1. СУБД

1.1. Архитектура построения СУБД

Под СУБД понимают совокупность языковых и программных средств, обеспечивающих создание, поддержание (редактирование) и доступ к данным как со стороны пользователей, так и со стороны приложений. СУБД предоставляет:

- средства поддержки логической модели данных;
- развитый пользовательский интерфейс;
- средства программирования высокого уровня, с помощью которых можно создавать свои собственные приложения – БД;
- средства администрирования, обеспечения целостности, безопасности и секретности информации.

Основные операции (функции) СУБД заключаются в хранении, управлении, обработки и представлении данных. Их можно условно разбить на следующие классы:

a) операции определения структуры данных;
b) операции изменения данных:
   - добавление (вставка) данных,
   - удаление данных,
   - модификация (редактирование) данных.

   Все эти операции зачастую заменяются одним термином – обновление данных.
c) операции поиска и доступа к информации:
   - выборка (получение данных из БД и обеспечение пользователю доступности выбранных данных);
   - поиск. Предусматривает установку курсора на записи, удовлетворяющей определенному критерию;
   - сортировка;
   - фильтрация;
d) операции поддержки целостности и восстановления данных;
e) операции контроля за доступом;
f) операции поддержки обмена данными;
g) операции администрирования.

Сюда также можно отнести так называемые низкоуровневые операции: управление данными во внешней памяти; управление буферами

оперативной памяти; управление параллельностью и транзакциями; ведение журнала изменений в БД (имеется не во всех СУБД).

Для работы с хранящейся в БД информацией СУБД предоставляет программам и пользователям следующие два вида языков:

- язык описания данных (DDL – data definition language) – Высокоуровневый непроцедурный язык декларативного типа, предназначенный для описания логической структуры данных. Может дополнительно подразделяться на SDL – storage definition language и VDL – view definition language.
- язык манипулирования данных (DML – data manipulation language) – совокупность инструкций, обеспечивающих выполнение основных операций по работе с данными: ввод, удаление, модификацию и выборку данных. Обычно инструкции DML указывают, какие данные необходимо извлечь, а не как они должны быть извлечены, и поэтому его также называют декларативным языком.

СУБД представляет собой сложную систему, состоящую из многих компонентов. Структурно схему построения СУБД можно представить следующей схемой (рис. 1).

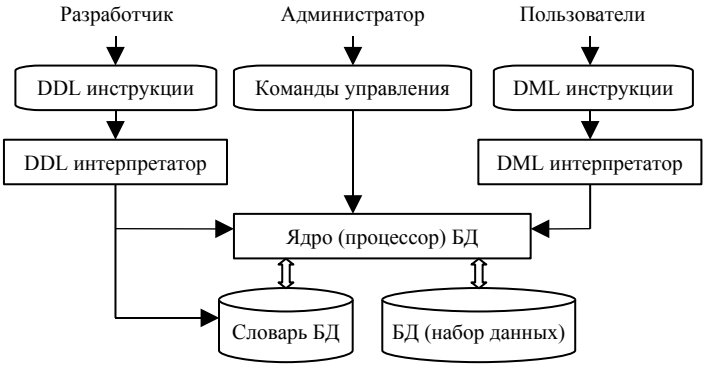

Рис. 1. Структурная схема СУБД

База данных и словарь данных (системный каталог, хранящий мета-данные) хранятся на внешних носителях. Операции доступа к данным на низком уровне (запись-чтение) контролируются OS. Если требуется высокая производительность, то такие операции могут контролироваться самой СУБД. На более высоком уровне операции доступа контролируются ядром БД, которая использует для этого информацию, содержащуюся в словаре данных. Практически всегда для каждой операции доступа в оперативной памяти (ОП) выделяется область, называемая буфером. Необходимость

буферизации обусловлена тем, что объем ОП всегда меньше объема внешней памяти. Такой подход повышает производительность системы и позволяет предоставить доступ к одним и тем же данным разным процессам (в конечном случае пользователям). Ядро БД также ответственно за поддержку безопасности и целостности БД.

DDL интерпретатор обрабатывает инструкции определения/изменения структуры данных и сохраняет результаты в системном каталоге. Запросы от пользователей интерпретируются интерпретатором DML и обрабатываются затем ядром БД. Так как зачастую запросы формируются через высокоразвитый интерфейс и могут представлять собой нечто достаточно абстрактное, то они проходят предварительную предкомпиляцию в инструкции DML. Операции управления и администрирования выполняются непосредственно ядром БД и промежуточной обработки не требуют.

Признаки, по которым можно различать СУБД:

1. По типу поддерживаемой модели: реляционная, объектная, объектно-реляционная, иерархическая, сетевая, другие (многомерная, постреляционная).
2. По количеству поддерживаемых пользователей: однопользовательские и многопользовательские.
3. По количеству компьютеров, на которых располагается СУБД: централизованные (один компьютер) и распределенные (Distributed DBMS - DDBMS).
4. По назначению: общецелевые и специальные (обслуживание банков, продажи и резервирования билетов и т.д. Часть из них образует категорию OLTP – on-line transaction processing systems).

1.2. Индексы

Индексы предназначены для ускорения доступа к данным и повышения скорости поиска. Основным их назначением является устранение последовательного или пошагового сканирования записей при поиске нужных данных. Помимо этого индексы также позволяют производить проверку значения поля на уникальность и сортировку записей в таблице. Индекс – это внутренняя таблица, имеющая два столбца: упорядоченные значения выражения, содержащего все поля, включенные в индекс, и местоположение каждой записи с данным значением индексного выражения. Поля, на основе которых создается индекс, обычно могут быть упорядочены как по возрастанию своих значений, так и по убыванию, хотя некоторые СУБД поддерживают только возрастающую сортировку. При выполнении сортировки изменение физического положения записей не

происходит. Меняется только визуальное представление очередности следования записей. Таким образом, порядок следования записей в базе данных легко менять, назначая соответствующий индекс таблице.

В первом поле индекса можно хранить значения индексных полей таблицы либо свертку поля (так называемый хеш-код). Преимущество хранения хеш-кода вместо соответствующих значений состоит в том, что длина свертки независима от длины исходных значений и имеет достаточно малую величину (например, 4 байта), что существенно снижает время поиска. Недостатком хеширования является необходимость выполнения операции свертки, что требует определенных затрат времени, и возможность возникновения коллизий (свертка различных значений может дать одинаковый хеш-код).

Для организации ссылки на запись таблицы зачастую используются абсолютные или относительные адреса дисковой памяти компьютера. Так как таблицы хранятся в виде совокупности блоков данных фиксированного размера, например целого числа кластеров, то преимущественно в качестве адресов записей используются адреса начала этих блоков.

Таким образом, индекс обычно сохраняет каждое значение индексированного поля вместе со списком указателей на все блоки данных физического носителя, которые содержат записи с данным значением индексируемого поля. Значения всегда хранятся в упорядоченном виде, что позволяет применить быстрые алгоритмы поиска, например бинарный поиск. Число обращений к индексу будет $\log_2 n$, где n – число записей в индексе. Индексный файл имеет намного меньший размер, чем файл данных, тем самым еще более повышая эффективность поиска.

Индексы различают на первичные и вторичные в зависимости от того, определяет ли поле, на котором основан индекс, физический порядок записей в таблице, или нет. Первичный (Primary) индекс создается на основе первичного ключа таблицы, т.е. поля, которое определяет физический порядок следования записей в таблице и является уникальным. Так как таблица может содержать не более одного поля, определяющего физический порядок записей в таблице, то для нее может быть определен только один первичный индекс. Дополнительно для таблицы можно определить несколько вторичных (secondary) индексов, основанных на неупорядоченных физически полях.

1.2.1. Первичные индексы

Первичный индекс состоит из двух полей. Первое поле имеет тот же тип данных, что и поле, на котором основан индекс, а второе содержит указатели на блоки данных физического носителя. Так как в одном блоке данных обычно помещается несколько записей, то первое поле индекса

будет содержать не все значения индексированного поля, а только первые для блока данных. Соответственно, в первом поле записывается значение первичного ключа для первой записи в блоке, а во втором – указатель на этот блок данных (смотри рис. 2).

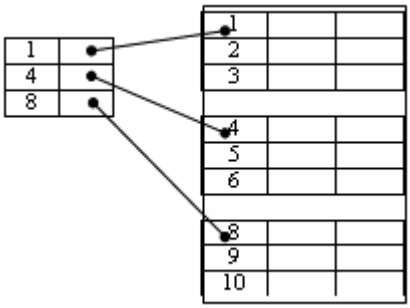

Рис. 2. Структура первичного индекса

Число записей в первичном индексе будет определяться числом блоков данных, отведенных для хранения таблицы. Поиск некоторой записи будет вначале проводиться по первому полю индекса, затем по упорядоченным значениям индексируемого поля в блоке данных. Скорость поиска данных с использованием первичного индекса будет максимально высокой, так как число записей в индексе намного меньше, чем в таблице и для индекса и для блоков данных можно использовать быстрый алгоритм поиска вследствие упорядоченности значений.

1.2.2. Вторичные индексы

Структура вторичных индексов отличается от первичного тем, что второе поле индекса содержит указатели на блоки данных с неупорядоченными записями и тем, что для индексируемых полей с повторяющимися значениями приходится хранить список указателей на все блоки данных физического носителя, которые содержат записи с данным значением индексируемого поля. Помимо указателей на начала блоков данных применяются также указатели на записи с данным значением индексируемого поля (смотри рис .3).

Иначе говоря, первое поле вторичного индекса содержит упорядоченные значения индексируемого поля, а второе – указатели на блоки данных физического носителя, которые содержат записи с данным значением индексируемого поля.

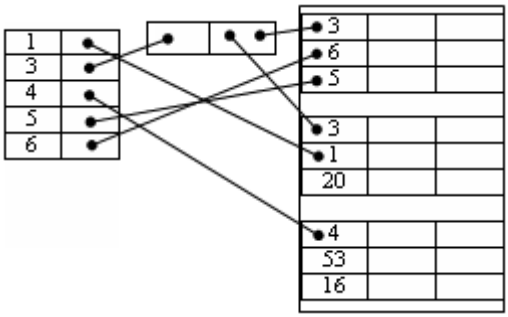

Рис. 3. Структура вторичного индекса

1.2.3. Многоуровневые индексы

По способу своей организации индексы разделяются на одноуровневые и многоуровневые. Разбиение некоторого индекса на несколько уровней производится для дальнейшего увеличения скорости поиска записей при увеличении числа записей в индексируемой таблице. Число записей в самом индексе и, следовательно, размер индексного файла, может быть настолько большим, что операция открытия индекса и поиск в нем может занимать значительное время. Бинарный поиск требует приблизительно $\log_2 b_i$ обращений к индексу, ссылающемуся на b_i блоков данных. Если разбить такой индекс на блоки записей одинаковой длины и включить в дополнительный уровень только значения первых записей полученных блоков и соответствующие указатели на эти блоки, то общее число обращений к индексу значительно сократиться. Количество обращений к индексу уже будет $\log_{fs} b_i$, где fs это число записей в блоке индекса первого уровня. В такой схеме второй уровень будет являться первичным индексом для первого уровня, как представлено на рис. 4.

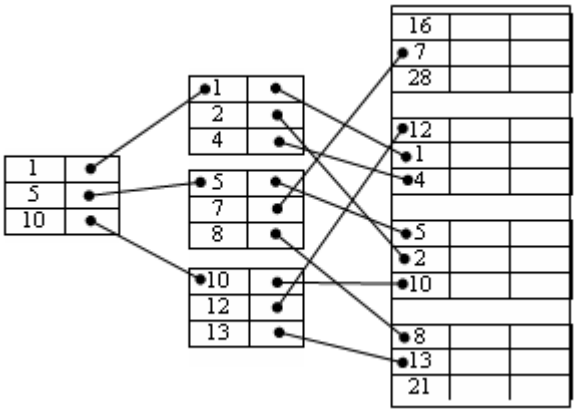

Рис. 4. Структура многоуровневого индекса

1.2.4. Составные индексы

Если в таблице поиск зачастую ведется по нескольким полям одновременно, то для ускорения поиска необходимо определить составной индекс. Составной индекс не отличается значительно по своей структуре от обычных индексов. Отличие состоит в том, что первое поле индекса содержит свертку значений всех полей, входящих в индекс. К примеру, нам требуется часто искать некоторое предприятие по стране регистрации, городу и собственно по его имени. Тогда при создании индекса вначале производится сортировка по всем полям, включенным в индекс, в порядке их следования (т.е. будет произведена сортировка по стране регистрации, затем внутри каждой страны по городу, и наконец, внутри каждого города по названию предприятия). Затем на основе отсортированных значений происходит создание первого поля индекса и заполнение второго поля указателями на блоки данных физического носителя, содержащими записи с данными значениями индексированных полей. Очевидно, что совокупность вторичных индексов, созданных для каждого из полей, входящих в составной индекс, не сможет заменить составного индекса, поскольку результат вложенной сортировки не равен результату последовательной сортировки.

Ускорение поиска достигается опять возможностью применения быстрых алгоритмов поиска для всех полей составного индекса (вследствие их упорядоченности). В большинстве систем существует одно ограничение на использование составного индекса, а именно только последнее условие (т.е. сравнение с последним полем индекса) может быть неравенством. Например, Страна = ‘Беларусь’ And Город = ‘Минск’ And Предприятие <‘С’.

Контрольные вопросы и задания

1. Дайте определение СУБД.
2. Перечислите основные операции СУБД.
3. Перечислите три основные операции СУБД, отвечающие за поиск данных.
4. Зарисуйте структурную схему СУБД и поясните назначение каждого блока.
5. Дайте классификацию СУБД.
6. Перечислите признаки, по которым можно различать СУБД.
7. Сформулируйте назначение индексов в БД.
8. Представьте схематично структуру первичного индекса.
9. Представьте схематично структуру вторичного индекса.
10. Для чего используются многоуровневые индексы.
11. Поясните, почему скорость поиска с использованием первичного индекса является наибольшей.
12. Чем отличается первичный индекс от вторичного.
13. Чем отличается составной индекс от простого.

2. АРХИТЕКТУРА БАЗ ДАННЫХ

В данной главе мы кратко рассмотрим архитектуры доступа к данным ведущего производителя программного обеспечения Microsoft, а также основные аспекты распределенного хранения данных и доступа к ним в многопользовательском режиме. Интерес к распределенному совместному доступу к данным возник не случайно. Сегодня любая крупная организация содержит большой штат работников и географически распределена. Доступ же к общим корпоративным данным может потребоваться каждому в любой момент времени. Простое дублирование информации на каждом персональном компьютере не решает проблему, поскольку данные могут динамично меняться. К тому же хранение больших объемов данных на каждом компьютере очень ресурсоемко. Решение подобных проблем требует серьезного развития технологий совместного доступа к данным и систем управления и обработки данных.

2.1. Технологии доступа к данным

2.1.1. Стандартные технологии доступа к данным.

В первых версиях операционной системы Windows пользователи могли совместно использовать данные в разных приложениях, копируя и вставляя их с помощью буфера обмена (clipboard). Затем был предложен протокол динамического обмена данными Dynamic Data Exchange (DDE) для обеспечения более динамичного режима обмена данными. Однако этот протокол функционировал медленно и ненадежно, и на смену ему был разработан значительно более эффективный протокол связывания и внедрения объектов Object Linking and Embedding (OLE).

OLE – это технология, которая позволяет создавать составные приложения, включающие в себя объекты, созданные с помощью других приложений. Объекты могут быть встроены в основное приложение или просто быть связаны с ним. Приложение, включающее в себя другие объекты, называется контейнером OLE, а приложение, поставляющее свои объекты для встраивания либо связывания – сервером OLE. Объектами OLE-приложения могут быть текстовые документы, диаграммы, электронные таблицы, электронные сообщения, графические изображения т.д. После вставки или внедрения объект отображается внутри клиентского приложения и хранится вместе с ним. Причем для редактирования связанных данных пользователю достаточно дважды щелкнуть мышью на встроенном объекте, в результате чего будет запущено приложение, в котором этот объект был создан. При связывании объектов в контейнере хранится лишь ссылка на объект-источник. После обновления исходного

файла объекта обновляется и его представление в составном приложении. Помимо встраивания и связывания объектов эта технология позволяет вызывать функции одного приложения из другого приложения. Другими словами OLE является объектно-ориентированной технологией разработки повторно используемых программных компонентов.

В целях дополнительной интеграции объектов концепция OLE была значительно расширена, позволив создавать самостоятельные функциональные компоненты, предоставляющие свои функции (сервисы) другим объектам. В такой архитектуре создание и сопровождение одних объектов может производиться совершенно независимо от других объектов. Взаимодействие объектов определятся посредством специально организованных интерфейсов. Компонентная модель объектов Component Object Model (COM) является объектно-ориентированной моделью, которая состоит из спецификации, определяющей интерфейс между объектами внутри системы, и конкретной реализации в виде динамически связываемой библиотеки Dynamic Link Library (DLL).

Технология COM предоставляет стандартный метод поиска и инициализации объектов, а также организации связи между приложением и объектом. Одним из основных достоинств технологии COM является то, что она предоставляет унифицированный (двоичный) стандарт взаимодействия, не зависящий от языка программирования, использовавшегося при создании приложения и объекта. Идеология COM была реализована в 1993 году в спецификации OLE 2.O.

COM позволяет создавать централизованные приложения. Для создания распределенных корпоративных систем была предложена архитектура Distributed Component Object Model (DCOM). DCOM расширяет архитектуру COM до распределенной компонентной среды, в которой компоненты одинаково выглядят для клиентов на локальном и удаленном компьютерах. DCOM реализует это, заменяя сообщение между процессами клиента и компонента соответствующим сетевым протоколом.

2.1.2. Технология Open Database Connectivity

Технология открытого доступа к данным ODBC была разработана фирмой Microsoft для обеспечения возможности взаимосвязи между различными SQL-совместимыми БД, причем в этой технологии SQL используется как стандартный механизм доступа к данным. Необходимость создания ODBC появилась вследствие того, что каждая фирма-разработчик СУБД использовала свой диалект SQL, что делало невозможным обмен данными между двумя БД различных форматов. Поэтому вначале был разработан общий стандарт на SQL, получивший название CLI (Call Level Interface). В его основу были положены уже существующие стандарты

X/Open и ISO. Затем в рамках ODBC каждой фирме-разработчику СУБД было предписано разработать драйвер перевода своего диалекта SQL в CLI, и наоборот.

Таким образом, основное назначение ODBC состоит в абстрагировании приложения от особенностей ядра используемой базы данных. Технология ODBC предусматривает создание дополнительного уровня между приложением и используемой СУБД. Предоставленный интерфейс обеспечивает высокую степень взаимодействия, позволяя одному приложению обращаться к разным базам данных с помощью одного и того же кода. Это позволяет создавать распределенные (преимущественно клиент-серверные) гетерогенные приложения без учета особенностей конкретных СУБД. В качестве сервера может выступать любой сервер БД, имеющий драйвер ODBC (MS SQL Server, Oracle и т.д.) или даже обычная БД, если требуется совместная обработка данных, написанных в разных форматах. ODBC находится как бы посередине между приложениями и используется как средство коммуникации между клиентской и серверной частями. Службы ODBC обеспечивают соединение с БД, получение от приложения запросов на выборку информации и перевод их на язык ядра адресуемой базы данных для доступа к хранимой в ней информации. Одна из главных целей создания ODBC – скрыть сложность соединения с сервером и по мере возможности автоматизировать выполнение многочисленных процедур, связанных с получением данных. ODBC требует от разработчика указания только имени источника данных (DSN – data source name), при этом функции, драйвера, адреса серверов, сети и шлюзы скрыты от пользователя.

Достоинством технологии ODBC является простота разработки приложений, обусловленная высоким уровнем абстрактности интерфейса доступа к данным практически любых существующих типов СУБД. Основной недостаток технологии ODBC связан с необходимостью трансляции запросов, что снижает скорость доступа к данным.

2.1.3. Технология Object Linking and Embedding Database

ODBC хоть и является универсальной технологией, но предоставляет доступ только к реляционным SQL ориентированным БД. Но реляционные БД – не единственный источник данных.Данные могут быть представлены в любом виде и формате. Например, в качестве данных могут выступать реляционные БД, электронные таблицы, документы в rtf-формате, почтовые системы и т. д. Соответственно возникла потребность или создать единый формат хранения данных, что дорого и неэффективно, либо нарастить имеющиеся технологии интерфейсами доступа к любым типам данных.

Технология OLE DB реализует это требование, являясь более универсальной технологией, нежели стандартные технологии OLE и COM.

В технологии OLE DB используется механизм провайдеров, под которыми понимают поставщики данных, находящиеся в надстройке над физическим форматом данных. Провайдер OLE DB представляет собой компонент COM, позволяющий принимать вызовы OLE DB API и выполнять все необходимое для обработки запроса к источнику данных. Кроме поставщиков данных имеются также сервис-провайдеры, реализующие самые различные сервисные функции. Технология OLE DB может использовать ODBC для доступа к реляционным БД. В этом случае применяется OLE DB–провайдер для доступа к ODBC данным. Таким образом, технология OLE DB не заменяет технологию ODBC, она позволяет организовывать доступ к источникам данных через различные интерфейсы и в том числе через ODBC.

2.1.4. Технологии Data Access Objects и ActiveX Data Objects

Хотя ODBC и OLE DB считаются хорошими интерфейсами передачи данных, но как программный интерфейс они имеют много ограничений, поскольку являются низкоуровневыми. Для снятия этих ограничений были предложены технологии Data Access Objects (DAO) и ActiveX Data Objects (ADO). Данные технологии представляют собой высокоуровневые объектные модели (библиотеки функций) и создают еще один уровень абстракции между приложением и функциями ODBC и OLE DB. Технология DAO предназначена преимущественно для создания БД с помощью СУБД Access, так как кроме замены совокупности низкоуровневых функций ODBC несколькими высокоуровневыми осуществляет также прямой доступ к функциям ядра Microsoft Jet базы данных Access.

Технология ADO предоставляет иерархическую модель объектов для доступа к различным OLE DB–провайдерам данных и характеризуется еще более высоким уровнем абстракции. Объектная модель ADO включает объекты, обеспечивающие соединение с провайдером данных, создание SQL запросов к данным, создание набора записей на основе запроса и т.д. Особенностью технологии ADO является возможность ее использования в Интранет/Интернет приложениях для доступа к различным источникам данных. В целом, технологию ADO можно охарактеризовать как наиболее современную технологию разработки приложений для работы с распределенными БД по технологии клиент-сервер.

В общем случае, архитектуру доступа к данным, предоставляемую фирмой Microsoft можно представать с помощью следующей схемы (смотри рис. 5).

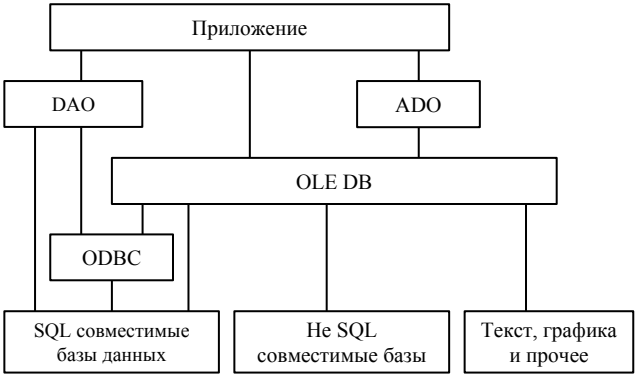

Рис. 5. Архитектура доступа к данным Microsoft

2.2. Архитектура клиент-сервер

Если речь идет о некоторой БД как самостоятельной функциональной единице, то под ней понимают совокупность набора данных и программы обслуживания. Программа обслуживания, реализующая функции управления, обработки и представления данных, может быть как некоторой коммерческой СУБД, так и самостоятельной программой, где ядро СУБД, обеспечивающее управление данными, будет представлена лишь несколькими DLL. При размещении БД в сети возможны различные варианты распределения данных и функций БД по узлам сети. Данные могут храниться на одном выделенном компьютере и могут быть распределены по всем узлам сети. Функции обработки и представления данных также могут быть самым различным способом распределены по узлам сети. В зависимости от числа узлов сети, между которыми выполняется распределение функций СУБД, можно выделить двухзвенные модели, трехзвенные и т.д.

Исторически первыми появились распределенные БД с применением файл-серверной архитектуры. В таких БД по запросам пользователей файлы базы данных передаются на персональные компьютеры (ПК), где и производится их обработка. В таком случае БД может располагаться на файл-сервере, в качестве которого может использоваться наиболее мощный компьютер. Функция файл-сервера заключается в основном в хранении БД и обеспечении доступа к ним пользователей, работающих на различных компьютерах. Эти функции обеспечиваются, как правило, той же СУБД, которая работает и на компьютерах пользователя. Для каждого клиента во время работы создается локальная копия данных, с которой он работает. При этом решаются задачи, связанные с возможным одновременным доступом нескольких пользователей к одним и тем же данным. Забота о

целостности данных при такой организации работы целиком возлагается на программы клиентов. Если они недостаточно продуманы, в базу данных можно легко занести ошибки, которые могут отразиться на всех пользователях. Если используемая СУБД или не имеет достаточных средств для обеспечения многопользовательского доступа к данным, или неправильно сконфигурирована, то нарушение целостности может произойти также и при попытке одновременного изменения одних и тех же данных. Зачастую такие действия приводит к полному повреждению внутренней структуры БД.

При небольших объемах данных архитектура файл-сервер вполне соответствует современным требованиям, но с увеличением числа компьютеров в сети или ростом БД начинают возникать проблемы, связанные с резким падением производительности системы. Это связано с увеличением объема данных, передаваемых по сети, т.к. вся обработка производится на компьютере пользователя. Если пользователю потребуется несколько строк из таблицы объемом в сотни тысяч записей, то сначала вся таблица с файл-сервера будет передана на его компьютер, а затем СУБД отберет нужные записи. Даже реализация постепенной постраничной подкачки данных не решает проблему. Намного более удобной для совместной обработки данных является архитектура клиент-сервер.

Архитектура клиент-сервер разделяет приложение на две части, используя лучшие качества с обеих сторон. Клиентская часть (Front-end) находится на компьютерах пользователей и обеспечивает легкий в использовании интерактивный интерфейс. Сервер (Back-end) находится на выделенном компьютере и обеспечивает управление данными, разделение информации, администрирование, обеспечение целостности, безопасности и секретности.

В общем случае сервером определенного ресурса в компьютерной сети называется компьютер (программа), управляющий этим ресурсом, клиентом – компьютер (программа), использующий этот ресурс. В качестве ресурса компьютерной сети могут выступать, к примеру, базы данных, файловые системы, службы печати, почтовые службы. Тип сервера определяется видом ресурса, которым он управляет. Например, если управляемым ресурсом является база данных, то соответствующий сервер называется сервером базы данных. Архитектура клиент-сервер предполагает централизованное хранение данных с двухзвенным распределением функций СУБД. В этой архитектуре данные обычно хранятся на выделенном компьютере под управлением специальной программы сервера, а доступ к данным и их представление организуется через клиентские программы. Достоинством организации информационной системы по архитектуре клиент-сервер является удачное сочетание централизованного

хранения, обслуживания и коллективного доступа к общей корпоративной информации с индивидуальной работой пользователей над персональной информацией. В такой системе легко реализовать многопользовательский доступ к данным, поскольку появляется возможность предоставлять одни и те же данные нескольким клиентам одновременно, решая при этом проблемы совместного доступа.

В архитектуре клиент-сервер клиентское приложение устанавливает соединение с сервером и формирует запрос к серверу БД. Выполнение запроса происходит на сервере. Затем результат запроса посылается клиенту для использования и просмотра. Так как обычно результатом запроса является небольшая часть хранимой в БД информации, то нагрузка на сеть сильно уменьшается. На сервере происходит также обработка транзакций и правил целостности (бизнес – логики). Так как SQL предоставляет стандартный интерфейс для СУБД различных типов, то он может использоваться как средство коммуникации между сервером и клиентом. В таком случае сервер часто называется сервером запросов (SQL сервером).

Если вся обработка данных происходит на стороне сервера, то клиент выполняет только функции интерфейса с пользователем. В этом случае клиентское приложение называют «тонким» клиентом. Если часть обработки данных производится на стороне клиента, то говорят о «толстом» клиенте. По разделению функций между клиентом и сервером можно выделить следующие типы архитектур (смотри рис. 6):

1. удаленное представление,
2. распределенная функция,
3. удаленный доступ к данным.

В модели удаленного доступа к данным (Remote Data Access — RDA) программы, реализующие функции представления информации и функции их обработки, совмещены и выполняются на компьютере-клиенте. Функции сервера фактически ограничиваются управлением данных и обработкой запросов со стороны клиентов. Основное достоинство RDA-модели состоит в наличии большого числа готовых СУБД и других инструментальных средств, обеспечивающих быстрое создание программ клиентской части. Недостатками RDA-модели являются, во-первых, довольно высокая загрузка системы передачи данных вследствие того, что вся логика обработки сосредоточена на компьютере-клиенте, а обрабатываемые данные расположены на удаленном узле. Во-вторых, RDA системы неудобны с точки зрения разработки, модификации и сопровождения. Основная причина состоит в том, что в получаемых приложениях прикладные функции и функции представления тесно взаимосвязаны.

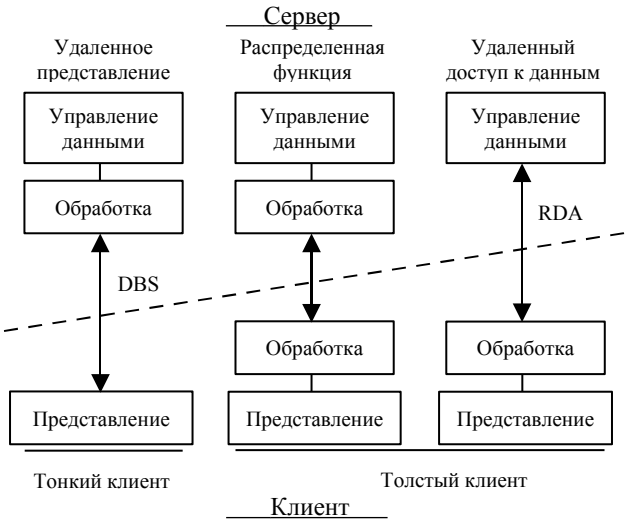

Рис. 6. Модели архитектуры клиент-сервер

Поэтому даже при незначительном изменении функций системы требуется переделка всей прикладной ее части. Если функции сервера заключаются лишь в хранении данных, то эта модель ничем не отличается от архитектуры файл-сервер.

Модель удаленного представления, иначе модель сервера БД (Database Server – DBS) отличается от предыдущей модели тем, что функции компьютера-клиента ограничиваются функциями представления, в то время как прикладные функции (обязательно включающие обеспечение целостности, безопасности и секретности) реализуются на стороне сервера. Логика работы приложения реализуются в виде хранимых процедур и триггеров. Процедуры хранятся в словаре БД и разделяются несколькими клиентами. В некоторых СУБД на сервере можно хранить и сами запросы, называемые в таком случае хранимыми командами. Хранимые команды выполняются значительно быстрее, так как не требуется каждый раз производить их синтаксический разбор и оптимизацию кода. Достоинствами модели DBS являются: возможность централизованного администрирования приложений и обеспечения целостности, а также эффективное использование вычислительных и коммуникационных ресурсов. К недостаткам модели следует отнести ограниченность действий, которых можно выполнять с помощью хранимых процедур и триггеров и сравнительно низкая эффективность использования вычислительных ресурсов обоих компьютеров.

В модели распределенной функции логика обработки данных распределена по двум узлам. Такую модель могут иметь БД, в которых общая часть прикладных функций реализована на компьютере-сервере, а специфические функции обработки информации выполняются на

компьютере-клиенте. Функции общего характера могут включать в себя стандартное обеспечение целостности данных, например, в виде хранимых процедур и триггеров, а оставшиеся прикладные функции реализуют специальную прикладную обработку.

Кроме перечисленных способов можно еще выделить распределенное представление, где все функции БД включая представление информации сосредоточены на сервере, а клиентская часть системы практически вырождена, и распределенную БД, где по функциональной нагрузке клиент фактически становится равен серверу. В распределенном представлении основной функцией клиентской части является просто отображение информации на экране монитора и связь с основным компьютером через локальную сеть. Модель распределенной БД наоборот предполагает использование мощного компьютера-клиента, причем данные могут храниться как на компьютере-сервере, так и на компьютере-клиенте.

2.3. Многоярусные базы данных (multi-tired databases)

Если функцию хранения данных вынести на отдельный компьютер сети, то получим трехъярусный (трехзвенный) вариант представления данных. На нижнем уровне на компьютерах пользователя расположены приложения клиентов, обеспечивающих пользовательский интерфейс. На втором уровне расположен сервер приложений, обеспечивающий управление данными и реализующий несколько прикладных функций, каждая из которых оформлена как служба предоставления услуг всем требующим этого программам. Серверов приложений может быть несколько, причем каждый из них предоставляет свой вид сервиса. Любая программа, запрашивающая услугу у сервера приложений, является для него клиентом. На третьем уровне расположен удаленный сервер баз данных. Поскольку центральным звеном является сервер приложений, такую модель называют моделью сервера приложений, или AS-моделью (Application Server). Достоинством AS-модели является разгрузка сервера БД, а к недостаткам можно отнести увеличение нагрузки на сеть.

2.4. Распределенное хранение данных

Распределенные базы данных можно рассматривать как подвид распределенных вычислительных систем, занимающихся обработкой данных. Распределенная вычислительная система состоит из совокупности элементов (не обязательно однородных), которые соединены между собой с помощью коммуникационной сети и взаимодействующих между собой при решении некоторой общей задачи. Можно выделить два преимущества такой системы: 1) увеличение мощности системы при решении общей задачи и 2) возможность автономной работы отдельных элементов системы.

Таким образом, мы можем определить распределенную базу данных (Distributed Database – DDB), как совокупность логически взаимосвязанных баз данных, распределенных в компьютерной сети, и распределенную СУБД (Distributed Database Management System– DDBMS) как совокупность программ предназначенных для управления распределенной базы данных.

Существует два основных способа организации распределенной БД с распределенным хранением данных: фрагментация и репликация (тиражирование). Фрагментация, в свою очередь, бывает горизонтальной и вертикальной. При горизонтальной фрагментации разбиение происходит за счет помещения в отдельные таблицы с одинаковой структурой не перекрывающихся групп строк. При вертикальной фрагментации разбиение происходит по столбцам, одни столбцы формируют одну таблицу, другие – другую. При этом для сохранения идентификации целой записи в отдельных фрагментах приходится в каждый фрагмент добавлять первичный ключ таблицы. Информация о местоположении каждого фрагмента обычно хранится в так называемом глобальном словаре данных, который в свою очередь также может быть распределенным. При раздельном хранении фрагментов данных СУБД должна обеспечивать такой механизм работы, чтобы для программ и пользователей распределенная система воспринималась как единая централизованная БД.

Если БД или хотя бы один фрагмент данных может располагаться более чем на одном компьютере, то говорят о репликации (или иначе тиражировании) данных. Соответственно репликация бывает полной и частичной. При полной репликации на всех компьютерах размещаются синхронизируемые копии одной и той БД. Безопасность и степень доступности данных в такой системе самая высокая. Система остается работоспособной пока хотя бы один компьютер системы находится в рабочем режиме. Скорость выборки локальных данных также максимальна (если пренебречь эффектом увеличения скорости вследствие уменьшения размера БД при ее разбиении). Недостатком такой системы можно считать трудность синхронизации реплик при обновлении данных и то, что между обновлениями копии БД могут отличаться друг от друга. Другой противоположностью полной репликации является отсутствие репликации, где каждый фрагмент данных имеет только одну копию. Между этими крайними случаями находится широкий спектр вариантов частичной репликации. Степень репликации зависит от требуемой доступности данных и от обеспечения необходимой производительности операций обновления данных. Например, если требуется максимальная степень доступности данных и нет необходимости в частом их обновлении, то полная репликация является наилучшим решением. К фрагментации данных прибегают в том случае, если доступ к некоторому фрагменту данных

требуется преимущественно для пользователей одного или нескольких компьютеров. Такой подход позволяет обеспечить максимальную скорость работы с данными для пользователей этих компьютеров. В общем случае поиск оптимального решения размещения данных может представлять сложную задачу оптимизации.

Первым фактором, по которому можно различать распределенные БД, является степень однородности. Если все пользователи РБД используют одно и тоже программное обеспечение (СУБД) для доступа к данным, и если данные, расположенные на всех компьютерах сети, контролируются все той же СУБД, то такую РБД называют однородной. В противном случае – неоднородной или гетерогенной.

Второй фактор – это степень автономности. С одной стороны, мы можем иметь РСУБД с полным отсутствием локальной автономности, которая имеет единую концептуальную схему данных, единый центр обработки запросов и транзакций, где части единой БД просто распределены по разным компьютерам. С другой стороны, мы можем иметь РСУБД, которая хоть и имеет некоторую общую схему данных, но составлена из полностью автономных СУБД. Такая СУБД называется федеративной СУБД (Federated Database System – FDBMS). Федеративная СУБД может быть даже составлена из СУБД, поддерживающих различные модели данных, типы, ограничения и языки манипулирования данными. Такой вариант распределенной СУБД является более надежным и перспективным, но связан со значительными сложностями реализации. Проблемы в таких системах могут возникать не только вследствие отличия поддерживаемых моделей данных, но и вследствие отличия имен таблиц и полей, и наоборот, когда поля с одинаковыми именами могут обозначать разные данные. Федеративная СУБД поддерживает глобальную схему, на основании которой пользователи могут строить распределенные запросы и обновлять данные. Федеративная СУБД работает только с общей схемой данных, поскольку все локальные СУБД имеют свой собственные схемы данных и собственными силами обеспечивают доступ к данным всех их пользователей. Глобальная схема создается посредством слияние локальных схем данных. Программное обеспечение федеративной СУБД предварительно транслирует глобальные запросы в запросы к локальным БД. Затем результаты всех локальных запросов объединяются и предоставляются пользователю.

Основываясь на трехуровневой архитектуре БД, архитектура РСУБД может быть представлена следующим образом (смотри рис. 7).

Рис. 7. Архитектура РСУБД

Глобальная концептуальная схема представляет собой логическое описание всех БД, предоставляя ее так, как будто она не является распределенной. Схемы фрагментации, репликации и распределения определяют распределение данных по локальным компьютерам. Для различных типов РСУБД эти схемы могут иметь разную значимость. Для федеративной СУБД схемы фрагментации, репликации и распределения могут быть опущены вообще, а локальные схемы отображения могут быть представлены локальными внешними схемами.

Распределенные БД имеют преимущества перед традиционными централизованными БД, но не лишены и некоторых недостатков.

Распределенная БД обладает следующими преимуществами:

1. Разделяемость и локальная автономность. Географическая распределенность организации может быть отображена в распределении ее данных. В результате пользователи отдельных частей БД получают локальный контроль над данными и могут устанавливать локальные ограничения и права доступа;
2. Управление распределенными данными на разных уровнях «прозрачности». В идеальном варианте реальное расположение данных должно быть полностью скрыто от пользователя. Он должен

работать с распределенной БД как с системой, расположенной в одном месте;

3. Увеличение стабильности и надежности системы. С выходом из строя отдельных частей РСУБД будет продолжать функционировать;
4. Увеличение производительности. В распределенных БД фрагменты данных можно разместить там, где они наиболее нужны. Следовательно, падает нагрузка на сеть при пересылке данных. Размер фрагмента данных на локальном компьютере будет много меньшим, что также приведет к увеличению скорости работы с БД;
5. Увеличение гибкости за счет модульности системы.

К недостаткам таких систем можно отнести:

6. Повышение сложности, влекущее увеличение стоимости и срока разработки РБД;
7. Усложнение контроля за целостностью данных;
8. Усложнение контроля за безопасностью и секретностью данных, связанное с обеспечением защиты не только БД самих по себе, но и сетевых соединений.

2.5. Многопользовательские базы данных

Важнейшей целью создания БД является организация параллельного доступа многих пользователей к общим данным, используемыми ими совместно. Обеспечить параллельный доступ несложно, если все пользователи будут только читать данные, помещенные в базу. В этом случае работа каждого из них не оказывает никакого влияния на работу остальных пользователей. Но если два или больше пользователей одновременно обращаются к базе данных и хотя бы один из них имеет целью обновить хранимую в базе информацию, возможно взаимное влияние процессов друг на друга, способное привести к потере согласованности данных.

Существует два подхода решения проблем совместного доступа к общим данным: установление блокировок и управление параллельностью выполнения транзакций. В первом случае, на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других пользователей временно запрещается или ограничивается. Во втором случае, при одновременном доступе к одним и тем же данным операции манипулирования данными, заключенные в рамки транзакции, чередуется, и таким образом достигается их параллельное выполнение. Однако, несмотря на то что каждая из транзакций может сама по себе выполнена вполне корректно, подобное чередование операций способно приводить к

неверным результатам, из-за чего целостность и согласованность базы данных будет нарушена. В большинстве случаев эти два подхода применяются совместно.

В свою очередь различают оптимистический и пессимистический подходы в управлении параллельностью выполнения транзакций. Пессимистический подход обеспечиваем более высокий уровень безопасности, поскольку откладывается выполнение любых транзакций потенциально способных войти в конфликт с другими транзакциями. Оптимистический подход позволяет асинхронное выполнение транзакций в предположении, что вероятность конфликта невысока. Проверка на наличие конфликта откладывается до завершения транзакции и ее фиксации в базе данных.

2.5.1. Блокировки

Для организации многопользовательского доступа в СУБД применяется механизм блокировок. Суть блокировки состоит в том, что на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других пользователей временно запрещается или ограничивается. Например, при копировании таблицы она блокируется от изменения, хотя и разрешено просматривать ее содержимое. Блокировки автоматически устанавливаются СУБД, но возможно и их явное определение.

Рассмотрим четыре вида блокировок, перечисленные в порядке убывания строгости ограничений на возможные действия:

1. Полная блокировка. Означает полное запрещение любых операций над объектами БД. Этот вид блокировки обычно применяется при изменении структуры объектов;
2. Блокировка от записи. Накладывается в случаях, когда можно читать данные, но не изменять их. Изменение структуры данных также запрещается;
3. Предохраняющая блокировка от записи. Предохраняет объект от наложения на него со стороны других операций полной блокировки, либо блокировки от записи. Этот вид блокировки позволяет тому, кто раньше «захватил» объект, успешно завершить модификацию объекта. Такая блокировка обычно применяется при совместном изменении данных несколькими пользователями;
4. Предохраняющая полная блокировка. Предохраняет объект от наложения на него со стороны других операций полной блокировки. Обеспечивает максимальный уровень совместного использования объектов. Такая блокировка используется, например, для обеспечения одновременного просмотра несколькими пользователями одной

таблицы. Тогда никому не будет позволено изменить структуру общей таблицы.

При одновременном выполнении различных операций над одним и тем же объектом производится попытка их совмещения. Совмещение возможно тогда, когда совместимыми сказываются блокировки, накладываемые конкурирующими операциями. В отношении перечисленных выше блокировок действуют следующие правила совмещения:

- полная блокировка несовместима с другими блокировками, т.е. при наличии полной блокировки над объектом нельзя производить другие операции, накладывающие со своей стороны любую блокировку;
- блокировка от записи совместима с аналогичной блокировкой и предохраняющей полной блокировкой;
- предохраняющая блокировка от записи совместима с обоими видами предохраняющих блокировок;
- предохраняющая полная блокировка совместима со всеми видами блокировок, кроме полной.

2.5.2. Транзакции

Понятие транзакции имеет непосредственную связь с понятием целостности БД. В системах с развитыми средствами контроля целостности каждая транзакция начинается при целостном состоянии БД и должна оставить это состояние целостным после своего завершения. Несоблюдение этого условия приводит к тому, что вместо фиксации результатов транзакции происходит ее откат, и БД остается в таком состоянии, в котором находилась к моменту начала транзакции, т.е. в целостном состоянии. Часто БД могут обладать такими ограничениями целостности, которые просто невозможно не нарушить, выполняя только одну операцию изменения данных. Поэтому для поддержания подобных ограничений целостности допускается их нарушение внутри транзакции с тем условием, чтобы к моменту завершения транзакции условия целостности были соблюдены.

Различаются два вида ограничений целостности: немедленно проверяемые и откладываемые. К немедленно проверяемым ограничениям целостности относятся такие ограничения, проверку которых бессмысленно или даже невозможно откладывать. Немедленно проверяемые ограничения целостности соответствуют уровню отдельных операторов языкового уровня СУБД. При их нарушениях не производится откат транзакции, а лишь отвергается соответствующий оператор. Откладываемые ограничения целостности - это ограничения на базу данных, а не на какие-либо отдельные операции. По умолчанию такие ограничения проверяются при

конце транзакции, и их нарушение вызывает автоматическую замену оператора COMMIT на оператор ROLLBACK. Однако в некоторых системах поддерживается специальный оператор насильственной проверки ограничений целостности внутри транзакции. Если после выполнения такого оператора обнаруживается, что условия целостности не выполнены, пользователь может сам выполнить оператор ROLLBACK или постараться устранить причины нецелостного состояния базы данных внутри транзакции.

2.5.3. Уровни изолированности транзакций

В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей. Действительно, если с каждым сеансом работы с базой данных ассоциируется транзакция, то каждый пользователь начинает работу с согласованным состоянием базы данных, т.е. с таким состоянием, в котором база данных могла бы находиться, даже если бы пользователь работал с ней в одиночку. Следовательно требуется, чтобы обновления, выполняемые некоторой транзакцией T1, не были доступны для любой другой транзакции T2 до тех и только до тех пор, пока не будет завершено выполнение транзакции T1. Завершение выполнения транзакции открывает доступ ко всем обновлениям, выполненным данной транзакцией.

Существует несколько уровней изолированности транзакций. По приоритету их можно расположить в следующем порядке: READ UNCOMMITTED (чтение незафиксированных данных) < READ COMMITTED (чтение зафиксированных данных) < REPEATABLE READ (гарантия повторяемости считывания) < SERIALIZABLE (способность к упорядочению). Далеко не каждая СУБД поддерживают все уровни изолированности транзакций. Последний уровень имеется лишь у наиболее развитых.

Если все транзакции выполняются на уровне способности к упорядочению (принятом по умолчанию), то чередующееся выполнение любого множества параллельных транзакций может быть упорядочено. Однако, если любая транзакция выполняется на более низком уровне изоляции, то существует множество различных способов нарушения способности к упорядочению. Существует четыре особых случая нарушения способности к упорядочению, а именно:

1. Потерянные изменения. Результаты успешно завершенной операции обновления одной транзакции могут быть перекрыты результатами выполнения другой транзакции. Допустим, транзакция *T1* изменяет объект базы данных *A*. До завершения транзакции *T1* транзакция *T2* также изменяет объект *A*, перекрывая таким образом результат

предыдущей транзакции. Чтобы избежать такой ситуации, явно противоречащей требованию изолированности пользователей, требуется до завершения транзакции *T1* запретить любой другой транзакции изменять объект *A*. Отсутствие потерянных изменений является минимальным требованием к многопользовательской СУБД, поскольку в этом случае обеспечивается требование целостности БД при завершении любой транзакции.

Метод устранения: до завершения *T1* никакая другая транзакция не должна изменять объект *A*;

2. Неаккуратное считывание или чтение «грязных данных». Допустим, что транзакция *T1* выполняет вставку новой строки, затем транзакция *T2* извлекает эту строку, после чего выполнение транзакции *T1* отменяется. В результате транзакция *T2* обнаружит, что данной строки больше не существует в том смысле, что она никогда не существовала (поскольку транзакция *T1* действительно не была выполнена); Можно рассмотреть еще один сценарий совместного выполнения транзакций *T1* и *T2*. Транзакция *T1* изменяет объект базы данных *A*, а транзакция *T2* читает объект *A*. Поскольку операция изменения еще не завершена, транзакция *T2* видит несогласованные «грязные» данные, поскольку в следующий момент времени транзакция *T1* может быть отвергнута вследствие нарушения немедленно проверяемого ограничения целостности. Чтобы избежать ситуации чтения «грязных» данных, до завершения транзакции *T1*, изменившей объект *A*, никакая другая транзакция не должна иметь возможности читать объект *A*.

   Метод устранения: до завершения *T1* никакая другая транзакция не должна читать объект *A*;

3. Неповторяемое считывание. Допустим, транзакция *T1* читает объект *A*. До завершения транзакции *T1* транзакция *T2* изменяет объект *A* и успешно завершается. Транзакция *T1* повторно читает объект *A* и видит его измененное состояние. Чтобы избежать неповторяющихся чтений, до завершения транзакции *T1* никакая другая транзакция не должна иметь возможности изменять объект *A*. Отсутствие неповторяющихся чтений зачастую является максимальным требованием изолированности транзакций, хотя это еще не гарантирует реальной изолированности пользователей.

   Метод устранения: до завершения *T1* никакая другая транзакция не должна иметь возможность изменять объект *A*;

4. Наличие фиктивных элементов. Допустим, что транзакция *T1* извлекает множество всех записей, которые удовлетворяют

некоторому условию (например, записи всех поставщиков из Москвы). Затем транзакция *T2* вставляет новую строку, которая удовлетворяет тому же условию и успешно завершается. Если транзакция *T1* вновь повторит ту же операцию извлечения, что и раньше, то ею будет обнаружена ранее отсутствовавшая – «фиктивная» строка. Данная ситуация также противоречит идее изолированности пользователей и может возникнуть даже на третьем уровне изолированности транзакций. Для ее устранения требуется более высокий уровень синхронизации транзакций.

Возможности возникновения этих нарушений при различных уровнях изоляции транзакций приведены в таблице 1:

Таблица 1

Уровень изоляции	Потерянные изменения	Неаккуратное считывание	Неповторяемое считывание	Фиктивные элементы
READ UNCOMMITTED	Нет	Да	Да	Да
READ COMMITTED	Нет	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Нет	Да
SERIALIZABLE	Нет	Нет	Нет	Нет

Для обеспечения реальной изолированности транзакций в СУБД применяются метод сериализации транзакций. Сериализация транзакций – это определение последовательности выполнения транзакций, когда результат совместного выполнения транзакций эквивалентен результату последовательного выполнения этих же транзакций. Система, в которой поддерживается сериализация транзакций, обеспечивает реальную изолированность пользователей. Основная проблема состоит в выборе такого метода сериализации, который не слишком ограничивал бы их параллельность. Тривиальным решением является действительно последовательное выполнение транзакций. Но существуют ситуации, в которых можно выполнять операторы разных транзакций в любом порядке с сохранением сериальности. Примерами могут служить только читающие транзакции, а также транзакции, не конфликтующие по доступу к объектам базы данных.

2.5.4. Методы сериализации транзакций

Существуют два базовых подхода к сериализации транзакций: подход, основанный на синхронизационных захватах (блокировках) объектов базы данных, и подход, основанный на использовании временных меток. Суть обоих подходов состоит в обнаружении конфликтов транзакций и их устранении. Практические методы сериализации транзакций основывается на учете этих конфликтов. Для каждого из подходов имеются две

разновидности – пессимистическая и оптимистическая. При применении пессимистических методов (ориентированных на ситуации, когда конфликты возникают часто) конфликты распознаются и разрешаются немедленно при их возникновении (реальном или подразумеваемом). Оптимистические методы основываются на том, что результаты всех операций модификации базы данных сохраняются в рабочей памяти транзакций. Реальная модификация базы данных производится только на стадии фиксации транзакции. Тогда же проверяется, не возникают ли конфликты с другими транзакциями.

Наиболее распространенным является подход, основанный на соблюдении двухфазного протокола синхронизационных блокировок объектов БД. В общих чертах протокол состоит в том, что перед выполнением любой операции в транзакции *T* над объектом базы данных *r* от имени транзакции *T* запрашивается синхронизационный захват объекта *r* в режиме, зависящим от вида операции (захват по чтению или захват по записи).

Одним из наиболее чувствительных недостатков метода сериализации транзакций на основе синхронизационных захватов является возможность возникновения тупиков (deadlocks) между транзакциями. Приведем пример возникновения тупика между транзакциями *T1* и *T2*: транзакции *T*1 и *T2* установили монопольные захваты объектов *r1* и *r2*, соответственно; после этого *T1* требуется совместный захват *r2*, а *T2* - совместный захват *r1*. Ни одна из транзакций не может продолжаться, следовательно, монопольные захваты не будут сняты, а совместные - не будут удовлетворены. Поскольку тупики возможны, и никакого естественного выхода из тупиковой ситуации не существует, то эти ситуации необходимо обнаруживать и искусственно устранять. Разрушение тупика начинается с выбора в цикле транзакций так называемой транзакции-жертвы, т.е. транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций. Критерием выбора обычно является стоимость транзакции. Стоимость транзакции определяется на основе многофакторной оценки, в которую с разными весами входят время выполнения, число накопленных захватов, их приоритет и т.д. После выбора транзакции-жертвы выполняется откат этой транзакции, который может носить полный или частичный характер. При этом освобождаются блокировки объектов и, следовательно, может быть продолжено выполнение других транзакций. Естественно, такое насильственное устранение тупиковых ситуаций является нарушением принципа изолированности пользователей, которого невозможно избежать.

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций, основан на использовании

временных меток. Основная идея метода (у которого существует множество разновидностей) состоит в следующем: если транзакция *T1* началась раньше транзакции *T2*, то система обеспечивает такой режим выполнения, как если бы *T1* была целиком выполнена до начала *T2*. Для этого каждой транзакции *T* предписывается временная метка *t*, соответствующая времени начала *T*. При выполнении операции над объектом *r* транзакция *T* помечает его своей временной меткой и типом операции (чтение или изменение). Перед выполнением операции над объектом *r* транзакция *T1* проверяет, не закончилась ли транзакция *T*, пометившая этот объект. Если *T* закончилась, *T1* помечает объект *r* и выполняет свою операцию. Если транзакция *T* не завершилась, то *T1* проверяет конфликтность операций. Если операции неконфликтны, при объекте *r* остается или проставляется временная метка с меньшим значением, и транзакция *T1* выполняет свою операцию. Если операции *T1* и *T* конфликтуют, то при $t(T) > t(T1)$ (т.е. транзакция *T* является более «молодой», чем *T1*), производится откат *T*, и *T1* продолжает работу. Если же $t(T) < t(T1)$, то *T1* получает новую временную метку и начинается заново.

К недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования синхронизационных захватов. Это связано с тем, что конфликтность транзакций определяется более грубо. Кроме того, в распределенных системах не очень просто вырабатывать глобальные временные метки. Но все эти недостатки окупаются тем, что не нужно распознавать и устранять тупики, реализация чего в распределенных системах стоит очень дорого.

Контрольные вопросы и задания

1. Дайте характеристику технологий доступа к данным Microsoft.
2. Что такое распределенная БД.
3. Что подлежит распределению в распределенной СУБД.
4. В чем состоит отличие архитектуры файл-сервер от архитектуры клиент-сервер.
5. Чем отличается удаленное представление от удаленного доступа к данным в архитектуре клиент-сервер.
6. Назовите преимущества и недостатки распределенных БД.
7. Назовите основные способы организации распределенной БД с распределенным хранением данных.
8. Что такое федеративная РСУБД.

9. Перечислите способы решения проблем совместного доступа к общим данным в многопользовательских БД.
10. Перечислите виды блокировок и правила их совмещения.
11. Какой из уровней изоляции транзакций обеспечивает максимальную изолированность пользователей: READ UNCOMMITTED, REPEATABLE READ, READ COMMITTED, SERIALIZABLE.
12. Какие вы знаете случаи нарушения способности к упорядочению транзакций.
13. На каком уровне изоляции транзакций может происходить чтение «грязных данных».