Лекция 6. Многопользовательские БД. Распределенные БД. Архитектура клиент-сервер.

В данной лекции рассмотрены 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.

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 обмена данными **Dynamic Data Exchange (DDE)** для более динамичного режима обмена данными.

Однако он функционировал медленно и ненадежно, и на смену ему был разработан значительно более эффективный протокол связывания и внедрения объектов **Object Linking and Embedding (OLE).**

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, 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. 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, состоящей 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 г. в спецификации **OLE 2.0.**

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.

Open Database Connectivity (ODBC)

Технология открытого доступа к данным **Open Database Connectivity (ODBC)** была разработана фирмой MS 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 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 БД. 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, 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 БД 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, драйверы, 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.

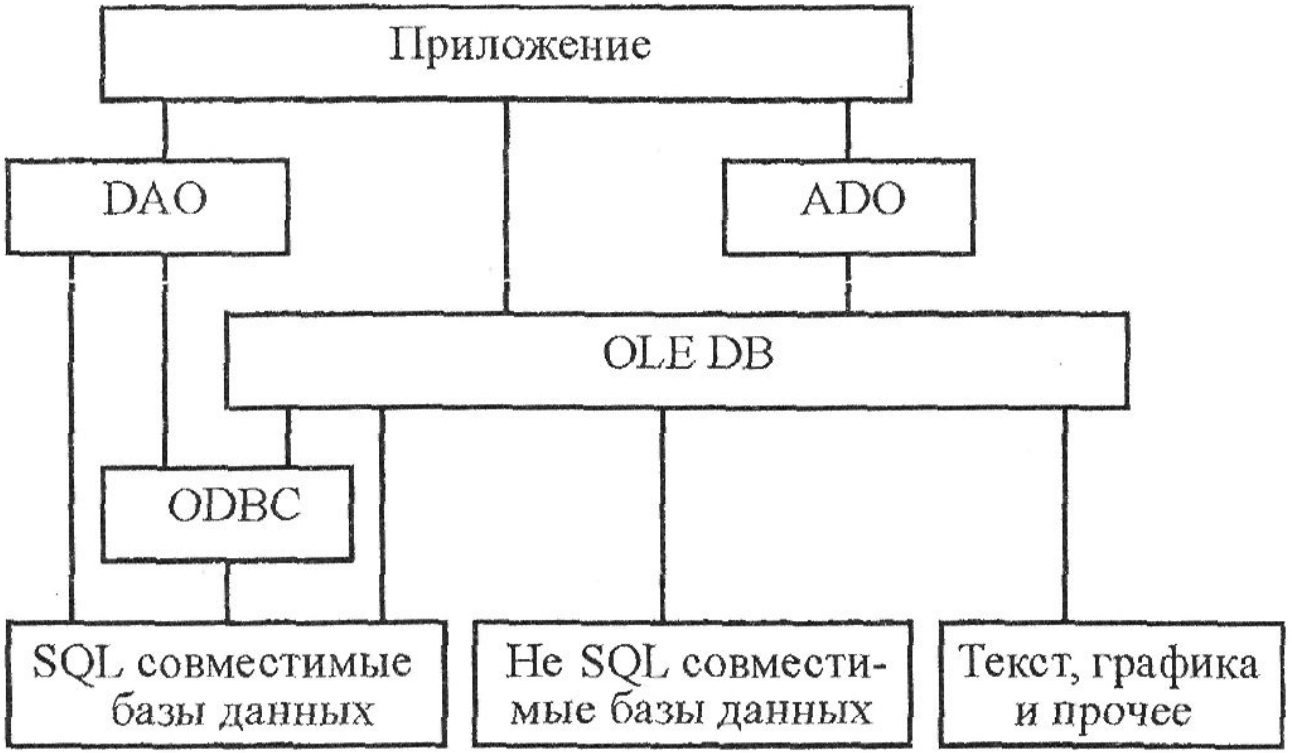
Реляционные БД - не единственный источник данных. Данные могут быть представлены в любом виде и формате. Например, в качестве данных могут- выступать объектно-ориентированные БД, электронные таблицы, документы в RTF, XML формате, почтовые системы и т. д. Соответственно возникла потребность либо создать единый формат хранения данных, что дорого и неэффективно, либо нарастить имеющиеся технологии интерфейсами доступа к любым типам данных. **Технология OLE DB** (Object Linking and Embedding Database) реализует это требование, являясь более универсальной нежели стандартные технологии OLE и COM.

Технология OLE DB

В технологии **OLE DB** используется механизм провайдеров, под которыми понимают поставщиков данных, находящихся в надстройке над физическим форматом данных. Провайдер OLE DB представляет собой компонент COM, позволяющий принимать вызовы OLE DB 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.

Хотя **ODBC и OLE DB** считаются хорошими интерфейсами передачи данных, но как программный интерфейс они имеют много ограничении, поскольку являются низкоуровневыми. Для снятия этих ограничений бы- ли 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 ODBC-провайдерам данных и характеризуется еще более высоким уровнем абстракции. Объектная модель **ADO** включает объекты, обеспечивающие соединение с провайдером данных, создание запросов SQL к данным, создание набора записей на основе запроса и т. д. Особенностью технологии **ADO** является возможность ее использования в Приложениях Интранет/Интернет для доступа к различным источникам данных. В целом **ADO** можно охарактеризовать как технологию разработки приложений для работы с распределенными БД по архитектуре клиент-сервер.

Архитектура доступа к данным MS



Архитектура файл-сервер, 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 представлено 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 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.**

БД с применением файл-серверной архитектуры

В БД с применением файл-серверной архитектуры по запросам пользователей файлы БД 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 БД можно легко занести ошибки, которые могут отразиться па всех пользователях. Если используемая СУБД не имеет достаточных средств для обеспечения многопользовательского доступа к данным или неправильно сконфигурирована, то нарушение целостности может произойти и при попытке одновременного изменения одних и тех же данных. Часто такие действия приводят 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, так как 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

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, БД, службы печати, почтовые службы. Тип сервера определяется видом ресурса, которым он управляет. Например, если управляемым ресурсом является БД, то соответствующий сервер называется сервером БД.

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 организуются 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 клиент 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:

1) удаленное представление (Database Server - DBS) ;

2) распределенная функция;

3) удаленный доступ к данным (Remote Data Access - RDA) .

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 функций и функции представления тесно взаимосвязаны. 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 реализуется 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, которые можно выполнять с помощью хранимых процедур и триггеров и сравнительно низкая эффективность использования вычислительных ресурсов обоих компьютеров.

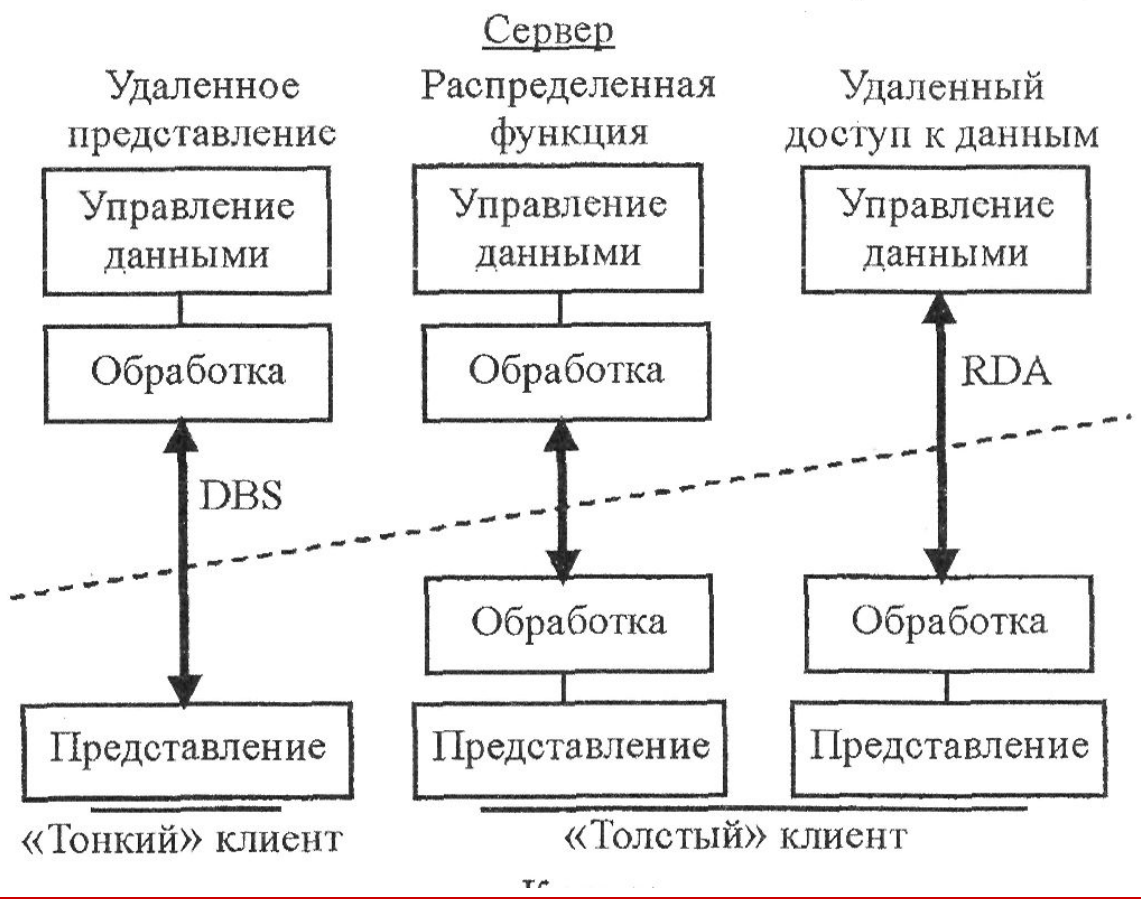
Распределенная функция

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 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 удаленном 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. 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, каждый из них предоставляет свой вид сервиса. Любая программа, запрашивая услугу у сервера приложений, является для него клиентом. На **третьем** уровне расположен удаленный сервер БД. Поскольку центральным звеном является сервер приложении, 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.

Модели архитектуры клиент-сервер



Распределенные базы данных (РБД)

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

Таким образом, мы можем определить **распределенную БД (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 БД.

Существуют два основных способа организации РБД с распределенным хранением данных: **фрагментация и репликация** (тиражирование). **Фрагментация 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 же БД. Безопасность и степень доступности данных в такой системе самые высокие. 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. 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 РБД, является **степень однородности**. Если все пользователи РБД используют одно и то же 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 распределенную СУБД 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 распределенную СУБД, которая хоть и имеет некоторую общую схему данных, но составлена из полностью автономных СУБД. Такая СУБД называется **федеративной СУБД (Federated Database Management System - FDBMS).**

Федеративная СУБД может быть даже составлена из СУБД, поддерживающих различные модели данных, типы, ограничения и языки манипулирования данными. Такой вариант распределенной СУБД более надежен и перспективен, 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. Она работает только с общей схемой данных, поскольку все локальные СУБД имеют свои собственные схемы данных и собственными силами обеспечивают доступ к данным всех их пользователей. Глобальная схема создается посредством слияния 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.

Глобальная концептуальная схема представляет собой логическое описание всех БД, предоставляя ее так, как будто она не является распределенной.

Схемы фрагментации, репликации и распределения определяют размещение данных по локальным компьютерам. Для различных типов распределенных СУБД. 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.

Основываясь на трехуровневой архитектуре БД, архитектура распределенной СУБД может быть представлена следующим образом.

РБД обладает следующими **преимуществами**.

1. Разделяемость и локальная автономность. Географическая распределеленность организации может быть отображена в распределении ее данных. В результате пользователи отдельных частей БД получают локальный контроль над данными и могут устанавливать локальные ограничеия 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 распределенная СУБД 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:

• повышение сложности, влекущее увеличение стоимости и срока разработки РБД;

• усложнение контроля за целостностью данных;

• усложнение контроля за безопасностью и секретностью данных, связанное с обеспечением защиты не только БД самих по себе, но и сетевых соединений.