ВВЕДЕНИЕ

Современные объемы хранимых данных, обязательные требования к их доступности и скорости обработки, динамика развития систем обуславливают важность исследования факторов, влияющих на качество баз данных, лежащих в основе современных информационных систем.

К концу 80-х годов возникли новые условия работы для БД: большие объемы информации возникают во многих местах (например, розничная торговля, полиграфическое и другие производства). Источником большого количества данных мог быть и центр, но к этим данным требуется быстрый доступ с периферии (географически распределенное производство, работающее по одному графику). К тому же данные могут запрашиваться и центром, и удаленными потребителями в удаленных местах. Имеется большое количество данных, которые используются в срочных запросах, чаще всего местного характера (продажа авиа- и железнодорожных билетов).

Во многих производствах, например, в компьютерном интегрированном полиграфическом производстве, необходимыми являются распределенные базы данных, связывающие в единое целое процесс управления комплексом различных технологических процессов. Здесь осуществляется работа не с одним приложением, а с системой приложений.

Централизованные БД, особенно построенные на классическом подходе, не могли удовлетворить новым требованиям.

Быстрое распространение сетей передачи данных, резкое увеличение объема внешней памяти ПК при ее удешевлении в 80-е годы способствовали широкому внедрению распределенных баз данных.

К достоинствам распределенных баз данных относятся:

1) соответствие структуры распределенных баз данных структуре организаций;

2) гибкое взаимодействие локальных БД;

3) широкие возможности централизации узлов;

4) непосредственный доступ к информации, снижение стоимости передач (за счет уплотнения и концентрации данных);

5) высокие системные характеристики (малое время отклика за счет распараллеливания процессов, высокая надежность);

6) модульная реализация взаимодействия, расширения аппаратных средств, возможность использования объектно-ориентированного подхода в программировании;

7) возможность распределения файлов в соответствии с их активностью;

8) независимые разработки локальных БД через стандартный интерфейс.

Вместе с тем распределенные базы данных обладают более сложной структурой, что вызывает появление дополнительных проблем (избыточность, несогласованность данных по времени, согласование процессов обновления и запросов, использования телекоммуникационных ресурсов, учет работы дополнительно подсоединенных локальных БД, стандартизация общего интерфейса) согласования работы элементов.

Серьезные проблемы возникают при интеграции в рамках распределенных баз данных однородных (гомогенных) локальных БД с одинаковыми, чаще всего реляционными, моделями данных.

Проблемы значительно усложняются, если локальные БД построены с использованием различных моделей данных (неоднородные, гетерогенные распределенные базы данных).

Целью данной курсовой работы является исследование распределенных баз данных и распределенных СУБД. Для достижения поставленной цели в работе были реализованы следующие задачи:

* Рассмотрено понятие распределенных баз данных;
* Рассмотрены свойства распределенных БД;
* Рассмотрено понятие целостности данных;
* Рассмотрен принцип построения распределенных баз данных на примере SYSTEM R\*
* Разработано приложение в среде Delphi.

1 РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ И БАЗЫ ДАННЫХ

* 1. **Понятие распределенных баз данных**

Под распределенной (Distributed DataBase - DDB) обычно подразумевают базу данных, включающую фрагменты из нескольких баз данных, которые располагаются на различных узлах сети компьютеров, и, возможно управляются различными СУБД. Распределенная база данных выглядит с точки зрения пользователей и прикладных программ как обычная локальная база данных. В этом смысле слово "распределенная" отражает способ организации базы данных, но не внешнюю ее характеристику. ("распределенность" базы данных невидима извне).

Основная задача систем управления распределенными базами данных состоит в обеспечении средства интеграции локальных баз данных, располагающихся в некоторых узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных как к единой базе данных [1].

При этом должны обеспечиваться:

* простота использования системы;
* возможности автономного функционирования при нарушениях связности сети или при административных потребностях;
* высокая степень эффективности.

Возможны однородные и неоднородные распределенные базы данных. В однородном случае каждая локальная база данных управляется одной и той же СУБД. В неоднородной системе локальные базы данных могут относиться даже к разным моделям данных. Сетевая интеграция неоднородных баз данных - это актуальная, но очень сложная проблема. Многие решения известны на теоретическом уровне, но пока не удается справиться с главной проблемой - недостаточной эффективностью интегрированных систем. Заметим, что более успешно практически решается промежуточная задача - интеграция неоднородных SQL-ориентированных систем. Понятно, что этому в большой степени способствует стандартизация языка SQL и общее следование производителей СУБД принципам открытых систем [2].

**1.2 Свойства распределенных баз данных**

Определение распределенных баз данных (DDB) предложил Дэйт (C.J. Date). Он установил 12 свойств или качеств идеальной DDB [4]:

* Локальная автономия (local autonomy)
* Независимость узлов (no reliance on central site)
* Непрерывные операции (continuous operation)
* Прозрачность расположения (location independence)
* Прозрачная фрагментация (fragmentation independence)
* Прозрачное тиражирование (replication independence)
* Обработка распределенных запросов (distributed query processing)
* Обработка распределенных транзакций (distributed transaction processing)
* Независимость от оборудования (hardware independence)
* Независимость от операционных систем (operationg system independence)
* Прозрачность сети (network independence)
* Независимость от баз данных (database independence)

Локальная автономия. Это качество означает, что управление данными на каждом из узлов распределенной системы выполняется локально. Будучи фрагментом общего пространства данных, БД , в то же время функционирует как полноценная локальная база данных; управление ею выполняется локально и независимо от других узлов системы [4].

Независимость от центрального узла. В идеальной системе все узлы равноправны и независимы, а расположенные на них базы являются равноправными поставщиками данных в общее пространство данных. База данных на каждом из узлов самодостаточна - она включает полный собственный словарь данных и полностью защищена от несанкционированного доступа.

Непрерывные операции. Это качество можно трактовать как возможность непрерывного доступа к данным (известное «24 часа в сутки, семь дней в неделю») в рамках DDB вне зависимости от их расположения и вне зависимости от операций, выполняемых на локальных узлах. Это качество можно выразить лозунгом «данные доступны всегда, а операции над ними выполняются непрерывно» [4].

Прозрачность расположения. Это свойство означает полную прозрачность расположения данных. Пользователь, обращающийся к DDB, ничего не должен знать о реальном, физическом размещении данных в узлах информационной системы. Все операции над данными выполняются без учета знаний их местонахождения. Транспортировка запросов к базам данных осуществляется встроенными системными средствами.

Прозрачная фрагментация. Это свойство трактуется как возможность распределенного (то есть на различных узлах) размещения данных, логически представляющих собой единое целое. Существует фрагментация двух типов: горизонтальная и вертикальная. Горизонтальная означает хранение строк одной таблицы на различных узлах (фактически, хранение строк одной логической таблицы в нескольких идентичных физических таблицах на различных узлах). Вертикальная означает распределение столбцов логической таблицы по нескольким узлам.

Рассмотрим пример, иллюстрирующий оба типа фрагментации. Имеется таблица employee (emp\_id, emp\_name, phone), определенная в базе данных на узле в Фениксе. Имеется точно такая же таблица, определенная в базе данных на узле в Денвере. Обе таблицы хранят информацию о сотрудниках компании. Кроме того, в базе данных на узле в Далласе определена таблица emp\_salary (emp\_id, salary). Тогда запрос «получить информацию о сотрудниках компании» может быть сформулирован так:

SELECT \* FROM employee@phoenix, employee@denver ORDER BY emp\_id

В то же время запрос "получить информацию о заработной плате сотрудников компании" будет выглядеть следующим образом:

SELECT employee.emp\_id, emp\_name, salary FROM employee@denver, employee@phoenix, emp\_salary@dallas ORDER BY emp\_id

Прозрачность тиражирования. Тиражирование данных - это асинхронный процесс переноса изменений объектов исходной базы данных в базы, расположенные на других узлах распределенной системы. В данном контексте прозрачность тиражирования означает возможность переноса изменений между базами данных средствами, невидимыми пользователю распределенной системы. Данное свойство означает, что тиражирование возможно и достигается внутрисистемными средствами.

Обработка распределенных запросов. Это свойство DDB трактуется как возможность выполнения операций выборки над распределенной базой данных, сформулированных в рамках обычного запроса на языке SQL. То есть операцию выборки из DDB можно сформулировать с помощью тех же языковых средств, что и операцию над локальной базой данных. Например,

SELECT customer.name, customer.address, order.number, order.date FROM customer@london, order@paris WHERE customer.cust\_number = order.cust\_number

Обработка распределенных транзакций. Это качество DDB можно трактовать как возможность выполнения операций обновления распределенной базы данных (INSERT, UPDATE, DELETE), не разрушающее целостность и согласованность данных. Эта цель достигается применением двухфазного протокола фиксации транзакций (two-phase commit protocol), ставшего фактическим стандартом обработки распределенных транзакций. Его применение гарантирует согласованное изменение данных на нескольких узлах в рамках распределенной транзакции.

Независимость от оборудования. Это свойство означает, что в качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей - от мэйнфреймов до "персоналок".

Независимость от операционных систем. Это качество вытекает из предыдущего и означает многообразие операционных систем, управляющих узлами распределенной системы.

Прозрачность сети. Доступ к любым базам данных осуществляется по сети. Спектр поддерживаемых конкретной СУБД сетевых протоколов не должен быть ограничением системы с распределенными базами данных. Данное качество формулируется максимально широко - в распределенной системе возможны любые сетевые протоколы.

Независимость от баз данных. Это качество означает, что в распределенной системе могут мирно сосуществовать СУБД различных производителей, и возможны операции поиска и обновления в базах данных различных моделей и форматов.

Исходя из определения Дэйта, можно рассматривать DDB как слабосвязанную сетевую структуру, узлы которой представляют собой локальные базы данных. Локальные базы данных автономны, независимы и самоопределены; доступ к ним обеспечиваются СУБД, в общем случае от различных поставщиков. Связи между узлами - это потоки тиражируемых данных. Топология DDB варьируется в широком диапазоне - возможны варианты иерархии, структур типа «звезда» и т.д. В целом топология DDB определяется географией информационной системы и направленностью потоков тиражирования данных [4].

**1.3 Целостность данных**

В DDB поддержка целостности и согласованности данных, ввиду свойств 1-2, представляет собой сложную проблему. Ее решение - синхронное и согласованное изменение данных в нескольких локальных базах данных, составляющих DDB - достигается применением протокола двухфазной фиксации транзакций. Если DDB однородна - то есть на всех узлах данные хранятся в формате одной базы и на всех узлах функционирует одна и та же СУБД, то используется механизм двухфазной фиксации транзакций данной СУБД. В случае же неоднородности DDB для обеспечения согласованных изменений в нескольких базах данных используют менеджеры распределенных транзакций. Это, однако, возможно, если участники обработки распределенной транзакции - СУБД, функционирующие на узлах системы, поддерживают XA-интерфейс, определенный в спецификации DTP консорциума X/Open. В настоящее время XA-интерфейс имеют СУБД CA-OpenIngres, Informix, Microsoft SQL Server, Oracle, Sybase.

Если в DDB предусмотрено тиражирование данных, то это сразу предъявляет дополнительные жесткие требования к дисциплине поддержки целостности данных на узлах, куда направлены потоки тиражируемых данных. Проблема в том, что изменения в данных инициируются как локально - на данном узле - так и извне, посредством тиражирования. Неизбежно возникают конфликты по изменениям, которые необходимо отслеживать и разрешать [4].

Обработка распределенных запросов. Обработка распределенных запросов (Distributed Query -DQ) - задача, более сложная, нежели обработка локальных и она требует решения с помощью особого компонента - оптимизатора DQ. Обратимся к базе данных, распределенной по двум узлам сети. Таблица detail хранится на одном узле, таблица supplier - на другом. Размер первой таблицы - 10000 строк, размер второй - 100 строк (множество деталей поставляется небольшим числом поставщиков). Результирующая таблица представляет собой объединение таблиц detail и supplier. Данный запрос - распределенный, так как затрагивает таблицы, принадлежащие различным локальным базам данных. Для его нормального выполнения необходимо иметь обе исходные таблицы на одном узле. Следовательно, одна из таблиц должна быть передана по сети. Очевидно, что это должна быть таблица меньшего размера, то есть таблица supplier. Следовательно, оптимизатор DQ запросов должен учитывать такие параметры, как, в первую очередь, размер таблиц, статистику распределения данных по узлам, объем данных, передаваемых между узлами, скорость коммуникационных линий, структуры хранения данных, соотношение производительности процессоров на разных узлах и т.д. От интеллекта оптимизатора DQ впрямую зависит скорость выполнения распределенных запросов.

Межоперабельность. В контексте DDB межоперабельность означает две вещи.

Во-первых, - это качество, позволяющее обмениваться данными между базами данных различных поставщиков.

Во-вторых, это возможность некоторого унифицированного доступа к данным в DDB из приложения. Возможны как универсальные решения (стандарт ODBC), так и специализированные подходы. Очевидный недостаток ODBC - недоступность для приложения многих механизмов каждой конкретной СУБД, поскольку они могут быть использованы в большинстве случаев только через расширения SQL в диалекте языка данной СУБД, но в стандарте ODBC эти расширения не поддерживаются.   
Специальные подходы - это, например, использование шлюзов, позволяющее приложениям оперировать над базами данных в "чужом" формате так, как будто это собственные базы данных. Вообще, цель шлюза - организация доступа к унаследованным (legacy) базам данных и служит для решения задач согласования форматов баз данных при переходе к какой-либо одной СУБД. Следовательно, шлюзы можно рассматривать как средство, облегчающее миграцию, но не как универсальное средство межоперабельности в распределенной системе. Вообще, универсального рецепта решения задачи межоперабельности в этом контексте не существует - все определяется конкретной ситуацией, историей информационной системы и массой других факторов [2].

Технология тиражирования данных. Тиражирование данных (Data Replication - DR) - это асинхронный перенос изменений объектов исходной базы данных в базы, принадлежащие различным узлам распределенной системы. Функции DR выполняет, как правило, специальный модуль СУБД - сервер тиражирования данных, называемый репликатором (так устроены СУБД CA-OpenIngres и Sybase). В Informix-OnLine Dynamic Server репликатор встроен в сервер, в Oracle 7 для использования DR необходимо приобрести дополнительно к Oracle7 DBMS опцию Replication Option [7].

Детали тиражирования данных полностью скрыты от прикладной программы. В этом, собственно, состоит качество 6 в определении Дэйта. Синхронное обновление DDB и DR-технология - в определенном смысле антиподы. Краеугольный камень первой - синхронное завершение транзакций одновременно на нескольких узлах распределенной системы. Ee "Ахиллесова пята" - жесткие требования к производительности и надежности каналов связи. Если база данных распределена по нескольким территориально удаленным узлам, объединенным медленными и ненадежными каналами связи, а число одновременно работающих пользователей составляет сотни и выше, то вероятность того, что распределенная транзакция будет зафиксирована в обозримом временном интервале, становится чрезвычайно малой. В таких условиях (характерных, кстати, для большинства отечественных организаций) обработка распределенных данных практически невозможна.  
DR-технология не требует синхронной фиксации изменений, и в этом ее сильная сторона. Можно накапливать изменения в данных в виде транзакций в одном узле и периодически копировать эти изменения на другие узлы.

Преимущества DR-технологии. Во-первых, данные всегда расположены там, где они обрабатываются - следовательно, скорость доступа к ним существенно увеличивается. Во-вторых, передача только операций, изменяющих данные (а не всех операций доступа к удаленным данным позволяет значительно уменьшить трафик. В-третьих, со стороны исходной базы для принимающих баз репликатор выступает как процесс, инициированный одним пользователем, в то время как в физически распределенной среде с каждым локальным сервером работают все пользователи распределенной системы, конкурирующие за ресурсы друг с другом. Наконец, в-четвертых, никакой продолжительный сбой связи не в состоянии нарушить передачу изменений. Дело в том, что тиражирование предполагает буферизацию потока изменений (транзакций); после восстановления связи передача возобновляется с той транзакции, на которой тиражирование было прервано [7].

DR-технология данных не лишена недостатков. Например, невозможно полностью исключить конфликты между двумя версиями одной и той же записи. Он может возникнуть, когда вследствие асинхронности два пользователя на разных узлах исправят одну и ту же запись в тот момент, пока изменения в данных из первой базы данных еще не были перенесены во вторую. При проектировании распределенной среды с использованием DR-технологии необходимо предусмотреть конфликтные ситуации и запрограммировать репликатор на какой-либо вариант их разрешения. В этом смысле применение DR-технологии - наиболее сильная угроза целостности DDB.

Прозрачность расположения. Это качество DDB в реальных продуктах должно поддерживаться соответствующими механизмами. Разработчики СУБД придерживаются различных подходов. Рассмотрим пример из Oracle. Допустим, что DDB включает локальную базу данных, которая размещена на узле в Лондоне. Создадим вначале ссылку (database link), связав ее с символическим именем (london\_unix), транслируемым в IP-адрес узла в Лондоне.

CREATE PUBLIC DATABASE LINK london.com CONNECT TO london\_unix USING oracle\_user\_ID;

Теперь мы можем явно обращаться к базе данных на этом узле, запрашивая, например, в операторе SELECT таблицу, хранящуюся в этой базе:

SELECT customer.cust\_name, order.order\_date FROM customer@london.com, order WHERE customer.cust\_number = order.cust\_number;

Очевидно, однако, что мы написали запрос, зависящий от расположения базы данных, поскольку явно использовали в нем ссылку. Определим customer и customer@london.com как синонимы:

CREATE SYNONYM customer FOR customer@london.com;

и в результате можем написать полностью независимый от расположения базы данных запрос:

SELECT customer.cust\_name, order.order\_date FROM customer, order WHERE customer.cust\_number = order.cust\_number

Задача решается с помощью оператора SQL CREATE SYNONYM, который позволяет создавать новые имена для существующих таблиц. При этом оказывается возможным обращаться к другим базам данных и к другим компьютерам.

Во многих СУБД задача управления именами объектов DDB решается путем использования глобального словаря данных, хранящего информацию о DDB: расположение данных, возможности других СУБД (если используются шлюзы), сведения о скорости передачи по сети с различной топологией и т.д.

#### 1.4 Распределенная система управления базами данных System R\*

#### 1.4.1 Основная цель проекта System R

#### Мы ограничимся рассмотрением проблем однородных распределенных СУБД на примере System R\*.

Основную цель проекта можно сформулировать следующим образом: обеспечить средства интеграции локальных баз данных System R, располагающихся в узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим базам данных так, как если бы они были централизованы. При этом должны обеспечиваться:

* легкость использования системы;
* возможности автономного функционирования при нарушениях связности сети или при административных потребностях;
* высокая степень эффективности.

Для решения этих проблем было необходимо принять ряд проектных решений, касающихся декомпозиции исходного запроса, оптимального выбора способа выполнения запроса, согласованного выполнения транзакций, обеспечения синхронизации, обнаружения и разрешения распределенных тупиков, восстановления состояния баз данных после разного рода сбоев узлов сети.

Легкость использования системы достигается за счет того, что пользователи System R\* (разработчики прикладных программ и конечные пользователи) остаются в среде языка SQL, т.е. могут продолжать работать в тех же внешних условиях, что и в System R (и SQL/DS и DB2). Возможность использования SQL основывается на обеспечении System R\* прозрачности местоположения данных. Система автоматически обнаруживает текущее местоположение упоминаемых в запросе пользователя объектов данных; одна и та же прикладная программа, включающая предложения SQL, может быть выполнена в разных узлах сети. При этом в каждом узле сети на этапе компиляции запроса выбирается наиболее оптимальный план выполнения запроса в соответствии с расположением данных в распределенной системе.

Обеспечению автономности узлов сети в System R\* уделяется очень большое внимание. Каждая локальная база данных администрируется независимо от других. Возможны автономное подключение новых пользователей, смена версии автономной части системы и т.д. Система спроектирована таким образом, что в ней не требуются централизованные службы именования объектов или обнаружения тупиков. В индивидуальных узлах не требуется наличие глобального знания об операциях, выполняющихся в других узлах сети; работа с доступными базами данных может продолжаться при выходе из строя отдельных узлов сети или линий связи [7].

**1.4.2 Средства повышения эффективности**

Высокая степень эффективности системы является одним из наиболее ключевых требований к распределенным системам управления базами данных вообще и к System R\* в частности. Для достижения этой цели используются два основных приема.

Во-первых, как и в System R, в System R\* выполнению запроса предшествует его компиляция. В ходе этого процесса производится поиск употребляемых в запросе имен объектов баз данных в распределенном каталоге и замена имен на внутренние идентификаторы; проверка прав доступа пользователя, от имени которого производится компиляция, на выполнение соответствующих операций над базами данных и выбор наиболее оптимального глобального плана выполнения запроса, который затем подвергается декомпозиции и по частям рассылается в соответствующие узлы сети, где производится выбор оптимальных локальных планов выполнения компонентов запроса и происходит генерация модулей доступа в машинных кодах. В результате множество действий производится на стадии компиляции до реального выполнения запроса. Обработанная посредством прекомпилятора System R\* прикладная программа, включающая предложения SQL, может в дальнейшем выполняться много раз без дополнительных накладных расходов. Использование распределенного каталога, распределенная компиляция и оптимизация запросов являются наиболее интересными и оригинальными аспектами проекта System R\*.

Вторым средством повышения эффективности системы является возможность перемещения удаленных отношений в локальную базу данных. Диалект SQL, используемый в System R\*, включает предложение MIGRATE TABLE, при выполнении которого указанное отношение переносится в локальную базу данных. Это средство, находящееся в распоряжении пользователей, конечно, в ряде случаев может помочь добиться более эффективного прохождения транзакций. Естественно, как и для всех операций, операция MIGRATE по отношению к указанному отношению доступна не любому пользователю, а лишь тем, которые обладают соответствующим правом.

Прежде, чем перейти к более детальному изложению наиболее интересных аспектов реализации System R\*, упомянем некоторые средства, которые разработчики этой системы предполагали реализовать на начальной стадии проекта, но которые реализованы не были (причем некоторые из них, видимо, и не будут никогда реализованы). Предполагалось иметь в системе средства горизонтального и вертикального разделения отношений распределенной базы данных, средства дублирования отношений в нескольких узлах с поддержкой согласованности копий и средства поддержания мгновенных снимков состояния баз данных в соответствии с заданным запросом.

Для задания горизонтального разделения отношений в SQL была введена конструкция вида

DISTRIBUTE TABLE <table-name> HORIZONTALLY INTO

<name> WHERE <predicate> IN SEGMENT <segment-name site>

<name> WHERE <predicate> IN SEGMENT <segment-name site>

При выполнении предложения такого типа указанное отношение разбивалось на ряд подотношений, содержащих кортежи, удовлетворяющие соответствующему предикату из раздела WHERE, и каждое полученное таким образом подотношение посылалось в казанный узел для хранения в сегменте с указанным именем. Гарантируется согласованное состояние разделов при изменении отношения.

Вертикальное разделение производилось с помощью оператора

DISTRIBUTE TABLE <table-name> VERTICALLY INTO

<name> WHERE <column-name-list> IN SEGMENT <segment-name site>

<name> WHERE <column-name-list> IN SEGMENT <segment-name site>

При выполнении такого предложения также образовывался набор подотношений с помощью проекции заданного отношения на атрибуты из заданного списка. Каждое полученное подотношение затем посылалось для хранения в сегменте с указанным именем в соответствующий узел. После этого система ответственна за поддержание согласованного состояния образованных разделов [7].

Горизонтальное и вертикальное разделение отношений реально не используются в System R\*, хотя очевидно, что выполнение собственно оператора DISTRIBUTE никаких технических трудностей не вызывает. Трудности возникают при обеспечении согласованности разделов (смотри ниже). Кроме того, разделенные отношения очень трудно использовать. В соответствии с идеологией системы учет наличия разделов отношения в разных узлах сети должен производить оптимизатор, т.е. количество потенциально возможных планов выполнения запросов, которые должны оцениваться оптимизатором, еще более возрастает. При том, что в распределенной системе число возможных планов и так очень велико, и оптимизатор работает на пределе сложности, разумным образом использовать разделенные отношения невозможно. Разработчики оптимизатора System R\* не были в состоянии учитывать разделенность отношений. Поэтому и вводить в систему разделенные отношения пока бессмысленно.

Для задания требования поддержки копий отношения в нескольких узлах сети предлагалось использовать новую конструкцию SQL

DISTRIBUTE TABLE <table-name> REPLICATED INTO

<name> IN SEGMENT <segment-name site>

<name> IN SEGMENT <segment-name site>

При выполнении такого предложения должна была производиться рассылка копий указанного отношения для хранения в именованных сегментах указанных узлов сети. Система должна автоматически поддерживать согласованность копий.

Как и в случае разделенных отношений, кроме существенных проблем поддержания согласованности копий, проблемой является и разумное использование копий, наличие которых должно было бы учитываться оптимизатором.

Создание мгновенного снимка состояния баз данных в соответствии с заданным запросом на выборку должно было производиться с использованием новой конструкции SQL.

DEFINE SNAPSHOT <snapshot-name> (<attribute-list>)

AS <query>

REFRESHED EVERY <period>

При выполнении предложения фактически производится выполнение указанного в нем запроса на выборку, а результирующее отношение сохраняется под указанным в предложении именем в локальной базе данных в том узле, в котором выполняется предложение. После этого мгновенный снимок периодически обновляется в соответствии с запомненным запросом.

Можно обновить мгновенный снимок, не дожидаясь истечения временного интервала, указанного в определении, путем выполнения предложения REFRESH SNAPSHOT <snapshot-name>.

Разумное использование мгновенных снимков более реально, чем использование разделенных отношений и копированных отношений, поскольку их можно в некотором смысле рассматривать как материализованные представления базы данных. Имя мгновенного снимка можно было бы использовать прямо в запросе на выборку там, где можно использовать имена базовых отношений или представлений. Большие проблемы связаны с обновлением отношений через их мгновенные снимки, поскольку в момент обновления содержимое мгновенного снимка может расходиться с текущим содержимым базового отношения [7].

По отношению к мгновенным снимкам проблем поддержания согласованного состояния мгновенного снимка и базовых отношений не существует, поскольку автоматическое согласование не требуется. Что же касается разделенных отношений и раскопированных отношений, то для них эта проблема общая и достаточно трудная. Во-первых, согласование разделов и копий вызывает существенные накладные расходы при выполнении операций модификации хранимых отношений. Для этого требуется выработка и соблюдение специальных протоколов модификации.

Во-вторых, введение копированных отношений обычно производится не столько для увеличения эффективности системы, сколько для увеличения доступности данных при нарушении связности сети. В системах, в которых применяется этот подход, при нарушении связности сети работа с распределенной базой данных обычно продолжается только в одной из образовавшихся подсетей. При этом для выбора подсети используются алгоритмы голосования; решение принимается на основе учета количества связных узлов сети. Применяются и другие подходы, но все они очень дорогостоящие, а самое главное, они плохо согласуются с базовым подходом System R\* по поводу выбора способа выполнения запроса на стадии его компиляции. Поэтому, как нам кажется, в System R\* никогда не будут реализованы средства, позволяющие тем или иным способом поддерживать копии отношений в нескольких узлах сети.

Далее мы рассмотрим аспекты проекта System R\*, которые нашли отражение в ее реализации и являются на наш взгляд наиболее интересными: средства именования объектов и организацию распределенного каталога баз данных; подход к распределенным компиляции и выполнению запросов; особенности использования представлений; средства оптимизации запросов; особенности управления транзакциями; средства синхронизации и распределенный алгоритм обнаружения синхронизационных тупиков.

##### 1.4.3 Именование объектов и организация распределенного каталога

Напомним прежде всего, что полное имя отношения (базового или представления) в базе данных System R имеет вид имя-пользователя.имя-отношения, где имя-пользователя идентифицирует пользователя - создателя отношения, а имя-отношения - это то имя, которое было указано в предложениях CREATE TABLE или CREATE VIEW. В запросах можно указывать либо это полное имя отношения, либо его локальное имя. Во втором случае при компиляции используются стандартные правила дополнения локального имени до полного с использованием в качестве составляющей имя-пользователя идентификатора пользователя, от имени которого выполняется компиляция.

В System R\* используется развитие этого подхода. Системное имя отношения включает четыре компонента: идентификатор пользователя-создателя отношения; идентификатор узла сети, в котором выполнялась операция создания отношения; локальное имя отношения, присвоенное ему при создании; идентификатор узла, в котором отношение располагалось непосредственно после своего создания (напомним, что отношение может перемещаться из одного узла в другой при выполнении операции MIGRATE).

В запросе на SQL можно использовать системные имена объектов, но разрешается использовать и короткие локальные имена (либо локальное имя, квалифицированное именем пользователя). При этом возможны две интерпретации локального имени. Оно может интерпретироваться как часть системного имени, и в этом случае по умолчанию дополняется до системного, исходя из идентификатора узла, в котором производится компиляция, и идентификатора пользователя, от имени которого она производится (если имя пользователя не указано явно). Вторая возможная интерпретация локального имени заключается в рассмотрении его как имени ранее определенного синонима системного имени.

Для определения синонимов SQL расширен оператором вида:

DEFINE SYNONYM <relation-name> AS <system-wide-name>.

При выполнении такого предложения в локальный каталог заносится соответствующая информация.

Таким образом, при компиляции запроса всегда можно определить системные имена всех употребляемых в нем отношений: либо они явно указаны, либо могут быть получены на основе информации из локальных отношений-каталогов.

Концепция распределенного каталога System R\* основана на наличии у каждого объекта распределенной базы данных уникального системного имени. Принято следующее соглашение: информация о размещении любого объекта базы данных (идентификатор текущего узла, в котором размещен объект) сохраняется в локальном каталоге того узла, в котором объект располагался непосредственно после создания (родового узла) [7].

Следовательно, для получения полной информации об отношении в общем случае необходимо сначала воспользоваться локальным каталогом узла, в котором происходит компиляция, затем обратиться к удаленному каталогу родового узла данного отношения и в заключение воспользоваться каталогом текущего узла. Таким образом, для получения точной системной информации о любом отношении распределенной базы данных может потребоваться самое большее два удаленных доступа к отношениям-каталогам.

Применяется некоторая оптимизация этой процедуры. В локальном каталоге узла могут храниться копии элементов каталога других узлов (своего рода кэш-каталог). Согласованность копий элементов каталога не поддерживается. Эта информация используется на первой стадии компиляции запроса (мы рассматриваем распределенную компиляцию в следующем подразделе), а затем, на второй стадии, если информация, касающаяся некоторого объекта, оказалась неточной, она уточняется на основе локального каталога того узла, в котором объект хранится в настоящее время. Обнаружение некорректности копии элемента каталога производится за счет наличия при каждом элементе каталога номера версии. Если учесть достаточную инерционность системной информации, эта оптимизация может оказаться существенной.

##### 1.4.4 Распределенная компиляция запросов

Как мы уже отмечали, запросы на языке SQL до своего реального выполнения подвергаются компиляции. Как и в случае System R компиляция запроса может производиться на стадии прекомпиляции прикладной программы, написанной на традиционном языке программирования (PL/1, Cobol, ассемблер) с включением предложений SQL, или в динамике выполнения транзакции при выполнении предложения PREPARE. С точки зрения пользователей процесс компиляции в System R\* приводит к тем же результатам, что и в System R: для каждого предложения SQL образуется программа к машинных кодах (секция модуля доступа), вызовы которой помещаются в текст исходной прикладной программы.

Однако, в действительности процесс компиляции запроса в System R\* намного более сложен, чем в System R, что и естественно по причине гораздо более сложных сетевых взаимодействий, которые потребуются при реальном выполнении транзакции. Распределенная компиляция запросов в System R\* включает множество технических ухищрений и тонкостей. Мы не будем касаться их всех в этой статье по причинам недостатка информации и ограниченности объема. Рассмотрим только общую схему распределенной компиляции.

Будем называть главным узлом тот узел сети, в котором инициирован процесс компиляции предложения SQL, и дополнительными узлами - те узлы, которые вовлекаются в этот процесс в ходе его выполнения. На самом грубом уровне процесс компиляции можно разбить на следующие фазы:

1. В главном узле производится грамматический разбор предложения SQL с построением внутреннего представления запроса в виде дерева. На основе информации из локального каталога главного узла и удаленных каталогов дополнительных узлов производится замена имен объектов, фигурирующих в запросе, на их системные идентификаторы.
2. В главном узле генерируется глобальный план выполнения запроса, в котором учитывается лишь порядок взаимодействий узлов при реальном выполнении запроса. Для выработки глобального плана используется расширение техники оптимизации, применяемой в System R. Глобальный план отображается в преобразованном соответствующим образом дереве запроса.
3. Если в глобальном плане выполнения запроса участвуют дополнительные узлы, производится его декомпозиция на части, каждую из которых можно выполнить в одном узле (например, локальная фильтрация отношения в соответствии с заданным в условии выборки предикате ограничения). Соответствующие части запроса (во внутреннем представлении) рассылаются в дополнительные узлы.
4. В каждом узле, участвующем в глобальном плане выполнения запроса (главном и дополнительных) выполняется завершающая стадия выполнения компиляции. Эта стадия включает, по существу, две последние фазы процесса компиляции запроса в System R: оптимизацию и генерацию машинных кодов. Производится проверка прав пользователя, от имени которого производится компиляция, на выполнение соответствующих действий; происходит обработка представлений базы данных (здесь имеются тонкости, связанные с тем, что представления могут включать удаленные отношения; ниже мы еще остановимся на этом, а пока будем считать, что в запросе употребляются только имена базовых отношений); осуществляется локальная оптимизация обрабатываемой части запроса в соответствии с имеющимися индексами; наконец, производится генерация кода [7].

##### 1.4.5 Управление транзакциями и синхронизация

Выполнение транзакции в распределенной системе управления базами данных System R\*, естественно, является распределенным. Транзакция начинается в главном узле при обращении к какой-либо секции ранее подготовленного (на этапе компиляции) модуля доступа. Как и в System R, модуль доступа загружается в виртуальную память задачи, обращение к секции модуля доступа - это вызов подпрограммы. Однако, в отличие от System R, эта подпрограмма, кроме своего локального программного кода и вызовов функций RSS, содержит еще и вызовы удаленных подсекций модуля доступа. Эти вызовы интерпретируются в духе вызовов удаленных процедур. Тем самым выполнение одной транзакции, инициированной в некотором узле сети A влечет, вообще говоря, инициирование транзакций в дополнительных узлах. Основной новой по сравнению с System R проблемой является проблема согласованного завершения распределенной транзакции, чтобы результаты ее выполнения во всех затронутых ею узлах были либо отображены в состояние локальных баз данных, либо полностью отсутствовали.

Для достижения этой цели в System R\* используется двухфазный протокол завершения распределенной транзакции. Этот протокол является общеупотребимым в распределенных системах баз данных и описан во многих литературных источниках. Поэтому мы здесь опишем его очень кратко и неформально.

Для описания протокола используется следующая модель. Имеется ряд независимых транзакций-участников распределенной транзакции, выполняющихся под управлением транзакции-координатора. Решение об окончании распределенной транзакции принимается координатором. После этого выполняется первая фаза завершения транзакции, когда координатор передает каждому из участников сообщение "подготовиться к завершению". Получив такое сообщение, каждый участник переходит в состояние готовности как к немедленному завершению транзакции, так и к ее откату. В терминах System R\* это означает, что буфер журнала с записями об изменениях базы данных участника выталкиваются на внешнюю память, но синхронизационные захваты не снимаются. После этого каждый участник, успешно выполнивший подготовительные действия, посылает координатору сообщение «готов к завершению». Если координатор получает такие сообщения ото всех участников, то он начинает вторую фазу завершения, рассылая всем участникам сообщение «завершить транзакцию», и это считается завершением распределенной транзакции. Если не все участники успешно выполнили первую фазу, то координатор рассылает всем участникам сообщение «откатить транзакцию», и тогда эффект воздействия распределенной транзакции на состояние баз данных отсутствует.

По отношению к особенностям реализации двухфазного протокола завершения транзакции в System R\* заметим еще следующее. В качестве координатора выступает транзакция, выполняющаяся в главном узле, т.е. та, по инициативе которой возникли дополнительные транзакции. Тем самым, наличие центрального координирующего узла не требуется, что соответствует требованию автономности узлов. Для откатов транзакций используется базовый механизм точек сохранения System R. Наконец, классический протокол двухфазного завершения оптимизирован, чтобы сократить число необходимых сообщений.

Как и в System R, согласованность состояния баз данных при параллельном выполнении нескольких транзакций в System R\* обеспечивается на основе механизма синхронизационных захватов объектов базы данных при соблюдении двухфазного протокола захватов. Напомним, что это означает разбиение каждой транзакции с точки зрения синхронизации на две фазы - рабочую фазу, на которой захваты только устанавливаются, и фазу завершения, когда все захваты объектов базы данных, произведенные данной транзакцией, снимаются. Синхронизация производится в точности так же, как и в System R: каждая транзакция-участник обращается к локальной базе данных через RSS своего узла. Основной новой проблемой является проблема возможных распределенных тупиков, которые могут возникнуть между несколькими распределенными транзакциями, выполняющимися параллельно. (Тупики между транзакциями - участниками одной распределенной транзакции невозможны, поскольку все участники получают один общий идентификатор транзакции и не конфликтуют по синхронизации). Для обнаружения распределенных синхронизационных тупиков в System R\* применяется оригинальный распределенный алгоритм, не нарушающий требования автономности узлов сети и минимизирующий число передаваемых по сети сообщений и необходимую процессорную обработку.

Основная идея алгоритма состоит в том, что в каждом узле периодически производится анализ на предмет существования тупика с использованием информации о связях транзакций по ожиданию ресурсов, локальной в данном узле и полученной от других узлов. При проведении этого анализа обнаруживаются либо циклы ожиданий, что означает наличие тупика, либо потенциальные циклы, которые необходимо уточнить в других узлах. Эти потенциальные циклы представляются в виде специального вида строк. Строка представляет собой по сути дела список транзакций. Все транзакции упорядочены в соответствии со значениями своих идентификаторов («номеров транзакций»). Строка передается для дальнейшего анализа в следующий узел (узел, в котором выполняется самая правая в строке транзакция) только в том случае, если номер первой транзакции в строке меньше номера последней транзакции. (Это оптимизация, уменьшающая число передаваемых по сети сообщений). Этот процесс продолжается до обнаружения тупика.

Если обнаруживается наличие синхронизационного тупика, он разрушается за счет уничтожения (отката) одной из транзакций, входящей в цикл. В качестве жертвы выбирается транзакция, выполнившая к этому моменту наименьший объем работы. Эта информация также передается по сети вместе со строками, описывающими связи транзакций по ожиданию.

2 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ НА ПРИМЕРЕ «АРМ КЛАДОВЩИКА»

* 1. **Обследование предметной области, выявление запросов пользователей и построение концептуальной информационной модели ПО**

Анализ приведенных ниже объектов и атрибутов позволяет выделить сущности проектируемой базы данных, приняв решение о создании реляционной базы данных, можно построить ее модель.

* Каждая таблица проектируемой базы данных должна содержать информацию на отдельную тему, а каждое поле таблицы – содержать сведения по теме таблицы.

Проектирование БД заключается в определении состава полей ее таблиц и связей между таблицами. От того, насколько тщательно проведен анализ и насколько грамотно спроектирована БД, в существенной мере зависит эффективность будущей СУБД и ее полезность для пользователя.

Предметная область при проектировании базы данных склад. Автоматизации подлежит задача «Учет поступления и отпуска товаров» и решается с целью получения актуальной информации о выдаче товара со склада по заказам клиентов, поступления товаров на склад от постащиков, о клиентах, поставщиках и товарах.

В результате решения задачи предоставляются следующие выходные документы:

- «Инвентаризационная ведомость»

- «Договор поставки»

В предметной области сформируем запросы, необходимые для решения задачи:

1.Кто является получателем товара?

2.Какие товары хранятся на складе?

3.Кто работает на складе?

4.Кто поставляет товар?

Для удобства работы с атрибутами введем их идентификаторы. Наименование атрибута и идентификатор каждого используемого в дальнейшем атрибута приведен ниже (см. таблицу 1).

Таблица 1

Атрибуты и их идентификаторы

|  |  |  |
| --- | --- | --- |
| № | Наименование атрибута | Идентификатор |
| 1 | Табельный номер кладовщика | ТН |
| 2 | ФИО кладовщика | ФИО |
| № | Домашний адрес кладовщика | АДРЕС |
| 3 | Телефон | ТЕЛ |
| 4 | Дата рождения | ДАТА\_РОЖ |
| 5 | РНН | РНН |
| 6 | Стаж | СТАЖ |
| 7 | Оклад | ОКЛАД |
| 8 | Должность | ДОЛЖ |
| 9 | ФИО клиента | КЛИЕНТ |
| 10 | Банковский счет клиента | БАНК |
| 11 | Адрес клиента | АДР\_КЛ |
| 12 | Фирма клиента | НАЗ\_ФИРМ |
| 13 | Номер заказа | НЗ |
| 14 | Телефон клиента | ТЕЛ\_КЛ |
| 15 | Срок поставки товара | СРОК |
| 16 | Количество поставки товара | КОЛ\_ВО |
| 17 | Номенклатурный номер товара | ННТОВ |
| 18 | Цена товара отпускная | ЦЕНА\_ПОС |
| 19 | Наименование товара | НАЗ\_ТОВ |
| 20 | Стоимость товара | СТОИМ |
| 21 | Стоимость без НДС | БЕЗ\_НДС |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| № | Наименование атрибута | Идентификатор |
| 22 | Количество товара на складе | КОЛ\_ВО\_СКЛ |
| 23 | Единица измерения товара | ЕД\_ИЗМ |
| 24 | ТМБ | ТМБ |
| 25 | Марка товара | МАРКА |
| 26 | ГОСТ | ГОСТ |
| 27 | Идентификационный номер поставщика | ИДП |
| 28 | ФИО представителя поставщика | ФИО\_ПОС |
| 29 | Наименование фирмы-поставщика | НАИМ\_ФИРМ |
| 30 | Телефон поставщика | ТЕЛ\_ПОС |
| 31 | Адрес поставщика | АДР\_ПОС |
| 32 | Счет поставщика | СЧЕТ |
| 33 | РНН | РС |
| 34 | МФО | МФО |

На основании необходимых запросов выделим следующие сущности с атрибутами:

ЗАКАЗ (Номер заказа, ФИО клиента, номер банковского счета, название фирмы, адрес, телефон, номенклатурный номер товара, цена, количество, дата поставки);

ТОВАР (номенклатурный номер товара, наименование товара, стоимость, стоимость без НДС, количество, единица измерения, ТМБ, марка товара, гост);

ПОСТАВЩИК (Идентификационный код поставщика, ФИО, наименование фирмы, адрес поставщика, телефон, счет, рнн, мфо );

КЛАДОВЩИК (табельный номер, ФИО, адрес, телефон, дата рождения, рнн, стаж, оклад, должность);

Проведем анализ связей между сущностями:

КЛАДОВЩИК, ЗАКАЗ - оформляет; связь типа М:М;

КЛАДОВЩИК, ТОВАР – получает, связь типа М:М;

ПОСТАЩИК, ТОВАР - поставляет, связь типа М:М;

После выбора сущностей, задания атрибутов и анализа связей между сущностями проектируем концептуальную схему БД. (см. рис. 7)

Заказ

Оформляет

Кладовщик

Получает

Товар

Поставщик

выбирает

Рис. 7 Концептуальная схема «Склад».

**2.2 Этап логического проектирования**

Этап логического проектирования позволяет осуществить переход от концептуальной информационной схемы ПО к логической модели БД, ориентированной на выбранную СУБД и конфигурацию ЭВМ. Этап логического проектирования можно представить как совокупность процессов выбора СУБД и отображения концептуальной модели БД на логическую.

Для отображения концептуальной модели на логическую приведем отношения, сформированные на предыдущем этапе проектирования к 3НФ. При этом необходимо произвести декомпозицию предметной области до получения множества отношений, каждое из которых является составной неделимой единицей информации. Проведем анализ функциональных зависимостей между атрибутами в пределах каждого отношения.

1. ЗАКАЗ (НЗ, Клиент, банк, наз\_фирм, адр\_кл, тел\_кл, ннтов, цена, кол\_во, срок);

Заказ определяется по номеру заказа, поэтому в качестве ключевого выберем атрибут «Номер заказа».

НЗ а [все атрибуты].

Кроме того, атрибуты клиент и банк определяют информацию о клиенте, поэтому, чтобы избежать транзитивной зависимости выделим зависимые атрибуты в новое отношение. В результате декомпозиции получим следующие отношения:

ЗАКАЗ (НЗ, Клиент, ннтов, цена, кол\_во, срок);

КЛИЕНТ(Клиент, банк, наз\_фирм, адр\_кл, тел\_кл);

В новом отношении Клиент ключевыми атрибутами являются Клиент и банк, то есть

КЛИЕНТ, БАНКа [все атрибуты].

Других функциональных зависимостей в отношениях нет, оба отношения находятся в 3НФ

2. ТОВАР (ннтов, наз\_тов, стоим, без\_НДС, кол\_во\_скл, ед\_изм, ТМБ, марка, гост);

Товары учитываются по номенклатурным номерам, поэтому в качестве ключевого атрибута выберем «ННТОВ» - номенклатурный номер товара.

ННТОВ а [все атрибуты].

Других ФЗ отсутствуют, отношение находится в 3НФ.

3. ПОСТАВЩИК (ИДП, ФИО\_пос, наим\_фир, адр\_пос,тел\_пос, счет, рс, мфо );

.

Поставщики определяется по идентификационному коду, поэтому в качестве ключевого выберем атрибут «ИДП».

ИДП а[все атрибуты].

Другие ФЗ отсутствуют, отношение находится в 3НФ.

4. КЛАДОВЩИК (тн, ФИО, адрес, тел, дата\_рож, рнн, стаж, оклад, долж);

Кладовщик определяется по табельному номеру, поэтому в качестве ключевого выберем атрибут «ТН».

ТН а[все атрибуты].

Другие ФЗ отсутствуют, отношение находится в 3НФ.

Для отображения информационной модели, полученной на этапе концептуального проектирования, на логическую модель БД необходимо каждое отношение, представленное аналитически, перевести в таблицу, которая и будет представлять одно отношение логической модели БД. В таблице столбец соответствует атрибуту отношения.

Отношение КЛИЕНТ:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Клиент | банк | наз\_фирм | адр\_кл | тел\_кл |

Отношение ЗАКАЗ:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| НЗ | Клиент | ннтов | цена | кол\_во | срок |

Отношение ТОВАР:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ннтов | наз\_тов | стоим | без\_НДС | кол\_во\_скл | ед\_изм | ТМБ | марка | гост |

Отношение ПОСТАВЩИК:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ИДП | ФИО\_пос | наим\_фир | адр\_пос | тел\_пос | счет | рс | мфо |

Отношение КЛАДОВЩИК:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| тн | ФИО | адрес | тел | дата\_рож | рнн | стаж | оклад | долж |

Ключи отношений выделены подчеркиванием.

**2.3 Этап машинного проектирования**

Этап машинного проектирования включает в себя разработку пользовательского интерфейса, при помощи которого пользователь взаимодействует с программой: вводит запрашиваемые данные, загружает и сохраняет рабочие файлы, выполняет интересующие запросы и т.д.

На данном этапе выполняется описание структуры таблиц с указанием имен, типов, размерностей полей, входящих в состав БД.

Для выполнения работы выбираем реляционную модель данных и СУБД My SQL, т.к. она наиболее близко отражает внутреннюю модель данных, удовлетворяет пользователей базы данных с точки зрения технических характеристик, а также обладает широкими возможностями при проектировании удаленных клиентских приложений.

ЗАКЛЮЧЕНИЕ

В данной курсовой работе были рассмотрены аспекты создания распределенных баз данных на примере Systrm R\*. Рассмотрена структура распределенных БД, требования к распределенным БД.

Направление интегрированных или федеративных систем неоднородных БД и мульти-БД появилось в связи с необходимостью комплексирования систем БД, основанных на разных моделях данных и управляемых разными СУБД.

Основной задачей интеграции неоднородных БД является предоставление пользователям интегрированной системы глобальной схемы БД, представленной в некоторой модели данных, и автоматическое преобразование операторов манипулирования БД глобального уровня в операторы, понятные соответствующим локальным СУБД. В теоретическом плане проблемы преобразования решены, имеются реализации.

При строгой интеграции неоднородных БД локальные системы БД утрачивают свою автономность. После включения локальной БД в федеративную систему все дальнейшие действия с ней, включая администрирование, должны вестись на глобальном уровне. Поскольку пользователи часто не соглашаются утрачивать локальную автономность, желая тем не менее иметь возможность работать со всеми локальными СУБД на одном языке и формулировать запросы с одновременным указанием разных локальных БД, развивается направление мульти-БД. В системах мульти-БД не поддерживается глобальная схема интегрированной БД и применяются специальные способы именования для доступа к объектам локальных БД. Как правило, в таких системах на глобальном уровне допускается только выборка данных. Это позволяет сохранить автономность локальных БД.

Как правило, интегрировать приходится неоднородные БД, распределенные в вычислительной сети. Это в значительной степени усложняет реализацию. Дополнительно к собственным проблемам интеграции приходится решать все проблемы, присущие распределенным СУБД: управление глобальными транзакциями, сетевую оптимизацию запросов и т.д. Очень трудно добиться эффективности.

Как правило, для внешнего представления интегрированных и мульти-БД используется (иногда расширенная) реляционная модель данных. В последнее время все чаще предлагается использовать объектно-ориентированные модели, но на практике пока основой является реляционная модель. Поэтому, в частности, включение в интегрированную систему локальной реляционной СУБД существенно проще и эффективнее, чем включение СУБД, основанной на другой модели данных.

При курсовом проектировании достигнуты поставленные в начале работы цели, а именно исследование распределенных баз данных. Решена задача создания проектирования БД.

ГЛОССАРИЙ

|  |  |  |
| --- | --- | --- |
| №п/п | Новое понятие | Содержание |
| 1 | **Распределенная база данных (Distributed DataBase - DDB)** | база данных, включающая фрагменты из нескольких баз данных, которые располагаются на различных узлах сети компьютеров, и, возможно управляются различными СУБД |
| 2 | **Однородная распределенная база данных** | каждая локальная база данных управляется одной и той же СУБД |
| 3 | **Неоднородная распределенная база данных** | локальные базы данных могут относиться даже к разным моделям данных |
| 4 | **Локальная автономия** | управление данными на каждом из узлов распределенной системы выполняется локально |
| 5 | **Независимость от центрального узла** | все узлы равноправны и независимы, а расположенные на них базы являются равноправными поставщиками данных в общее пространство данных |
| 6 | **Непрерывные операции** | возможность непрерывного доступа к данным в рамках DDB вне зависимости от их расположения и вне зависимости от операций, выполняемых на локальных узлах |
| 7 | **Прозрачность расположения** | Пользователь, обращающийся к DDB, ничего не должен знать о реальном, физическом размещении данных в узлах информационной системы. |
| 8 | **Прозрачная фрагментация** | возможность распределенного (то есть на различных узлах) размещения данных, логически представляющих собой единое целое |
| 9 | **Тиражирование данных** | асинхронный процесс переноса изменений объектов исходной базы данных в базы, расположенные на других узлах распределенной системы |
| №п/п | Новое понятие | Содержание |
| 10 | **Прозрачность тиражирования** | возможность переноса изменений между базами данных средствами, невидимыми пользователю распределенной системы. |
| 11 | **Обработка распределенных запросов** | возможность выполнения операций выборки над распределенной базой данных, сформулированных в рамках обычного запроса на языке SQL. |
| 12 | **Обработка распределенных транзакций** | возможность выполнения операций обновления распределенной базы данных (INSERT, UPDATE, DELETE), не разрушающее целостность и согласованность данных. |
| 13 | **Независимость от оборудования** | качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей - от мэйнфреймов до "персоналок". |
| 14 | **Независимость от операционных систем** | многообразие операционных систем, управляющих узлами распределенной системы. |
| 15 | **Прозрачность сети** | в распределенной системе возможны любые сетевые протоколы. |
| 16 | **Независимость от баз данных** | распределенной системе могут мирно сосуществовать СУБД различных производителей, и возможны операции поиска и обновления в базах данных различных моделей и форматов. |

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бройдо В. Л., Крылова В.С. «Научные основы организации управления и построения АСУ», Высшая школа, Москва, 1990 г.
2. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем. М., Финансы и статистика, 1998.
3. Вендров А.М. Проектирование программного обеспечения экономических информационных систем. М., Финансы и статистика, 2000.
4. Дейт К. «Введение в системы управления базами данных», БИНОМ, Москва, 1999 г.
5. Иоффе А. Ф. «Персональные ЭВМ в организационном управлении», Наука, Москва, 1988 г.
6. Информатика и вычислительная техника: пособие для студ. Вузов инж. - педагогич. спец. В. В. Вьюгин, С. В. Кудимов, В. Г. Накрохин и др.; под ред. В. Н. Ларионова. – М.: Высш. Шк. 1992. – 287 с.: ил.
7. Интернет-сайты: http://ibphoenix.com/main.nfs?a=ibphoenix&s=1132891400: 150223&page=ibp\_config, www.ais.khstu.ru, www.libertarium.ru.
8. Ковязин А.Н., С.М. Востриков "Мир InterBase. Архитектура, администрирование и разработка приложений баз данных в InterBase/Firebird/Yaffil (3-е издание)" - М.: Кудиц-образ, 2005
9. Марин Дж. «Организация баз данных в вычислительных системах», Мир, Москва, 1990г.
10. Немнюгин С.А.– «Delphi»- СПб, Питер – 2000 г.
11. Оптимизация приложений С++Builder в архитектуре клиент/сервер, Наталия Елманова, Центр информационных технологий  
    (http://www.citforum.ru/programming/cpp/cb\_498.shtml
12. Фаронов В.В.– «Delphi 7.0» – изд. «Нолидж» - 2001 г.
13. Фаронов В. В. – «Delphi 7.0» – изд. «Нолидж» - 1999г;
14. Фигурнов В.Э.– «IBM PC для пользователя. Краткий курс» – изд. «ИНФРА-М» - 2000г
15. Чен П.П. Модель “сущность-связь” – шаг к единому представлению данных. СУБД, N3, 1995 г
16. Чертовской В.Д. Базы и банки данных. СПб, БХВ-Петербург – 2005 г.

ПРИЛОЖЕНИЕ 1

Листинг основной программы

unit Umain;

interface

uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, Menus, ImgList;

type

TFmain = class(TForm)

MainMenu1: TMainMenu;

ImageList1: TImageList;

N1: TMenuItem; N2: TMenuItem; N3: TMenuItem; N4: TMenuItem;

N5: TMenuItem; N6: TMenuItem; N7: TMenuItem; N8: TMenuItem;

N9: TMenuItem; N11: TMenuItem; N12: TMenuItem;

N14: TMenuItem; N15: TMenuItem; N13: TMenuItem;

N16: TMenuItem;

procedure N5Click(Sender: TObject); procedure N12Click(Sender: TObject);

procedure N8Click(Sender: TObject); procedure N7Click(Sender: TObject);

procedure N15Click(Sender: TObject); procedure N6Click(Sender: TObject);

procedure N14Click(Sender: TObject); procedure N13Click(Sender: TObject);

procedure N9Click(Sender: TObject); procedure N16Click(Sender: TObject);

procedure N3Click(Sender: TObject);

private { Private declarations }

public { Public declarations }

end;

var

Fmain: TFmain;

implementation

uses UDM, UDobTovar, UTovar, UDobFirm, UFirm, UZakaz, UDobZakaz, UInvVed1, URepIVed, UKlad, Upass, UDogf, UTPoisk, UAbout, USPoisk, UZapr;

Продолжение приложения 1

{$R \*.dfm}

procedure TFmain.N5Click(Sender: TObject);

begin FTovar.Show; end;

procedure TFmain.N12Click(Sender: TObject);

begin FPass.Close; close; end;

procedure TFmain.N8Click(Sender: TObject);

begin FFirm.Show; end;

procedure TFmain.N7Click(Sender: TObject);

begin FZakaz.Show; end;

procedure TFmain.N15Click(Sender: TObject);

begin FInvVed.Show; end;

procedure TFmain.N6Click(Sender: TObject);

begin FKlad.Show; end;

procedure TFmain.N14Click(Sender: TObject);

begin fdog.Show; end;

procedure TFmain.N13Click(Sender: TObject);

begin FTPoisk.Show; end;

procedure TFmain.N9Click(Sender: TObject);

begin AboutBox.Show; end;

procedure TFmain.N16Click(Sender: TObject);

begin FSPoisk.Show; end;

procedure TFmain.N3Click(Sender: TObject);

begin Form1.Show; end;

end.

unit UNewPass;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Продолжение приложения 1

Dialogs, StdCtrls, Buttons, Registry;

type

TFNewPass = class(TForm)

Edit1: TEdit; Edit2: TEdit; Edit3: TEdit; SpeedButton1: TSpeedButton; Label1: TLabel; Label2: TLabel; Label3: TLabel;

procedure SpeedButton1Click(Sender: TObject);

procedure FormActivate(Sender: TObject);

procedure Edit1KeyPress(Sender: TObject; var Key: Char);

procedure Edit2KeyPress(Sender: TObject; var Key: Char);

procedure Edit3KeyPress(Sender: TObject; var Key: Char);

private { Private declarations }

public { Public declarations }

end;

var FNewPass: TFNewPass;

implementation

uses Upass;

{$R \*.dfm}

procedure TFNewPass.SpeedButton1Click(Sender: TObject);

begin close; end;

procedure TFNewPass.FormActivate(Sender: TObject);

begin Edit1.Text := ''; Edit2.Text := ''; Edit3.Text := ''; Edit1.Enabled := true;

Edit1.SetFocus; Edit2.Enabled := false; Edit3.Enabled := false; end;

procedure TFNewPass.Edit1KeyPress(Sender: TObject; var Key: Char);

begin if (key = #13) and (Edit1.Text = Fpass.LabeledEdit1.Text) then

begin Edit2.Enabled := true; Edit1.Enabled := false; Edit2.SetFocus;

end; end;

procedure TFNewPass.Edit2KeyPress(Sender: TObject; var Key: Char);

begin if key = #13 then begin Edit3.Enabled := true; Edit3.SetFocus;

Продолжение приложения 1

Edit2.Enabled := false; end; end;

procedure TFNewPass.Edit3KeyPress(Sender: TObject; var Key: Char);

var Reg:TRegistry;

begin if (key = #13) and (Edit2.Text = Edit3.Text) then begin

Reg := TRegistry.Create; with Reg do begin

RootKey := HKEY\_LOCAL\_MACHINE;

OpenKey('Software', True); if not KeyExists('Cardi') then CreateKey('Cardi');

OpenKey('Cardi', True); WriteString('Log', Edit3.Text); end; close; end;

end; end.

unit Upass;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Buttons, StdCtrls, ExtCtrls, Registry;

type

TFPass = class(TForm)

LabeledEdit1: TLabeledEdit; SpeedButton1: TSpeedButton;

SpeedButton2: TSpeedButton; SpeedButton3: TSpeedButton;

procedure SpeedButton1Click(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure SpeedButton2Click(Sender: TObject);

procedure SpeedButton3Click(Sender: TObject);

private { Private declarations }

public { Public declarations }

end;

var FPass: TFPass; pass, newpass:string; x:byte;

implementation

uses UZast, UNewPass;

{$R \*.dfm}

Продолжение приложения 1

procedure TFPass.SpeedButton1Click(Sender: TObject);

begin x:= x+1; if x<3 then Begin if LabeledEdit1.Text = pass then

begin FZast.Show; Visible:=false; end else begin

LabeledEdit1.SetFocus; MessageDlg('Пароль не верен!!! Повторите ввод', mtInformation,[mbOk],0); end end else

begin MessageDlg('Пароль не верен!!! Доступ запрещен!!!', mtInformation,

[mbOk], 0); Close; end; end;

procedure TFPass.FormCreate(Sender: TObject);

var reg:Tregistry;

begin x:=0; LabeledEdit1.Text := ''; Reg := TRegistry.Create; with Reg do

begin RootKey := HKEY\_LOCAL\_MACHINE; OpenKey('Software', True);

if not KeyExists('Cardi') then CreateKey('Cardi');

OpenKey('Cardi', True); if not ValueExists('Log') then begin

newpass := ''; WriteString('Log', newpass); end else pass := ReadString('Log'); end; end;

procedure TFPass.SpeedButton2Click(Sender: TObject);

begin close; end;

procedure TFPass.SpeedButton3Click(Sender: TObject);

begin FNewPass.Show; end;

end.

unit URepIVed;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, QRCtrls, QuickRpt, ExtCtrls;

Type TFRepIVed = class(TForm)

QuickRep1: TQuickRep; DetailBand1: TQRBand;

ColumnHeaderBand1: TQRBand; TitleBand1: TQRBand;

QRLabel1: TQRLabel; QRLabel2: TQRLabel;

Продолжение приложения 1

QRLabel3: TQRLabel; QRLabel4: TQRLabel;

QRLabel5: TQRLabel; QRLabel6: TQRLabel;

QRLabel7: TQRLabel; QRLabel8: TQRLabel;

QRLabel9: TQRLabel; QRLabel10: TQRLabel;

QRDBText1: TQRDBText; QRDBText2: TQRDBText;

QRDBText3: TQRDBText; QRDBText4: TQRDBText;

QRDBText5: TQRDBText; QRDBText6: TQRDBText;

QRDBText7: TQRDBText; QRDBText8: TQRDBText;

QRDBText9: TQRDBText; SummaryBand1: TQRBand;

QRLabel11: TQRLabel; QRLabel12: TQRLabel;

QRLabel13: TQRLabel;

Private { Private declarations }

Public { Public declarations }

end;

var FRepIVed: TFRepIVed;

implementation

uses UDM;

{$R \*.dfm}

end.

unit USPoisk;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Mask,db, DBCtrls, StdCtrls, ExtCtrls, Buttons, ComCtrls;

type

TFSPoisk = class(TForm)

PageControl1: TPageControl; TabSheet1: TTabSheet;

Label1: TLabel; Label2: TLabel; Label3: TLabel;

Label4: TLabel; Label5: TLabel; Label6: TLabel;

Продолжение приложения 1

Label7: TLabel; Label8: TLabel; SpeedButton2: TSpeedButton;

SpeedButton4: TSpeedButton; LabeledEdit1: TLabeledEdit;

DBEdit1: TDBEdit; DBEdit2: TDBEdit;

DBEdit3: TDBEdit; DBEdit4: TDBEdit;

DBEdit5: TDBEdit; DBEdit6: TDBEdit;

DBEdit7: TDBEdit; DBEdit8: TDBEdit;

TabSheet2: TTabSheet; SpeedButton1: TSpeedButton;

SpeedButton3: TSpeedButton; Label12: TLabel;

Label13: TLabel; Label14: TLabel;

Label15: TLabel; Label16: TLabel;

DBEdit12: TDBEdit; DBEdit13: TDBEdit;

DBEdit14: TDBEdit; DBEdit15: TDBEdit;

DBEdit16: TDBEdit; LabeledEdit2: TLabeledEdit;

TabSheet3: TTabSheet; SpeedButton5: TSpeedButton;

SpeedButton6: TSpeedButton; Label9: TLabel;

DBEdit9: TDBEdit; Label10: TLabel;

DBEdit10: TDBEdit; Label11: TLabel;

DBEdit11: TDBEdit; LabeledEdit3: TLabeledEdit;

Label17: TLabel; DBEdit17: TDBEdit;

Label18: TLabel; DBEdit18: TDBEdit;

Label19: TLabel; DBEdit19: TDBEdit;

Label20: TLabel; DBEdit20: TDBEdit;

Label21: TLabel; DBEdit21: TDBEdit;

Label22: TLabel; DBEdit22: TDBEdit;

Label23: TLabel; DBEdit23: TDBEdit;

Label24: TLabel; DBEdit24: TDBEdit;

procedure SpeedButton2Click(Sender: TObject);

procedure SpeedButton6Click(Sender: TObject);

procedure SpeedButton3Click(Sender: TObject);

Продолжение приложения 1

procedure SpeedButton5Click(Sender: TObject);

procedure SpeedButton1Click(Sender: TObject);

procedure SpeedButton4Click(Sender: TObject);

private { Private declarations }

public { Public declarations }

end;

var FSPoisk: TFSPoisk;

implementation

uses UDM;

{$R \*.dfm}

procedure TFSPoisk.SpeedButton2Click(Sender: TObject);

begin DM.Tklad.Locate('fio',LabeledEdit1.Text,[lopartialkey]); end;

procedure TFSPoisk.SpeedButton6Click(Sender: TObject);

begin DM.Tklad.Locate('stag',strtoint(LabeledEdit3.Text),[lopartialkey]); end;

procedure TFSPoisk.SpeedButton3Click(Sender: TObject);

begin DM.Tklad.Locate('dolj',LabeledEdit2.Text,[lopartialkey]); end;

procedure TFSPoisk.SpeedButton5Click(Sender: TObject);

begin close; end;

procedure TFSPoisk.SpeedButton1Click(Sender: TObject);

begin close; end;

procedure TFSPoisk.SpeedButton4Click(Sender: TObject);

begin close; end;

end.

unit UTovar;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Grids, DBGrids, ImgList, ComCtrls, ToolWin;

Продолжение приложения 1

Type TFTovar = class(TForm)

ToolBar1: TToolBar; TBFirst: TToolButton;

ToolButton2: TToolButton; ToolButton3: TToolButton;

ToolButton4: TToolButton; ToolButton5: TToolButton;

ToolButton6: TToolButton; ToolButton7: TToolButton;

ImageList1: TImageList; DBGrid1: TDBGrid;

ToolButton1: TToolButton;

procedure TBFirstClick(Sender: TObject);

procedure ToolButton2Click(Sender: TObject);

procedure ToolButton3Click(Sender: TObject);

procedure ToolButton4Click(Sender: TObject);

procedure ToolButton5Click(Sender: TObject);

procedure ToolButton6Click(Sender: TObject);

procedure ToolButton7Click(Sender: TObject);

procedure ToolButton1Click(Sender: TObject);

private { Private declarations }

public { Public declarations }

end;

var

FTovar: TFTovar;

implementation

uses UDM, UDobTovar, Umain;

{$R \*.dfm}

procedure TFTovar.TBFirstClick(Sender: TObject);

begin DM.Ttov.First; end;

procedure TFTovar.ToolButton2Click(Sender: TObject);

begin DM.Ttov.Prior; end;

procedure TFTovar.ToolButton3Click(Sender: TObject);

begin DM.Ttov.Next; end;

Продолжение приложения 1

procedure TFTovar.ToolButton4Click(Sender: TObject);

begin DM.Ttov.Last; end;

procedure TFTovar.ToolButton5Click(Sender: TObject);

begin DM.Ttov.Post; end;

procedure TFTovar.ToolButton6Click(Sender: TObject);

begin DM.Ttov.Delete; end;

procedure TFTovar.ToolButton7Click(Sender: TObject);

var n:integer; begin DM.Ttov.Last; n:=DM.Ttov.FieldByName('idn').Value;

DM.Ttov.Append; DM.Ttov.FieldByName('idn').Value:=n+1;

FDobTov.Show; close; end;

procedure TFTovar.ToolButton1Click(Sender: TObject);

begin Fmain.Show; close; end;

end.

unit UTPoisk;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Buttons, StdCtrls, Mask, db,DBCtrls, ExtCtrls, ComCtrls;

Type TFTPoisk = class(TForm)

PageControl1: TPageControl; TabSheet1: TTabSheet;

TabSheet2: TTabSheet; TabSheet3: TTabSheet;

TabSheet4: TTabSheet; TabSheet5: TTabSheet;

LabeledEdit1: TLabeledEdit; DBEdit1: TDBEdit;

DBEdit2: TDBEdit; DBEdit3: TDBEdit;

DBEdit4: TDBEdit; DBEdit5: TDBEdit;

DBEdit6: TDBEdit; DBEdit7: TDBEdit;

DBEdit8: TDBEdit; Label1: TLabel; Label2: TLabel; Label3: TLabel;

Label4: TLabel; Label5: TLabel; Label6: TLabel; Label7: TLabel;

Label8: TLabel; SpeedButton2: TSpeedButton;

Продолжение приложения 1

SpeedButton4: TSpeedButton; SpeedButton1: TSpeedButton;

SpeedButton3: TSpeedButton; DBEdit9: TDBEdit;

Label9: TLabel; DBEdit10: TDBEdit; Label10: TLabel;

DBEdit11: TDBEdit; Label11: TLabel; DBEdit12: TDBEdit;

Label12: TLabel; Label13: TLabel; DBEdit13: TDBEdit;

Label14: TLabel; DBEdit14: TDBEdit; Label15: TLabel;

DBEdit15: TDBEdit; DBEdit16: TDBEdit; Label16: TLabel;

LabeledEdit2: TLabeledEdit; SpeedButton5: TSpeedButton;

SpeedButton6: TSpeedButton; DBEdit17: TDBEdit;

Label17: TLabel; DBEdit18: TDBEdit; Label18: TLabel;

DBEdit19: TDBEdit; Label19: TLabel; DBEdit20: TDBEdit;

Label20: TLabel; Label21: TLabel; DBEdit21: TDBEdit;

Label22: TLabel; DBEdit22: TDBEdit; Label23: TLabel;

DBEdit23: TDBEdit; DBEdit24: TDBEdit; Label24: TLabel;

LabeledEdit3: TLabeledEdit; SpeedButton7: TSpeedButton;

SpeedButton8: TSpeedButton; DBEdit25: TDBEdit;

Label25: TLabel; DBEdit26: TDBEdit; Label26: TLabel;

DBEdit27: TDBEdit; Label27: TLabel; DBEdit28: TDBEdit;

Label28: TLabel; Label29: TLabel; DBEdit29: TDBEdit;

Label30: TLabel; DBEdit30: TDBEdit; Label31: TLabel;

DBEdit31: TDBEdit; DBEdit32: TDBEdit; Label32: TLabel;

LabeledEdit4: TLabeledEdit; SpeedButton9: TSpeedButton;

SpeedButton10: TSpeedButton; DBEdit33: TDBEdit;

Label33: TLabel; DBEdit34: TDBEdit; Label34: TLabel;

DBEdit35: TDBEdit; Label35: TLabel; DBEdit36: TDBEdit;

Label36: TLabel; Label37: TLabel; DBEdit37: TDBEdit;

Label38: TLabel; DBEdit38: TDBEdit; Label39: TLabel;

DBEdit39: TDBEdit; DBEdit40: TDBEdit; Label40: TLabel;

LabeledEdit5: TLabeledEdit;

Продолжение приложения 1

procedure SpeedButton2Click(Sender: TObject);

procedure SpeedButton3Click(Sender: TObject);

procedure SpeedButton6Click(Sender: TObject);

procedure SpeedButton8Click(Sender: TObject);

procedure SpeedButton10Click(Sender: TObject);

procedure SpeedButton4Click(Sender: TObject);

procedure SpeedButton1Click(Sender: TObject);

procedure SpeedButton5Click(Sender: TObject);

procedure SpeedButton7Click(Sender: TObject);

procedure SpeedButton9Click(Sender: TObject);

private { Private declarations }

public { Public declarations }

end;

var FTPoisk: TFTPoisk;

implementation

uses UDM;

{$R \*.dfm}

procedure TFTPoisk.SpeedButton2Click(Sender: TObject);

begin DM.Ttov.Locate('naz\_tov',LabeledEdit1.Text,[lopartialkey]); end;

procedure TFTPoisk.SpeedButton3Click(Sender: TObject);

begin DM.Ttov.Locate('bez\_nds',strtoint(LabeledEdit2.Text),[]); end;

procedure TFTPoisk.SpeedButton6Click(Sender: TObject);

begin DM.Ttov.Locate('tmb',LabeledEdit3.Text,[lopartialkey]); end;

procedure TFTPoisk.SpeedButton8Click(Sender: TObject);

begin DM.Ttov.Locate('gost',LabeledEdit4.Text,[lopartialkey]); end;

procedure TFTPoisk.SpeedButton10Click(Sender: TObject);

begin DM.Ttov.Locate('marka',LabeledEdit5.Text,[lopartialkey]); end;

procedure TFTPoisk.SpeedButton4Click(Sender: TObject);

begin close; end;

Продолжение приложения 1

procedure TFTPoisk.SpeedButton1Click(Sender: TObject);

begin close; end;

procedure TFTPoisk.SpeedButton5Click(Sender: TObject);

begin close; end;

procedure TFTPoisk.SpeedButton7Click(Sender: TObject);

begin close; end;

procedure TFTPoisk.SpeedButton9Click(Sender: TObject);

begin close; end;

end.

unit UZakaz; interface uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ImgList, ComCtrls, ToolWin, Grids, DBGrids, ExtCtrls, StdCtrls,

DBCtrls;

Type TFZakaz = class(TForm)

DBGrid1: TDBGrid; ToolBar1: TToolBar; TBFirst: TToolButton;

ToolButton2: TToolButton; ToolButton3: TToolButton;

ToolButton4: TToolButton; ToolButton5: TToolButton;

ToolButton6: TToolButton; ToolButton7: TToolButton;

ToolButton1: TToolButton; ImageList1: TImageList;

Panel1: TPanel; ToolButton8: TToolButton;

DBText1: TDBText; Label1: TLabel;

DBText2: TDBText; DBText3: TDBText;

DBText4: TDBText; DBText5: TDBText;

DBText6: TDBText; DBText7: TDBText;

DBText8: TDBText; DBText9: TDBText;

DBText10: TDBText; Label2: TLabel; Label3: TLabel;

Label4: TLabel; Label5: TLabel; Label6: TLabel;

Label7: TLabel; Label8: TLabel; Label9: TLabel;

Shape1: TShape; Label10: TLabel; Shape2: TShape;

Продолжение приложения 1

Label11: TLabel; Shape3: TShape; Label12: TLabel;

Edit1: TEdit; procedure TBFirstClick(Sender: TObject);

procedure ToolButton2Click(Sender: TObject);

procedure ToolButton3Click(Sender: TObject);

procedure ToolButton4Click(Sender: TObject);

procedure ToolButton5Click(Sender: TObject);

procedure ToolButton6Click(Sender: TObject);

procedure ToolButton7Click(Sender: TObject);

procedure ToolButton1Click(Sender: TObject);

procedure ToolButton8Click(Sender: TObject);

procedure FormActivate(Sender: TObject);

procedure Edit1Change(Sender: TObject);

private { Private declarations }

public { Public declarations }

end;

var

FZakaz: TFZakaz; a:byte;

implementation

uses UDM, Umain, UDobZakaz;

{$R \*.dfm}

procedure TFZakaz.TBFirstClick(Sender: TObject);

begin DM.Tzakaz.First; end;

procedure TFZakaz.ToolButton2Click(Sender: TObject);

begin DM.Tzakaz.Prior; end;

procedure TFZakaz.ToolButton3Click(Sender: TObject);

begin DM.Tzakaz.Next; end;

procedure TFZakaz.ToolButton4Click(Sender: TObject);

begin DM.Tzakaz.Last; end;

procedure TFZakaz.ToolButton5Click(Sender: TObject);

Продолжение приложения 1

begin DM.Tzakaz.post; end;

procedure TFZakaz.ToolButton6Click(Sender: TObject);

begin DM.Tzakaz.Delete; end;

procedure TFZakaz.ToolButton7Click(Sender: TObject);

var n:integer; begin DM.Tzakaz.Last; n:=DM.Tzakaz.FieldByName('nz').Value;

DM.Tzakaz.Append; DM.Tzakaz.FieldByName('nz').Value:=n+1;

FDobZakaz.show; end;

procedure TFZakaz.ToolButton1Click(Sender: TObject);

begin case a of 0:begin Height:=697; Panel1.Height:=201; Position:=poScreenCenter; a:=1; ToolButton1.ImageIndex:=9;

ToolButton1.hint:='Закрыть панель поиска' end;

1: begin DBGrid1.DataSource:= dm.DSZakaz; Height:=486; Panel1.Height:=0;

Position:=poScreenCenter; a:=0; ToolButton1.ImageIndex:=8;

ToolButton1.hint:='Открыть панель поиска'; Edit1.SetFocus; end; end;

end;

procedure TFZakaz.ToolButton8Click(Sender: TObject);

begin Fmain.Show; close; end;

procedure TFZakaz.FormActivate(Sender: TObject);

begin a:=0; end;

procedure TFZakaz.Edit1Change(Sender: TObject);

begin dm.QZakaz1.Close; dm.QZakaz1.Parameters[0].Value:=edit1.Text;

dm.QZakaz1.Open; DBGrid1.DataSource:= dm.DSQZakaz1; end; end.

unit UZapr;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Buttons, ExtCtrls, DBCtrls, Grids, DBGrids;

Type TForm1 = class(TForm)

DBGrid1: TDBGrid; DBNavigator1: TDBNavigator;

Продолжение приложения 1

SpeedButton2: TSpeedButton; SpeedButton4: TSpeedButton;

procedure SpeedButton4Click(Sender: TObject);

procedure SpeedButton2Click(Sender: TObject);

private { Private declarations }

public { Public declarations }

end;

var Form1: TForm1;

implementation

uses UDM;

{$R \*.dfm}

procedure TForm1.SpeedButton4Click(Sender: TObject);

begin close; end;

procedure TForm1.SpeedButton2Click(Sender: TObject);

begin DM.QZapros.Close; DM.QZapros.Open; end;

end.

unit UDobTovar;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Buttons, ExtCtrls, StdCtrls, DBCtrls, Mask, BDE;

type

TFDobTov = class(TForm)

GroupBox1: TGroupBox; DBEdit3: TDBEdit; DBEdit5: TDBEdit;

DBEdit7: TDBEdit; DBEdit8: TDBEdit; DBEdit9: TDBEdit;

DBEdit1: TDBEdit; DBEdit2: TDBEdit; Label1: TLabel;

Label2: TLabel; Label3: TLabel; Label4: TLabel;

Edit1: TEdit; Label5: TLabel; Label6: TLabel;

Label7: TLabel; Label8: TLabel; Label9: TLabel;

DBComboBox1: TDBComboBox; Panel1: TPanel;

Продолжение приложения 1

Panel2: TPanel; SpeedButton1: TSpeedButton;

SpeedButton2: TSpeedButton; SpeedButton3: TSpeedButton;

SpeedButton4: TSpeedButton; DBEdit4: TDBEdit;

procedure SpeedButton1Click(Sender: TObject);

procedure SpeedButton2Click(Sender: TObject);

procedure SpeedButton3Click(Sender: TObject);

procedure SpeedButton4Click(Sender: TObject);

procedure FormActivate(Sender: TObject);

private { Private declarations }

public { Public declarations }

end;

var FDobTov: TFDobTov;

implementation

uses UDM, UTovar;

{$R \*.dfm}

procedure TFDobTov.SpeedButton1Click(Sender: TObject);

begin

DBEdit4.Text:=floattostr(strtofloat(DBEdit3.Text)+(strtofloat(Edit1.text)/100)\*strtofloat(DBEdit3.Text)); DM.Ttov.post; end;

procedure TFDobTov.SpeedButton2Click(Sender: TObject);

begin DM.Ttov.Append; end;

procedure TFDobTov.SpeedButton3Click(Sender: TObject);

begin DM.Ttov.Cancel; FTovar.Show; Close; end;

procedure TFDobTov.SpeedButton4Click(Sender: TObject);

begin FTovar.Show; Close; end;

procedure TFDobTov.FormActivate(Sender: TObject);

var n:integer; begin DM.Ttov.Last; n:=DM.Ttov.FieldByName('idn').Value;

DM.Ttov.Append; DM.Ttov.FieldByName('idn').Value:=n+1; end;

end.

Продолжение приложения 1

unit UInvVed1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Mask, DBCtrls, Buttons;

type

TFInvVed = class(TForm)

ComboBox1: TComboBox; Label1: TLabel; DBEdit1: TDBEdit;

SpeedButton1: TSpeedButton; SpeedButton2: TSpeedButton;

procedure FormActivate(Sender: TObject);

procedure ComboBox1Change(Sender: TObject);

procedure SpeedButton2Click(Sender: TObject);

procedure SpeedButton1Click(Sender: TObject);

private { Private declarations }

public { Public declarations }

end;

var FInvVed: TFInvVed;

implementation

uses UDM, URepIVed;

{$R \*.dfm}

procedure TFInvVed.FormActivate(Sender: TObject);

begin DM.Tklad.First; while not DM.Tklad.Eof do begin

ComboBox1.Items.Add(DM.Tklad.FieldValues['fio']); DM.Tklad.next; end; end;

procedure TFInvVed.ComboBox1Change(Sender: TObject);

begin DM.Tklad.Locate('fio',ComboBox1.Text,[]); end;

procedure TFInvVed.SpeedButton2Click(Sender: TObject);

begin close; end;

procedure TFInvVed.SpeedButton1Click(Sender: TObject);

begin FRepIVed.QRLabel12.Caption:=DBEdit1.Text;

Продолжение приложения 1

FRepIVed.QRLabel13.Caption:=ComboBox1.Text;

FRepIVed.QuickRep1.Preview; end;

end.

ПРИЛОЖЕНИЕ 2

Образы экранов

