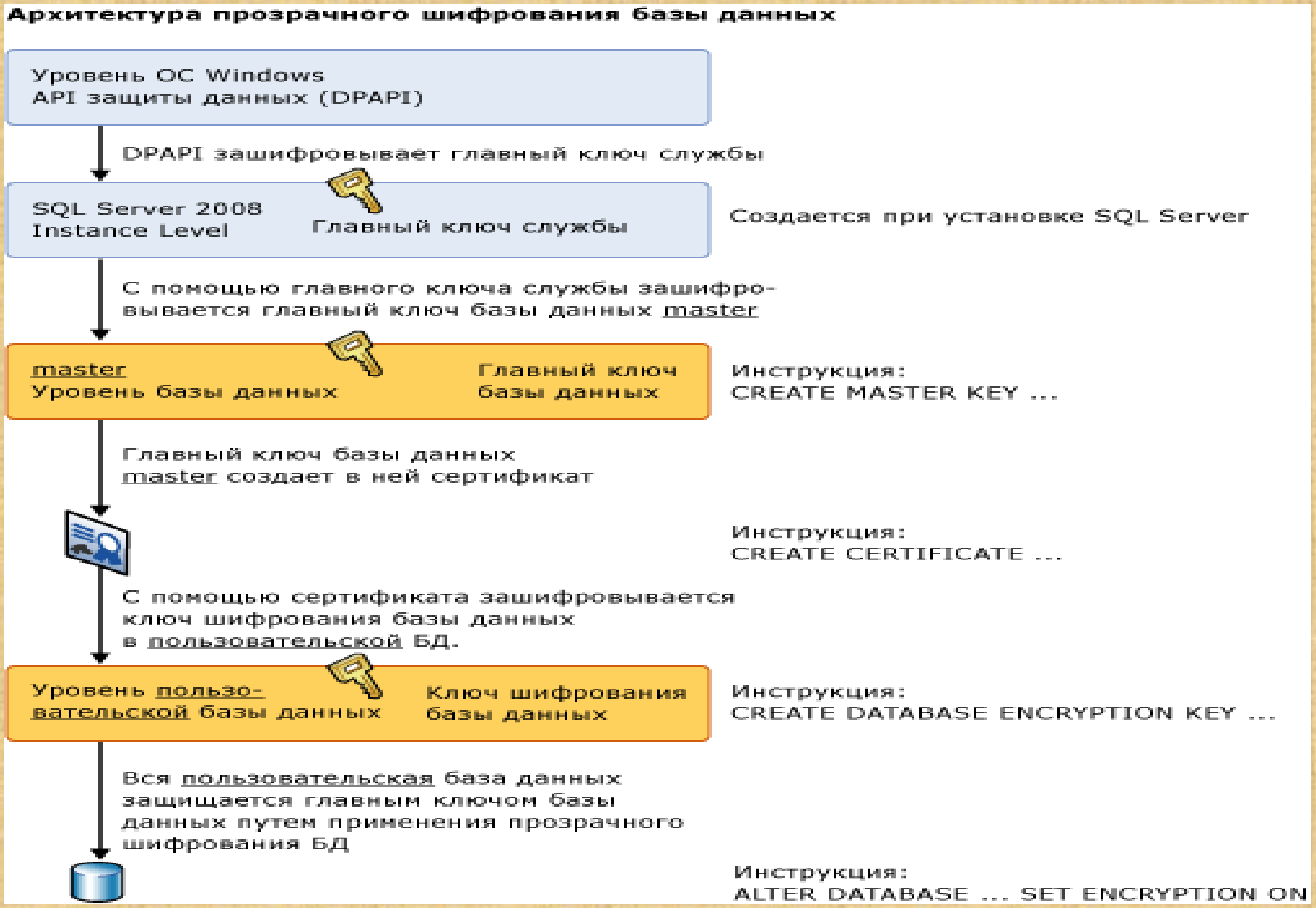
Функция *прозрачного шифрования данных* (TDE) выполняет в реальном времени шифрование и дешифрование файлов данных и журналов в операциях ввода-вывода. При шифровании используется ключ шифрования базы данных (DEK), который хранится в загрузочной записи базы данных для доступности при восстановлении. Ключ шифрования базы данных является симметричным ключом, защищенным сертификатом, который хранится в базе данных master на сервере, или асимметричным ключом, защищенным модулем расширенного управления ключами. Функция прозрачного шифрования данных защищает «неактивные» данные, то есть файлы данных и журналов. Она дает возможность обеспечивать соответствие требованиям многих законов, постановлений и рекомендаций, действующих в разных отраслях. Это позволяет разработчикам программного обеспечения шифровать данные с помощью алгоритмов шифрования AES и 3DES, не меняя существующие приложения.

При включении функции прозрачного шифрования данных необходимо немедленно создать резервную копию сертификата и закрытого ключа, связанного с этим сертификатом. Если сертификат когда-либо станет недоступным или придется восстанавливать или присоединять базу данных на другом сервере, для открытия базы данных необходимо иметь резервные копии как сертификата, так и закрытого ключа. Сертификат шифрования и ассиметричный ключ следует сохранить, даже если функция прозрачного шифрования в базе данных будет отключена. Даже в случае отсутствия шифрования в базе данных ключ шифрования необходимо хранить в базе данных, поскольку он может понадобиться для некоторых операций.

Шифрование файла базы данных проводится на уровне страниц. Страницы в зашифрованной базе данных шифруются до записи на диск и дешифруются при чтении в память. Прозрачное шифрование данных не увеличивает размер зашифрованной базы данных.

На следующем рисунке показана архитектура прозрачного шифрования данных.

Архитектура прозрачного шифрования базы данных



Файлы резервных копий баз данных, в которых включено TDE, также шифруются с помощью ключа шифрования базы данных. Поэтому для восстановления таких резервных копий необходимо иметь сертификат, защищающий ключ шифрования базы данных. Это значит, что помимо резервного копирования базы данных обязательно необходимо сохранять резервные копии сертификатов серверов, чтобы не допустить потери данных. Если сертификат станет недоступным, это приведет к потере данных.

В режиме TDE все файлы и файловые группы зашифрованы. Если какие-либо файловые группы в базе данных помечены атрибутом READ ONLY, то операция шифрования базы данных завершится сбоем.

Если база данных используется в зеркальном отображении базы данных или в доставке журналов, то зашифрованы будут обе базы данных. Транзакции журналов при передаче между ними будут передаваться в зашифрованном виде.

Все новые полнотекстовые индексы будут шифроваться после включения шифрования базы данных. Ранее созданные полнотекстовые индексы будут импортированы во время обновления и станут доступны для TDE после загрузки данных в SQL Server. Включение полнотекстового индекса в столбце может привести к тому, что во время полнотекстового индексирования данные из этого столбца будут записываться на диск в виде обычного текста. Не рекомендуется создавать полнотекстовый индекс на важных зашифрованных данных.

Зашифрованные данные подвергаются сжатию в значительно меньшей степени, чем незашифрованные. Если TDE используется для шифрования базы данных, сжатие резервной копии не сможет значительно сжать хранилище резервных копий. Поэтому не рекомендуется совместное использование TDE и сжатия резервных копий.

Кластеризация

Кластером называется группа компьютеров, которые обеспечивают друг для друга взаимное резервное копирование на случай отказов.

Обзор MSCS

MSCS является встроенной службой Windows 2000 Advanced Server, Windows 2000 Datacenter Server и Windows NT Enterprise Edition. MSCS применяется для формирования кластера серверов, который, как уже говорилось, является группой независимых серверов, работающих совместно как единая система. Кластер служит для обеспечения готовности (availability, этот термин может переводиться и как "доступность") клиентов к обслуживанию приложений в ситуации возникновения отказа или при запланированных отключениях. Если один из серверов кластера по какой-либо причине является недоступным, то ресурсы и приложения перемещаются на другой узел кластера.

Системы, исполняющие MSCS, обеспечивают высокую готовность и имеют много других достоинств. Некоторые из достоинств применения MSCS перечислены ниже.

- **Высокая готовность.** Системные ресурсы, такие как дисковые накопители и IP-адреса, автоматически передаются от отказавшего сервера к выжившему. Это явление называется переход по отказу (failover). При возникновении ситуации, когда приложение на кластере отказывает, MSCS автоматически запускает его на выжившем сервере или распределяет работу отказавшего сервера по другим оставшимся узлам кластера. Переход по отказу происходит быстро, поэтому для пользователей он представится как лишь мгновенная заминка в обслуживании.
- **Возврат к исходному узлу кластера после восстановления.** После того как отказавший сервер будет починен и введен в строй, MSCS автоматически перераспределяет нагрузку на кластере. Это явление называется возврат к исходному узлу кластера (failback).

- **Управляемость.** При помощи программного обеспечения Cluster Administrator можно управлять всем кластером как единой системой. Вы можете легко перемещать приложения на те или иные серверы в пределах кластера, перетаскивая объекты внутри Cluster Administrator. Точно так же можно перемещать и данные. При помощи этого перетаскивания можно производить ручное балансирование нагрузок на серверы, можно также снять нагрузку с какого-либо сервера, подготовив тем самым его к плановому отключению и техническому обслуживанию. При помощи Cluster Administrator можно также следить из любого места в сети за состоянием кластера и каждого из его узлов, а также за любыми доступными ресурсами.
- **Масштабируемость.** По мере роста требований к системе, MSCS может быть переконфигурирована для поддержки этого роста. Если суммарная нагрузка станет превышать возможности кластера, можно будет добавить в кластер дополнительные узлы.

Основные понятия

MSCS сокращает длительность простоев, осуществляя переходы по отказу между отдельными компьютерами, применяя при этом взаимосвязь между серверами и дисковую систему с общим доступом (рис. 2). В качестве взаимосвязи между серверами может применяться любое высокоскоростное соединение, например, сеть Ethernet или другое сетевое оборудование. Эта взаимосвязь функционирует как канал коммуникации между серверами, благодаря которому возможна двусторонняя передача информации о состоянии кластера и о конфигурации. Благодаря разделяемой дисковой системе возможен равноправный доступ всех серверов кластера к базе данных и к другим файлам с данными. Такая разделяемая дисковая система может быть реализована при помощи SCSI, SCSI поверх Fibre Channel, а также при помощи какого-либо нестандартного оборудования. Разделяемые диски могут быть как одиночными дисками, так и RAID-системой.

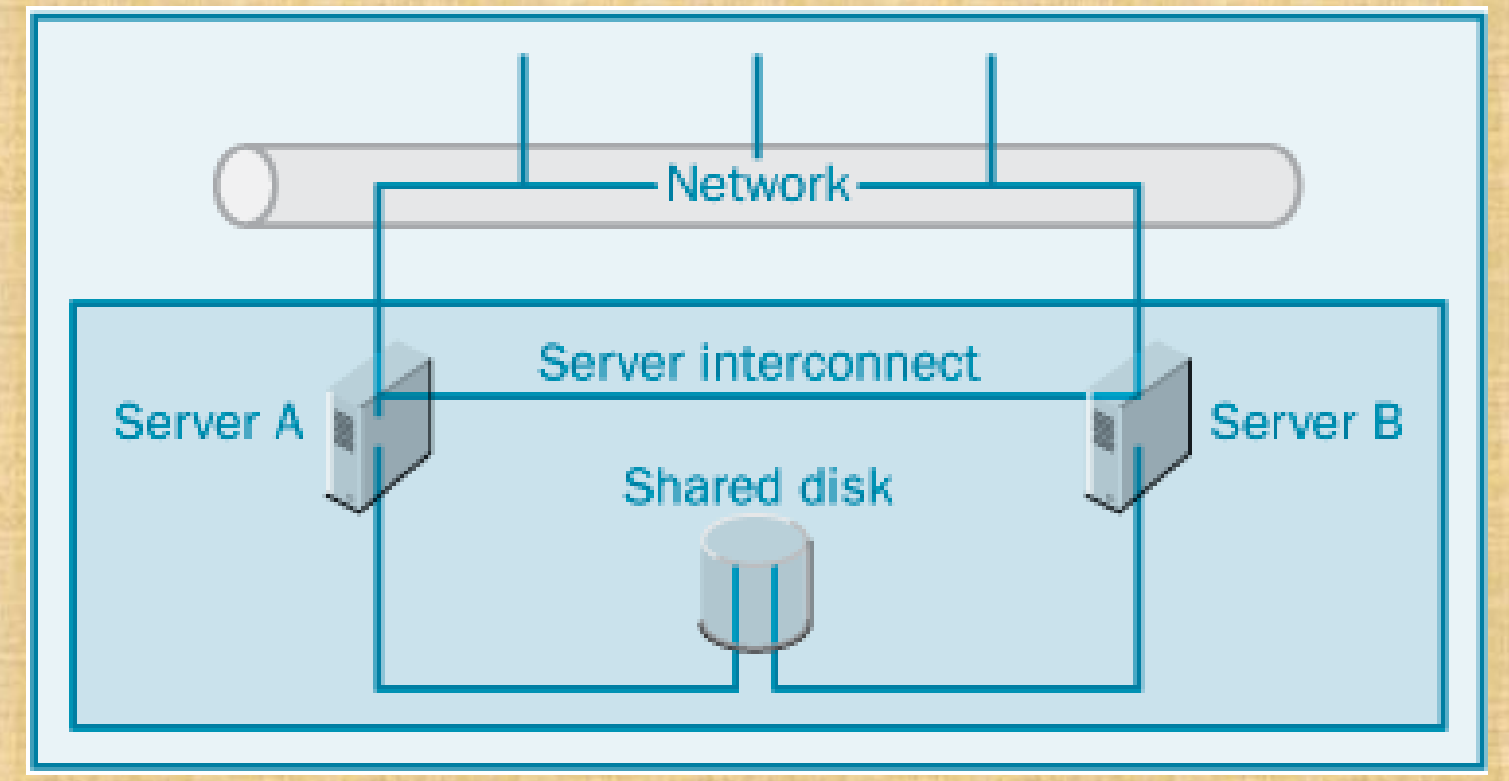


Рис.2 Кластер Windows

Как только система будет сконфигурирована как сервер кластера, она будет преобразована из обычного сервера в так называемый виртуальный сервер. Виртуальный сервер похож на обычный сервер, но для него применяется абстрагирование и отказ от настоящей физической сущности компьютера. Так как аппаратная часть компьютера, образующая виртуальный сервер, может со временем меняться, то пользователь не будет знать, на каком именно сервере выполняется его приложение в данный момент времени. Поэтому пользовательские приложения выполняются не на конкретном комплекте оборудования, а на виртуальном сервере.

Виртуальный сервер существует как элемент сети и ему сопоставлен IP-адрес, применяемый в протоколе TCP/IP. Этот IP-адрес может передаваться от одного компьютера к другому, благодаря чему пользователи продолжают "видеть" виртуальный сервер независимо от того, на каком именно оборудовании он работает. На самом деле, этот IP-адрес переходит от одного компьютера к другому, что обеспечивает одинаковое представление виртуального сервера для наружного наблюдателя. Приложение, направленное по некоторому адресу, все равно получит доступ, соответствующий этому адресу, даже если конкретный сервер, соответствующий этому адресу, и откажет (однако этот адрес теперь будет соответствовать другому серверу). Виртуальный сервер прячет от пользователя операции перехода по отказу, поэтому пользователь может продолжать свою работу, не зная о событиях, происходящих "за кулисами".

Компоненты кластера

Для создания кластера нужны некоторые компоненты: программное обеспечение для управления кластером, взаимосвязь между серверами и разделяемая дисковая система. Чтобы образовать кластер, эти компоненты должны быть сконфигурированы согласованно с приложениями, которые будут предназначены для работы на нем.

Программное обеспечение MSCS для управления кластером

Программное обеспечение для управления кластером – это совокупность инструментальных средств, применяемых для технического обслуживания, конфигурирования и работы кластера. Оно содержит следующие подкомпоненты, работающие совместно и выполняющие, при необходимости, переход по отказу:

- **Менеджер узлов (Node Manager).** Поддерживает членство в кластере и передает "пульс" (heartbeats) членам кластера (узлам). (Этот пульс представляет собой просто периодически отсылаемые сообщения, означающие "Я жив".) Если пульс от некоторого узла прекращается, то другой узел делает вывод, что этот узел перестал функционировать, и предпринимает шаги по приему на себя его функций. Менеджер узлов является одним из наиболее критичных элементов кластера, потому что он следит за состоянием кластера и решает, какие действия должны быть предприняты.
- **Менеджер базы данных конфигурации (Configuration Database Manager).** Поддерживает базу данных конфигурации кластера, в которой хранятся сведения обо всех компонентах кластера, как об абстрактных логических элементах (например, виртуальных серверах), так и физических элементах (например, разделяемых дисках). Эта база данных подобна системному реестру Windows NT/Windows 2000.

- **Менеджер ресурсов / Менеджер переходов по отказу (Resource Manager / Failover Manager).** Запускает и останавливает службу MSCS. Информацию (например, о потере узла, о добавлении узла и т.д.) Менеджер ресурсов / Менеджер переходов по отказу получает от Монитора ресурсов и Менеджера узлов.
- **Обработчик событий (Event Processor).** Инициализирует кластер и осуществляет маршрутизацию информации о событиях (routes event information) среди компонент кластера. Обработчик событий также инициирует расширение кластера, давая указание Менеджеру узлов о добавлении узла.
- **Менеджер коммуникаций (Communications Manager).** Управляет коммуникацией между узлами кластера. Все узлы кластера, для обеспечения своей правильной работы, должны постоянно осуществлять коммуникацию друг с другом. Если этой коммуникации между узлами не будет, то информация о состоянии кластера будет потеряна и кластер не сможет функционировать.
- **Менеджер глобального обновления (Global Update Manager).** Передает информацию о состоянии кластера (например, информацию о добавлении узлов в кластер, об удалении узлов и т.д.) всем узлам кластера.

- **Монитор ресурсов (Resource Monitor).** Отслеживает состояние различных ресурсов кластера и сообщает статистические данные. Эта информация может применяться для принятия решений о необходимости выполнения на кластере переходов по отказу.
- **Служба времени (Time Service).** Гарантирует, что все узлы кластера сообщают одинаковое системное время. Если бы Службы времени не было, то события могли бы представляться в неверной последовательности, что приводило бы к неверным решениям. Например, если бы один узел "думал" бы, что сейчас 2 часа дня и содержал бы старую копию файла, а другой узел "думал" бы, что сейчас 10 часов утра и содержал бы более новую версию этого файла, то кластер мог бы неправильно решить, что файл на первом узле является более свежим.

Взаимосвязь между серверами

Взаимосвязь между серверами – это просто соединение между узлами кластера. Так как для узлов кластера необходима постоянная коммуникация между ними (через Службу времени, Менеджер узлов и т.д.), то поддержка этой связи очень важна. Взаимосвязь между серверами должна быть надежным каналом коммуникации.

Во многих случаях в качестве взаимосвязи между серверами может применяться сеть Ethernet, на которой исполняется протокол TCP/IP или NetBIOS. Этого вполне достаточно, но вы можете захотеть применять нестандартную, высокоскоростную взаимосвязь между серверами, которая будет гораздо быстрее, чем Ethernet. Эти взаимосвязи можно приобрести у многих поставщиков оборудования, некоторые из них предоставляют как решения для разделяемых дисков, так и решения для коммуникации. Полный список одобренных устройств для взаимосвязи между серверами в списке совместимого оборудования на веб-сайте фирмы Microsoft по адресу http://www.microsoft.com/hcl/

- **Разделяемая дисковая система**
- Другая ключевая компонента, необходимая для создания кластера, – **разделяемая дисковая система**. Если к одной и той же дисковой системе могут иметь доступ многие компьютеры, то в случае отказа узла его работу может взять на себя другой узел. Разделяемая дисковая система должна предоставлять равноправный доступ многим компьютерам к одним и тем же дискам, т.е., каждый из компьютеров должен быть способен иметь доступ ко всем дискам. В нынешней версии MSCS в каждый момент времени только один компьютер может иметь доступ к дискам, но в будущих версиях станет возможен одновременный доступ к данным для многих компьютеров.
- В настоящее время доступно несколько видов разделяемых дисковых систем, разрабатываются и новые технологии для дисков. Дисковые подсистемы SCSI всегда поддерживали работу со многими инициаторами, когда можно иметь много контроллеров SCSI на одной и той же шине SCSI. Это делает SCSI идеальным выбором для применения в кластерах. Фактически, системы SCSI были первыми дисковыми подсистемами, которые могли применяться для кластеризации.

- Для поддержки кластеризации были разработаны также более современные дисковые технологии, такие как Fibre Channel и некоторые нестандартные фирменные решения. Системы Fibre Channel позволяют подключать диски, находящиеся на значительном удалении от компьютеров. Большинство систем Fibre Channel поддерживает применение многих контроллеров в одном кольце Fibre Channel. Для поддержки кластеризации были спроектированы или модифицированы некоторые RAID-контроллеры. Без модификаций или без внесения изменений в конфигурацию, большинство дисковых контроллеров не будет поддерживать кластеризацию.
- Вопрос о кэшах контроллеров, благодаря которым возможно кэширование записей в память, также надо решать при кластеризации, в случаях, когда кэш расположен на самом контроллере (рис. 2). В этом случае каждый узел имеет свой собственный кэш, и мы говорим, что он находится перед разделением дисков ("in front of" the disk sharing), потому что два кэша пользуются одними и теми же дисковыми накопителями.

Если каждый контроллер имеет кэш и кэш размещается на отказавшем компьютере, то данные из кэша могут быть потеряны. Из-за этого, когда в конфигурации кластера применяются внутренние кэши контроллера, они должны быть настроены как применяемые только для чтения (set as read-only). (При некоторых обстоятельствах такая настройка может уменьшить производительность некоторых систем.)

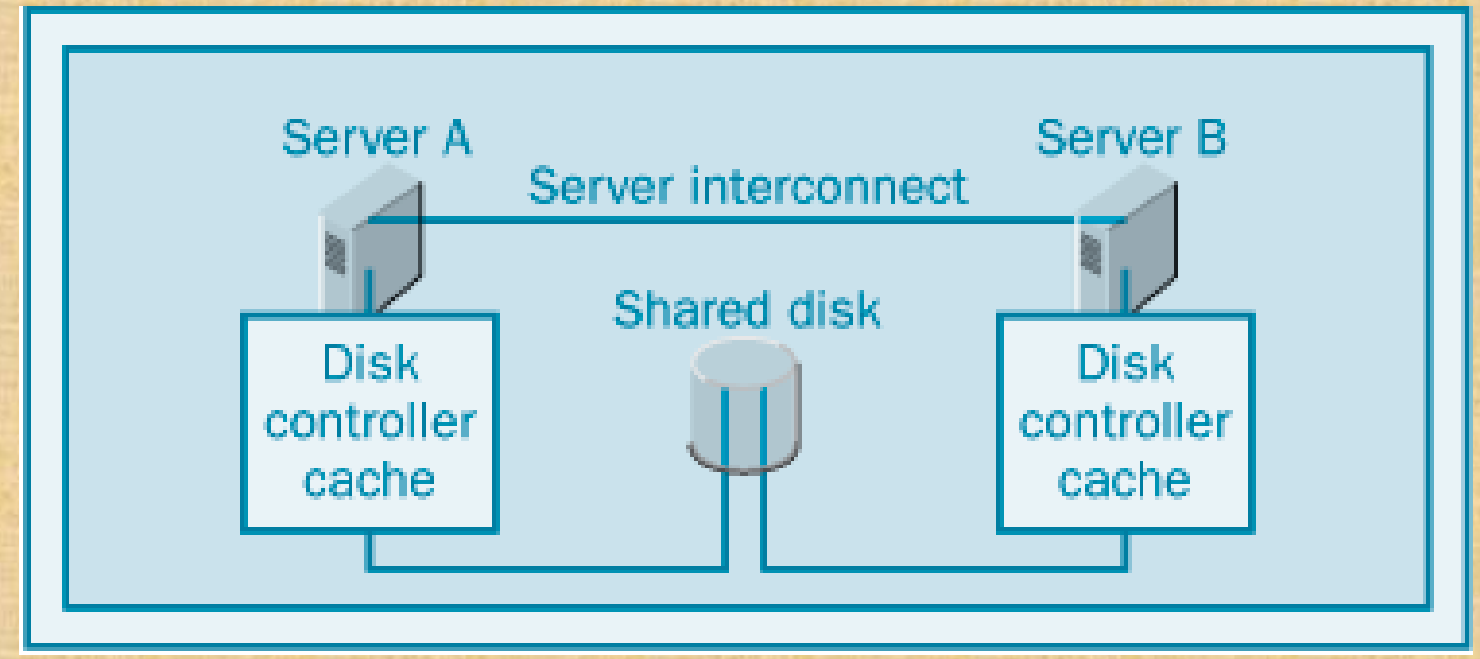


Рис. 2. Кэши контроллера перед разделением дисков

В других решениях этой проблемы с разделяемыми дисками предусматривается расслоение (чередование) RAID и кэширование внутри самой дисковой подсистемы. В этой конфигурации кэш разделяется (используется совместно) узлами и в данном случае мы говорим, что кэш находится *позади разделения дисков* (рис. 3). Теперь механизмы расслоения и кэш выглядят одинаково для всех контроллеров системы, и безопасными являются как чтение из кэша, так и запись в кэш.

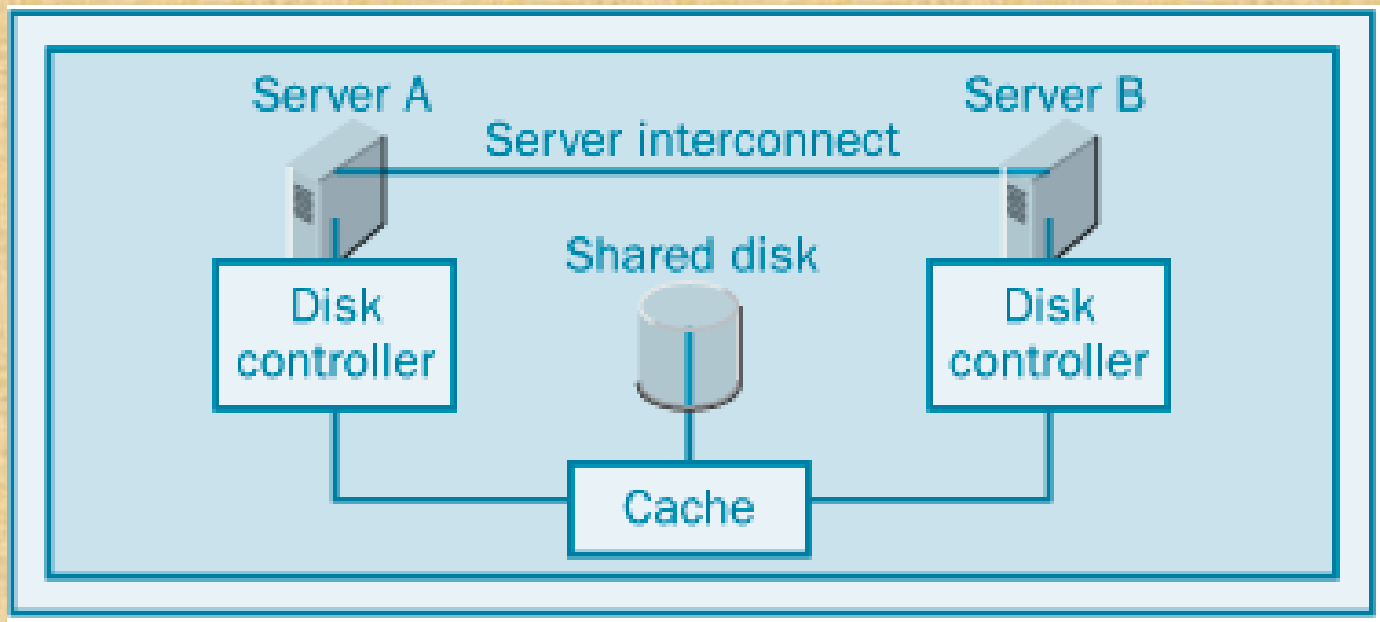
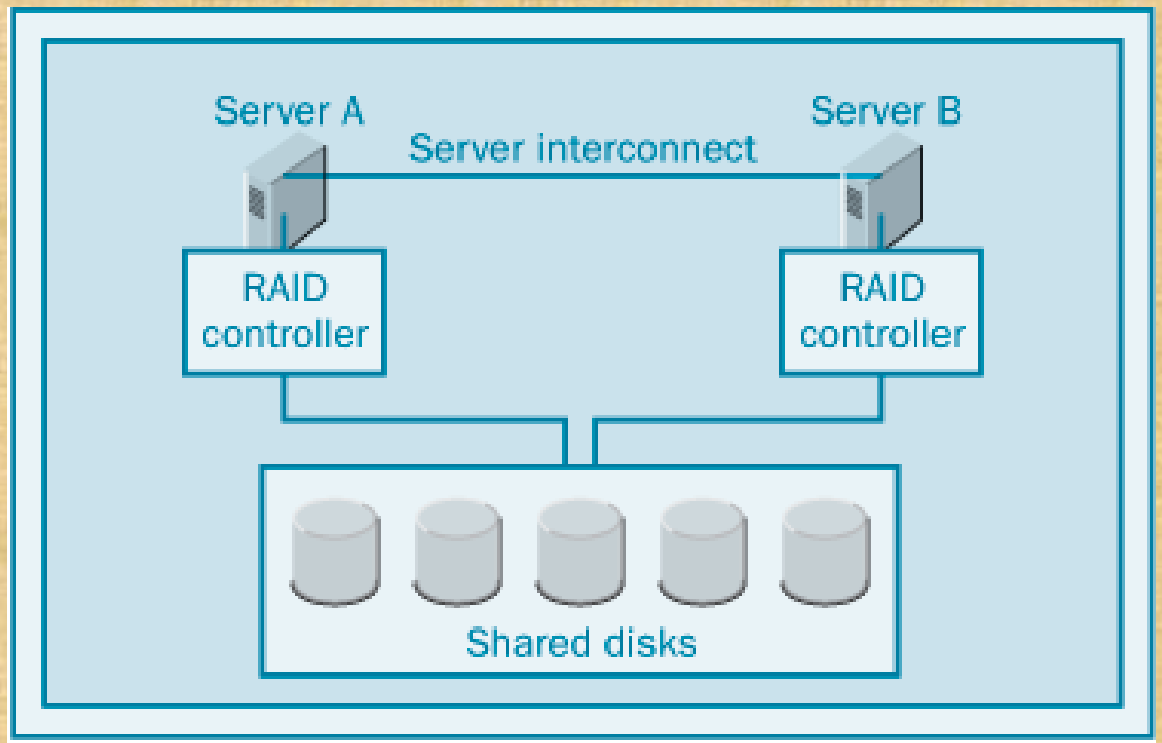


Рис. 3. Кэш контроллера позади разделения дисков

Последние дисковые подсистемы SCSI и Fibre Channel допускают размещение RAID- контроллеров в корпусах дисковых систем, а не в корпусах компьютеров. Такие системы обеспечивают хорошую производительность и отказоустойчивость.

Внутренняя RAID-система. Внутренние RAID-контроллеры спроектированы таким образом, что аппаратура, управляющая работой RAID, и кэш находятся на самом компьютере. Когда используется внутренняя RAID-система, разделяемая дисковая система разделяется до расслоения RAID, как показано на рис. 4.



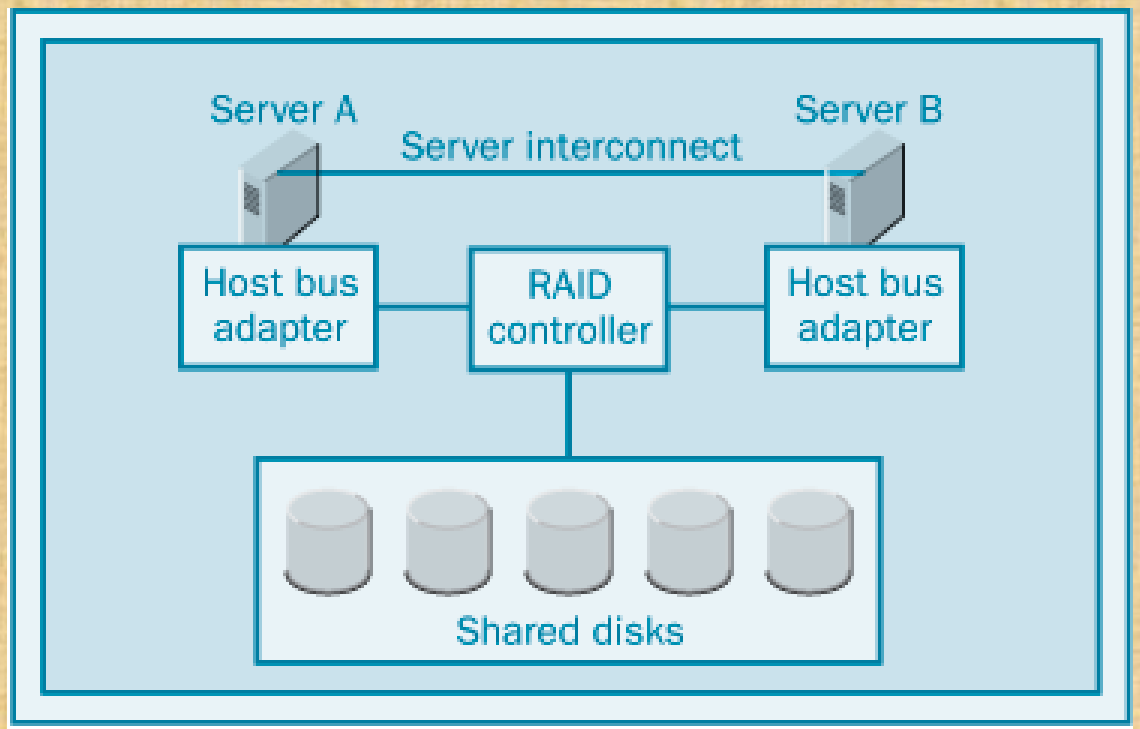
Так как кэш размещен на контроллере, который не находится в совместном использовании, все данные, которые были в кэше в момент отказа системы, станут недоступными. Это большая проблема в случае, когда работа идет с реляционными СУБД. Когда SQL Server записывает данные на диск, эти данные заносятся в журнал транзакций как записанные. Когда SQL Server пытается восстановиться после отказа системы, эти блоки данных не будут восстановлены, потому что SQL Server считает, что они уже были записаны на диск. В случае отказа при этом типе конфигурации база данных будет повреждена.

Поэтому поставщики RAID-контроллеров с кэшированием при работе их в составе кластеров сертифицируют их работу только при отключенном кэше (или, по крайней мере, при запрете записи в кэш). Если кэш был отключен, то SQL Server не получит сигнала о том, что операция записи была завершена, до тех пор, пока данные не будут действительно записаны на диск.

В некоторых ситуациях, если пользоваться кэшем контроллера можно добиться повышения производительности. Это особенно имеет смысл при использовании конфигураций RAID 10 и RAID 5, потому что для этих уровней RAID очень велика дополнительная нагрузка, появляющаяся при записи.

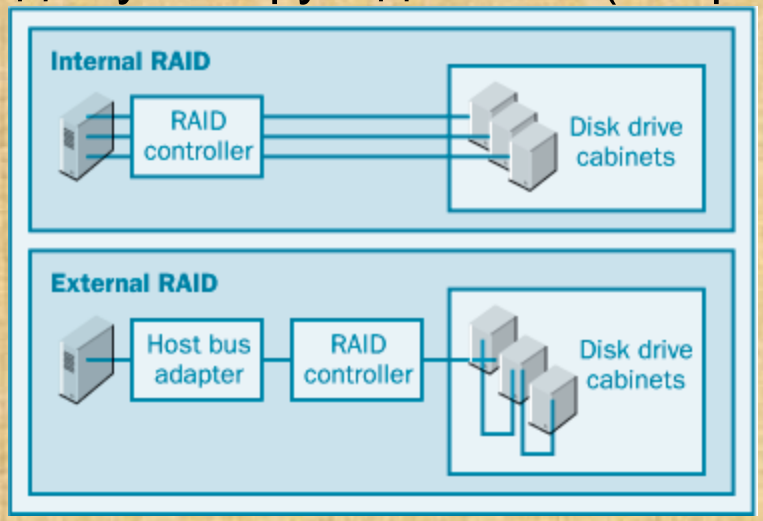
Чтобы пользоваться кэшем записи контроллера в кластерной конфигурации, нужно пользоваться внешней RAID-системой, в которой кэш находится в совместном пользовании и данные при переходе по отказу не теряются.

Внешняя RAID-система. Во внешних RAID-системах аппаратура RAID находится за пределами компьютеров (см. рис. 5). Каждый сервер содержит адаптер главной шины (HBA, host bus adapter), задачей которого является передача в RAID-систему максимально большего количества запросов ввода-вывода с наивысшей возможной скоростью. А место фактического размещения данных определяется RAID-системой.



Внешние RAID-системы иногда называют "RAID в шкафу" или "RAID в ящике", потому что расслоение RAID производится внутри корпуса с дисками. Такие внешние подсистемы RAID обладают многими достоинствами. Они являются не только идеальным решением для MSCS, но и вообще, универсальным лучшим решением. Ниже перечислены достоинства систем "RAID в шкафу":

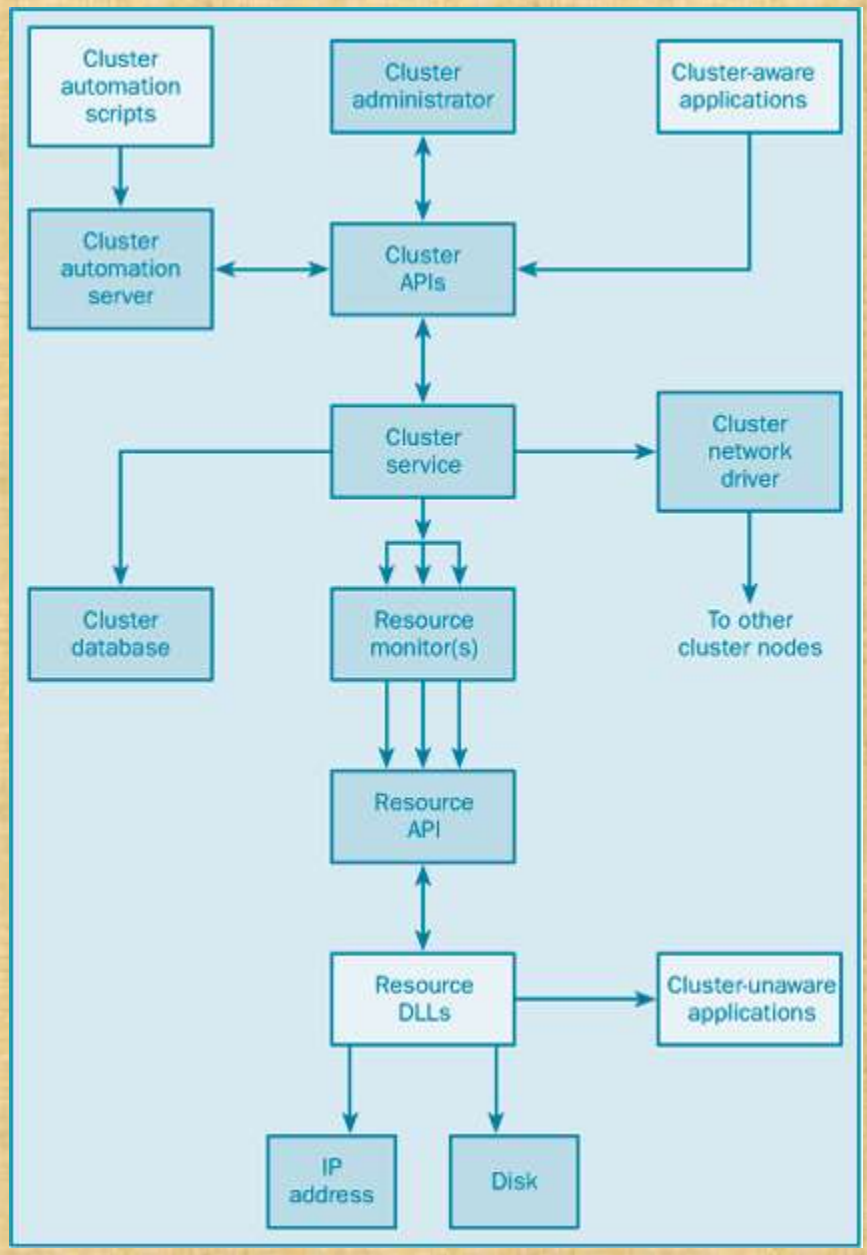
- **Удобство при подключении.** При использовании внутренних RAID-систем потребуются многочисленные кабели – по кабелю для каждого шкафа с дисками, выходящему из каждого RAID-контроллера. Для внешнего RAID понадобится проложить только один кабель от адаптера главной шины до RAID-контроллера и кабели от контроллера, формирующие соединение "гирляндой" (daisy chain) к каждому шкафу с дисками (см. рис. 6).



- **Обеспечивается избыточность RAID.** Многие внешние решения RAID разрешают одному контроллеру памяти осуществлять коммуникацию и с основным, и с вторичным RAID-контроллером, что обеспечивает полное резервирование и переход по отказу к другому узлу.
- **Обеспечивается кэширование в кластерах.** Применяя внешние решения RAID, кэширование можно выполнить гораздо более просто. При использовании внешних RAID можно разрешать применение как кэширования, так и средств для отказоустойчивости, не беспокоясь о согласованности работы кэшей из разных контроллеров (потому что имеется только один кэш и один контроллер). Фактически, при применении внешнего RAID, использование кэша записи является безопасным. При кэшировании данных от реляционных СУБД некоторый риск все же сохраняется, но при применении внешнего RAID он уменьшается. Обязательно проверьте, что поставщик ваших внешних RAID-систем поддерживает зеркальное дублирование кэшей. Зеркальное дублирование кэшей обеспечивает отказоустойчивость кэш-памяти в случае отказа ее микросхем.
- **Поддержка большого количества дисковых накопителей.** Для работы больших или высокопроизводительных систем иногда требуется конфигурировать дополнительные дисковые накопители

Категории приложений, работающих с кластерами

Приложения, которые работают на системах с MSCS можно разделить на следующие четыре категории. Эти категории приложений и их взаимодействие с MSCS показаны на рис. 7.



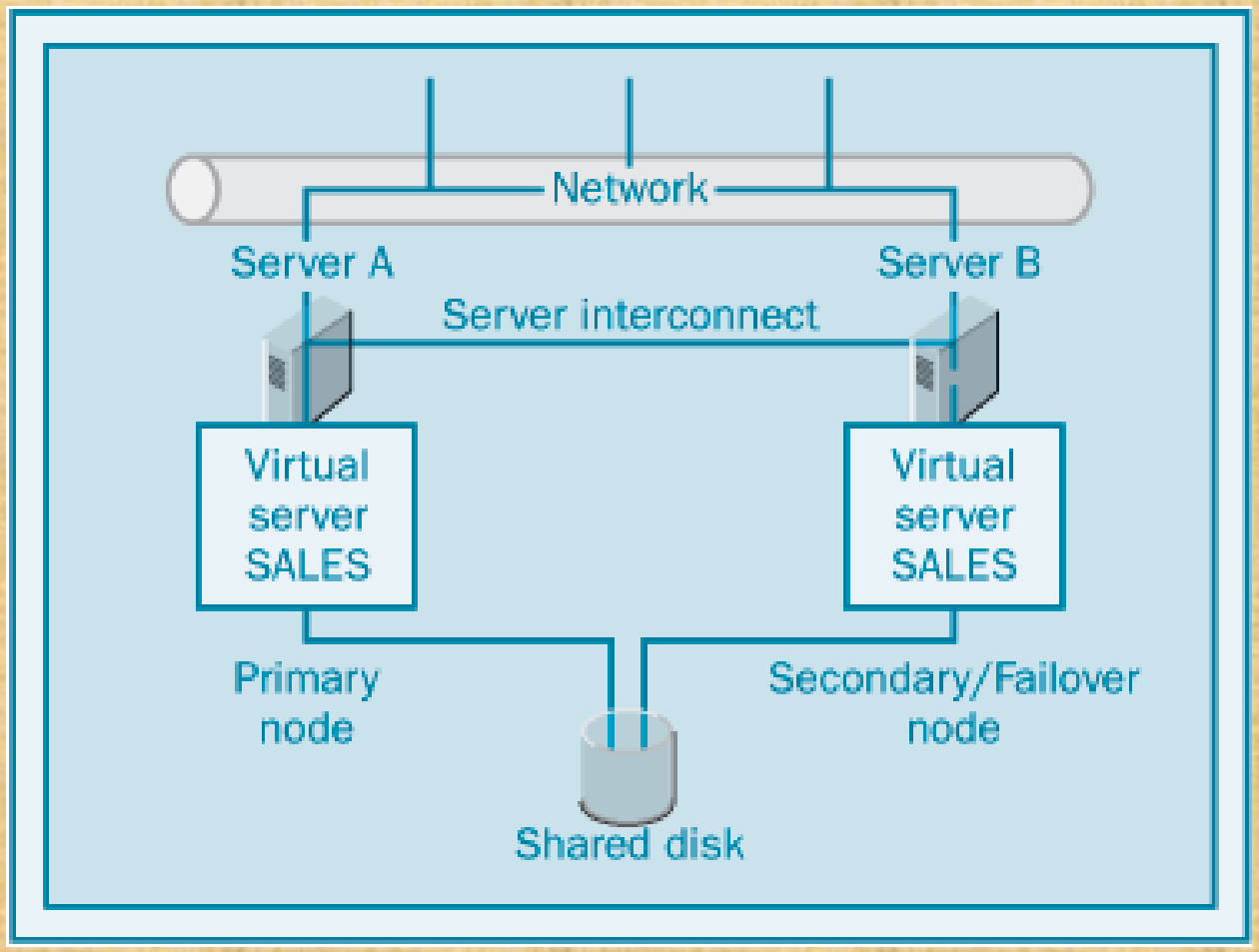
- **Приложения, не рассчитанные на кластеризацию.** Приложения этой категории никак не взаимодействуют с MSCS. Хотя при обычных условиях они могут работать нормально, но при отказах узлов они могут заработать неверно, что воспрепятствует их переходу на другой узел кластера.
- **Приложения, рассчитанные на кластеризацию.** Эти приложения рассчитаны на работу совместно с MSCS. Они могут воспользоваться достоинствами MSCS, повышающими производительность и масштабируемость. Они правильно реагируют на события, происходящие на кластере, и при отказе компонент и переходе на другой узел кластера, как правило, им не требуется уделять большого внимания (а иногда и вообще никаких забот не потребуется). SQL Server может служить примером таких приложений, рассчитанных на кластеризацию.
- **Приложения для управления кластером.** Приложения этой категории служат для наблюдения за кластером и для управления окружением MSCS.
- **Нестандартные типы ресурсов.** Эти приложения являются нестандартными ресурсами управления кластером для приложений, служб и устройств.

Режимы MSCS

Поддержку кластеров для SQL Server и MSCS можно запускать в различных режимах. В активно-пассивном режиме один сервер находится в состоянии ожидания, готовый принять на себя работу в случае отказа первичного сервера. В активно-активном режиме каждый сервер работает со своей базой данных SQL Server. В случае отказа любого из серверов другой сервер берет работу на себя, и дело кончается тем, что один сервер работает с двумя базами данных. В этом разделе мы рассмотрим преимущества и недостатки применения каждого из этих двух режимов.

Активно-пассивные кластеры

В активно-пассивных кластерах для работы приложений SQL Server применяется первичный узел (primary node), а сервер – вторичный узел (secondary node) является запасным, резервным сервером (рис. 8).



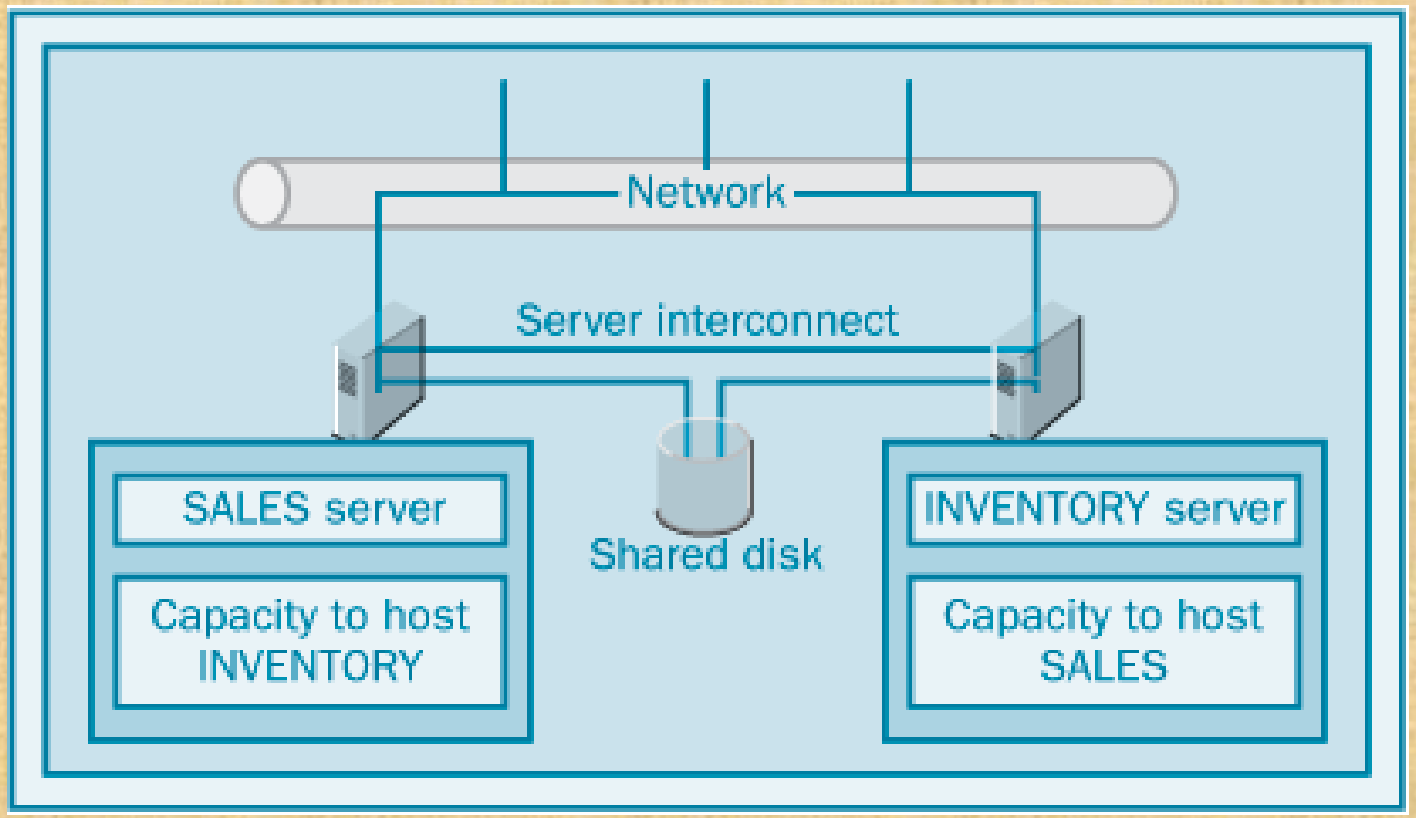
В этой конфигурации один из серверов практически не используется (может месяцами стоять без дела). Фактически, во многих случаях резервный сервер вообще никогда не будет использоваться. И тогда его могут расценивать как дорогостоящее оборудование, простаивающее без дела. Так как этот сервер не может выполнять другие задачи, то для удовлетворения нужд пользователей придется покупать другое оборудование, что делает активно-пассивный режим потенциально неэкономичным.

Но, несмотря на эту неэкономичность, активно-пассивный режим имеет свои достоинства. В этой конфигурации при отказе первичного узла все ресурсы вторичного узла полностью готовы к тому, чтобы взять на себя его работу. Это важно, когда выполняются критически важные приложения, для которых требуются заданные показатели производительности или времени отклика..

Рекомендуется, чтобы аппаратура вторичного узла была точно такая же, как и первичного узла (т.е., чтобы у первичного и вторичного узлов были бы одинаковые объемы оперативной памяти, одинаковые количество и типы центральных процессоров и т.д.). Если узлы имеют одинаковую аппаратуру, то вы можете быть уверены в том, что вторичная система будет работать почти с такой же скоростью, как и первичная.

Активно-активные кластеры

В активно-активных кластерах выполнять приложения могут оба сервера, причем каждый сервер служит вторичным сервером для другого узла (см. рис. 9).



Каждый из этих двух серверов работает сразу и как первичный узел для некоторых приложений, и как вторичный узел для приложений другого сервера. Эта конфигурация наиболее эффективна в экономическом плане, потому что никакое оборудование не простаивает, ожидая отказа другой системы. Обе системы активно обслуживают пользователей. Кроме того, один пассивный узел может служить вторичным узлом для нескольких первичных узлов.

Недостатком активно-активной конфигурации является лишь то, что в случае отказа производительность выжившего узла существенно снизится, потому что нагрузка на него вырастет. Выжившему узлу теперь придется исполнять не только приложения, которые работали на нем первоначально, но и приложения с первичного узла. Во многих случаях это снижение производительности окажется неприемлемым, и тогда придется применять активно-пассивную конфигурацию.

Примеры кластеризованных систем

В этом разделе мы рассмотрим четыре примера кластеризованных систем, применяющих MSCS. Эти примеры помогут вам решить, какой тип кластеров лучше всего соответствует вашим потребностям и окружению.

Пример 1 – система с высокой готовностью со статическим балансированием нагрузки

Такие системы обеспечивают высокую готовность для кластеров, на которых исполняется много приложений. Это, однако, достигается за счет некоторого падения производительности в случаях, когда на линии остается один узел. Такие системы обеспечивают максимум полезного использования аппаратных ресурсов, потому что доступен каждый из узлов. Конфигурация такого кластера показана на рис.10 (это – активно-активная конфигурация кластера).

Каждый из узлов этого кластера представляет для сети свой собственный набор ресурсов (в форме виртуального сервера) и сконфигурирован с некоторым избытком мощности, чтобы быть способным исполнять приложения другого узла в случае перехода по отказу. Готовность обслуживания клиентов отказавшего узла будет зависеть от имеющихся ресурсов и мощности сервера.

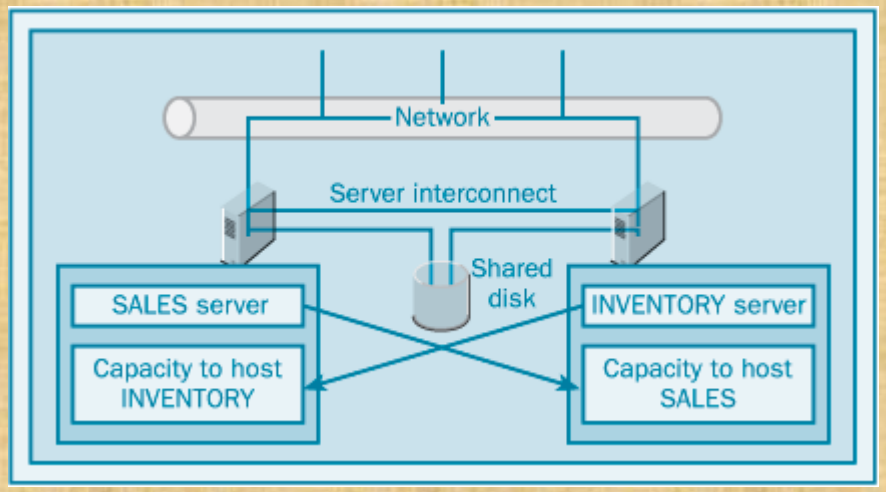


Рис. 10. Кластер с высокой готовностью, со статическим балансированием нагрузки

Пример 2 – система с "горячим резервированием" с максимальной готовностью

Такие системы обеспечивают максимальную готовность и производительность для всех системных ресурсов. Недостатком этой конфигурации является то, что денежные затраты в оборудование почти никогда не работают. Один из узлов работает как первичный узел и исполняет все клиентские запросы. А другой узел простаивает. Этот простаивающий узел служит для "горячего резервирования" и становится доступен только в случае перехода по отказу. Если первичный узел отказывает, то узел для горячего резервирования немедленно принимает на себя все операции и продолжает обслуживание клиентских запросов (рис. 11)

Данная конфигурация лучше всего подходит для наиболее критически важных приложений. Если ваша фирма зависит от продаж через Интернет, то вашему серверу для веб-торговли, возможно, следовало бы работать в этой конфигурации. От наличия и работоспособности системы зависит бизнес фирмы, и это оправдывает расходы на простаивающее оборудование.

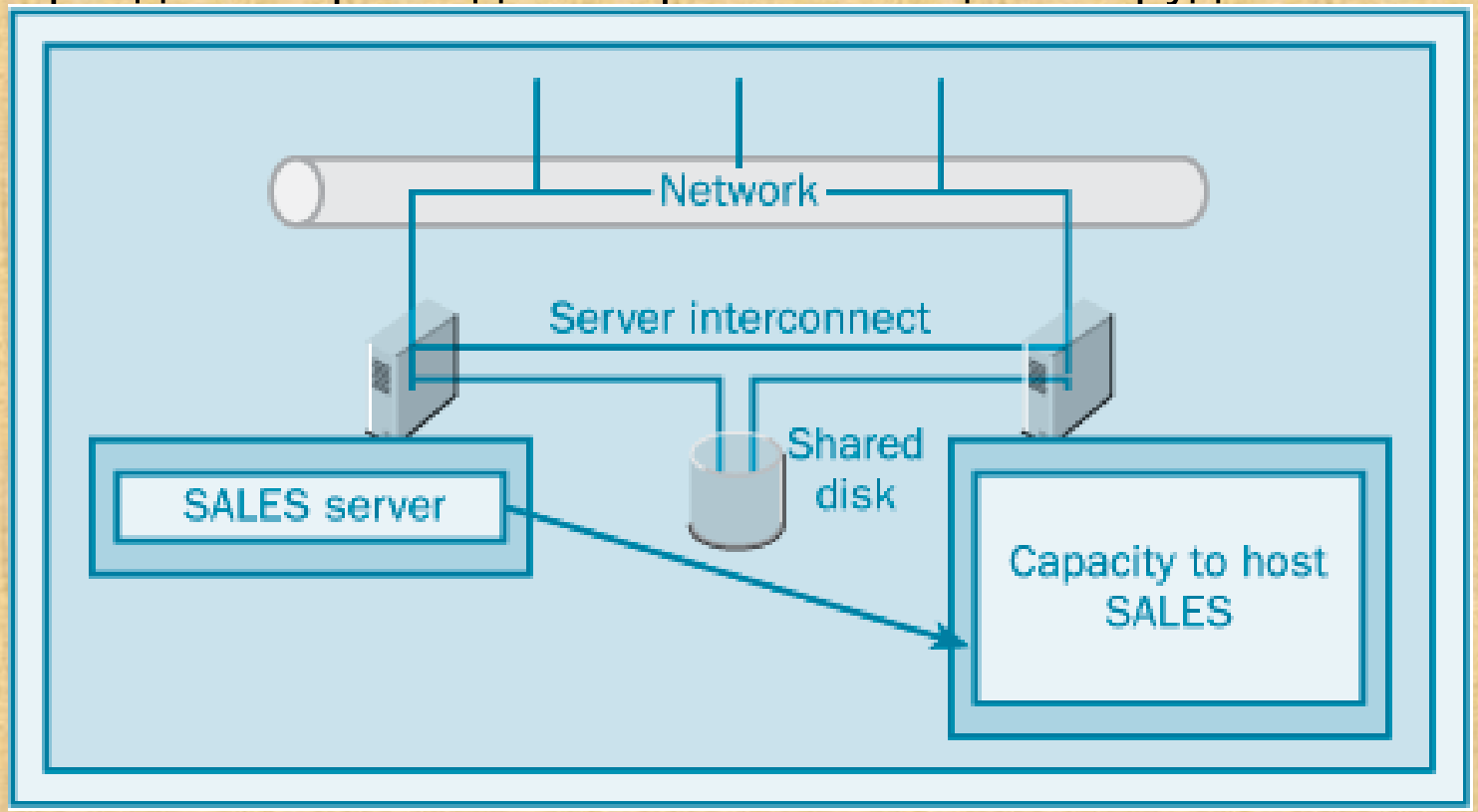


Рис. 11. Система с "горячим резервированием" с максимальной готовностью

Пример 3 – кластеризация части сервера

Конфигурация с кластеризацией части сервера является примером того, насколько гибкой может быть MSCS. В этой системе переход по отказу разрешен лишь для некоторых приложений. Как показано на рис. 12, можно задать, чтобы некоторые приложения оставались доступными при отказе узла, а некоторые – нет.

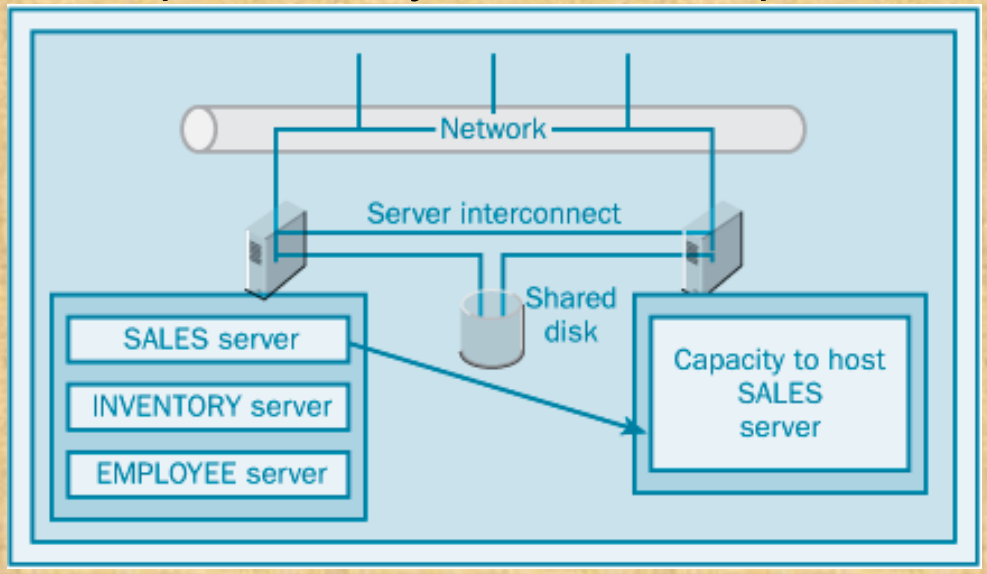


Рис. 12. Кластеризация части сервера

Эта конфигурация идеальна для случаев, когда необходимо максимально повысить полезное использование оборудования, но в то же время ограничить расходование мощностей, служащих для подстраховки критически важных приложений. Кроме того, такая конфигурация поддерживает работу приложений, не рассчитанных на кластеризацию, и вместе с тем обеспечивает переход по отказу для приложений, рассчитанных на кластеризацию.

Пример 4 – только виртуальные серверы, без переходов по отказам

Последний пример на самом деле не является кластером, но в нем используется служба MSCS и ее поддержка виртуальных серверов. Эта конфигурация помогает организовывать ресурсы и представлять их в сети (рис. 14). Применение виртуальных серверов позволяет задавать для ресурсов осмысленные, описательные имена, а не пользоваться обычным списком имен серверов. Кроме того, MSCS станет автоматически перезапускать приложения и ресурсы после отказов сервера. Эта возможность полезна для приложений, не имеющих внутренних механизмов для их перезапуска.