Министерство образования и науки РФ

ФЕДЕРАЛЬНОЕ АГЕНСТВО ПО ОБРАЗОВАНИЮ

ГОУ ВПО «АДЫГЕЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

**КУРСОВАЯ РАБОТА**

**По дисциплине:** «Сетевые технологии»

**«Объектно-ориентированные базы данных, работающие в распределенных сетях»**

**Выполнил:** студент группы 4А2

специальности 230102 (220200) АСОИУ

**Хмиляр М.А.**

**Научный руководитель:**

**Алиев М.В.**

**Рецензент:**

**Кизянов А.Ф.**

**Майкоп 2006 год**

СОДЕРЖАНИЕ

**ВВЕДЕНИЕ** 5

1. ОБЩИЕ СВЕДЕНЬЯ ОБ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ

БАЗАХ ДАННЫХ 6

2. ОСОБЕННОСТИ БАЗ ДАННЫХ ПОСТРОЕННЫХ НА СУБД

РАЗНЫХ ФИРМ 9

*2.1 Cache'* 9

*2.2 GemStone* 13

*2.3 ITASCA* 17

*2.4 Objectivity /DB* 20

*2.5 ObjectStore* 22

*2.6 Versant* 24

**ЗАКЛЮЧЕНИЕ** 28

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ** 29

ВВЕДЕНИЕ

Возникновение направления объектно-ориентированных баз данных (ООБД) определялось, прежде всего, потребностями практики: необходимостью разработки сложных информационных прикладных систем, для которых технология предшествующих систем баз данных не была вполне удовлетворительной.

Конечно, ООБД возникли не на пустом месте. Соответствующий базис обеспечивался как предыдущими работами в области баз данных, так и давно развивающимися направлениями языков программирования с абстрактными типами данных и объектно-ориентированных языков программирования.

Что касается связи с предыдущими работами в области баз данных, то наиболее сильное влияние на работы в области ООБД оказали проработки реляционных СУБД и следующего хронологически за ними семейства БД, в которых поддерживалось управление сложными объектами. Эти работы обеспечили структурную основу организации OOБД.

Актуальность.

Несмотря на то, что направление объектно-ориентированных баз данных возникло сравнительно давно и публикации появлялись уже в середине 1980-х. Однако наиболее активно это направление стало развиваться в последние годы. Это подтверждается тем, что с каждым годом увеличивается число публикаций и реализованных коммерческих и экспериментальных систем.

Цель работы.

1. Описать объектно-ориентированные базы данных.
2. Описать особенности построения баз данных приложениями разных фирм.
3. Выявить преимущества.
4. Выявить недостатки.

1. ОБЩИЕ СВЕДЕНЬЯ ОБ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ БАЗАХ ДАННЫХ

Объектно-ориентированные базы данных применяются с конца 1980-х годов для обеспечения управления базами данных, приложениями, построенными в соответствии с концепцией объектно-ориентированного программирования. Объектная технология расширяет традиционную методику разработки приложений новым моделированием данных и методами программирования. Для повторного использования кода и улучшения сохранности целостности данных в объектном программировании данные и код для их обработки организованы в объекты. Таким образом, практически полностью снимаются ограничения на типы данных.

Если данные состоят из коротких, простых полей фиксированной длины (имя, адрес, баланс банковского счета), то лучшим решением будет применение реляционной базы данных[4]. Если, однако, данные содержат вложенную структуру, динамически изменяемый размер, определяемые пользователем произвольные структуры (мультимедиа, например), представление их в табличной форме будет, как минимум, непростым. В то же время в ООСУБД каждая определенная пользователем структура – это объект, непосредственно управляемый базой данных.

В реляционных СУБД связи управляются пользователем, создающим внешние ключи. Затем для обнаружения связей динамически во время выполнения система просматривает две (или больше) таблицы, сравнивая внешние ключи до достижения соответствия. Этот процесс, называемый объединением (join), является слабой стороной реляционной технологии. Если более двух или трех уровней объединений – сигнал, чтобы искать лучшее решение. В ООСУБД пользователь просто объявляет связь, и СУБД автоматически генерирует методы управления, динамически создавая, удаляя и пересекая связи. Ссылки при этом прямые, нет необходимости в просмотре и сравнении или даже поиске индекса, который может сильно сказаться на производительности. Таким образом, применение объектной модели предпочтительнее для баз данных с большим количеством сложных связей: перекрестных ссылок, ссылок, связывающих несколько объектов с несколькими (many-to-many relationships) двунаправленными ссылками[1].

В отличие от реляционных, ООСУБД полностью поддерживают объектно-ориентированные языки программирования. Разработчики, применяющие С++ или Smalltalk, имеют дело с одним набором правил (позволяющих использовать такие преимущества объектной технологии, как наследование, инкапсуляция и полиморфизм)[7]. Разработчик не должен прибегать к трансляции объектной модели в реляционную и обратно. Прикладные программы обращаются и функционируют с объектами, сохраненными в базе данных, которая использует стандартную объектно-ориентированную семантику языка и операции. Напротив, реляционная база данных требует, чтобы разработчик транслировал объектную модель к поддерживаемой модели данных и включил подпрограммы, чтобы обеспечить это отображение во время выполнения. Следствием являются дополнительные усилия при разработке и уменьшение эффективности[2].

И, наконец, ООСУБД подходят (опять же без трансляций между объектной и реляционной моделями) для организации распределенных вычислений. Традиционные базы данных (в том числе и реляционные и некоторые объектные) построены вокруг центрального сервера, выполняющего все операции над базой. По существу, эта модель мало отличается от мэйнфреймовой организации 60‑х годов с центральной ЭВМ – мэйнфреймом (mainframe), выполняющей все вычисления, и пассивных терминалов. Такая архитектура имеет ряд недостатков, главным из которых является вопрос масштабируемости. В настоящее время рабочие станции (клиенты) имеют вычислительную мощность порядка 30 ‑ 50 % мощности сервера базы данных, то есть большая часть вычислительных ресурсов распределена среди клиентов. Поэтому все больше приложений, и в первую очередь базы данных и средства принятия решений, работают в распределенных средах, в которых объекты (объектные программные компоненты) распределены по многим рабочим станциям и серверам и где любой пользователь может получить доступ к любому объекту. Благодаря стандартам межкомпонентного взаимодействия все эти фрагменты кода комбинируются друг с другом независимо от аппаратного, программного обеспечения, операционных систем, сетей, компиляторов, языков программирования, различных средств организации запросов и формирования отчетов и динамически изменяются при манипулировании объектами без потери работоспособности[3].

2. ОСОБЕННОСТИ БАЗ ДАННЫХ ПОСТРОЕННЫХ НА СУБД РАЗНЫХ ФИРМ

2.1 Cache'

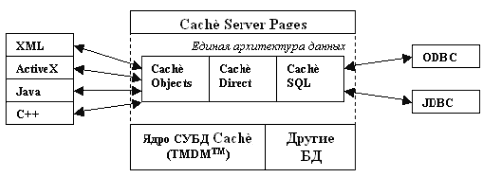
При всех достоинствах современной объектной технологии разработки баз данных имеется несколько препятствий, которые удерживают разработчиков от принятия решения о переходе с реляционной технологии на объектную. Основным препятствием является значительный объем разработок, опирающихся на реляционные СУБД. Ведь при переходе на объектную технологию необходимо многое начинать «с нуля», и поэтому возникает вопрос целесообразности такого перехода. Кроме того, объектная технология, поддерживаемая в ряде постреляционных СУБД, не имеет развитого и стандартизированного языка генерации отчетов и анализа данных, каким является структурированный язык запросов SQL. Данные проблемы были решены при создании постреляционной СУБД Cache' от InterSystems. СУБД Cache' обеспечивает не только реализацию основных возможностей объектно-ориентированной технологии, но и позволяет во многом облегчить переход с реляционной технологии на объектную, а также может выступать в роле шлюза к реляционным базам данных[5].

Отличительной особенностью СУБД Cache' является независимость хранения данных от способа их представления, что реализуется с помощью, так называемой, единой архитектуры данных Cache'. В рамках данной архитектуры существует единое описание объектов и таблиц, отображаемых непосредственно в многомерные структуры ядра базы данных, ориентированных на обработку транзакций. Как только определяется класс объектов, Cache' автоматически генерирует реляционное описание данных этого класса в формате SQL. Подобным же образом, как только в Словарь данных поступает DDL-описание в формате реляционной базы данных, Cache' автоматически генерирует реляционное и объектное описание данных, устанавливая тем самым доступ в формате объектов. При этом все описания ведутся согласованно, все операции по редактированию проводятся только с одним описанием данных. Это позволяет сократить время разработки, сэкономить вычислительные ресурсы и приложения будут работать значительно быстрее.

БД Cache' была первой базой данной, предназначенной для работы в сети Internet/Intranet. В версии Cache' 4.0 реализована технология создания динамических web-приложений Cache' Server Pages (CSP), которая пришла на смену технологии Weblink предыдущих версий Cache'. Кроме этого, системная библиотека «%Net» предоставляет классы, реализующие протоколы SMTP, POP3, HTTP, FTP и др.

Основные компоненты СУБД Cache'. На рис. 1. представлена архитектура Cache'.

Рисунок 1. Архитектура системы Cache'.



Основными компонентами СУБД Cache' являются следующие:

TMDM. Многомерное ядро системы, ориентирование на работу с транзакциями.

Сервер Cache' Objects. Представление многомерных структур данных ядра системы в виде объектов, инкапсулирующих как данные, так и методы их обработки.

Сервер Cache' SQL. Представление многомерных структур данных в виде реляционных таблиц.

Сервер прямого доступа. Предоставление прямого доступа к многомерным структурам данных ядра системы.

Рассмотрим подробнее назначение и функциональные возможности основных компонентов системы.

TMDM - многомерное ядро Cache', ориентированное на работу с транзакциями.

Данные в Cache' хранятся в виде разреженных массивов, носящих название глобалей. Количество индексов массива может быть произвольным, что позволяет описывать и хранить структуры данных произвольного уровня сложности. Индексы глобалей не типизированы, т.е. они могут быть любого литерального типа данных.

Применение разреженных массивов позволяет оптимизировать использование объема жесткого диска и сократить время, требуемое на выполнение операций ввода/вывода и извлечение данных.

В СУБД Cache' реализована развитая технология обработки транзакций и разрешения конфликтов. Блокировка данных производится на логическом уровне. Это позволяет учитывать особенность многих транзакций, производящих изменения небольшого объема информации. Кроме этого, в Cache' реализованы атомарные операции добавления и удаления без проведения блокировки, в частности, это применяется для счетчика ID объектов в базе данных[6].

Сервер Cache' Objects. В соответствии с ODMG каждый объект Cache' имеет определенный, единственный тип. Поведение объекта определяется операциями (методами), а состояние объекта - значениями его свойств. Свойства и операции составляют характеристики типа. Тип определяется одним интерфейсом, которому может соответствовать одна или большее число реализаций. Объектная модель Cache' представлена на рис. 2.

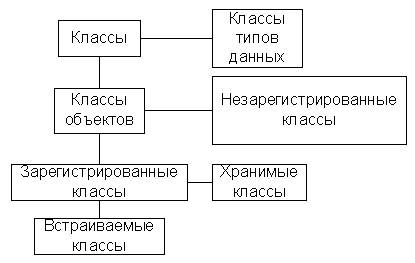


Рисунок 2. Объектная модель Cache'.

В соответствии со стандартом в Cache' реализовано два типа классов:

* классы типов данных (литералы);
* классы объектов (объекты).

Классы типов данных определяют допустимые значения констант (литералов) и позволяют их контролировать. Литерал не может существовать независимо от своего значения, в то время как объекты имеют уникальную идентификацию.

Классы типов данных подразделяется на два подкласса типов:

* атомарные;
* структурированные.

Атомарными литеральными типами в Cache' являются традиционные скалярные типы данных (%String, %Integer, %Float, %Date и др.). В Cache' реализованы две структуры классов типов данных - список и массив. Каждый литерал уникально идентифицируется индексом в массиве и порядковым номером в списке.

Различают два подтипа классов объектов - зарегистрированные и незарегистрированные. Зарегистрированные классы обладают предопределенным поведением, т.е. набором методов, наследуемых из системного класса «%RegisteredObject» и отвечающих за создание новых объектов и за управление размещением объектов в памяти. Незарегистрированные классы не обладают предопределенным поведением, разработка функций (методов) класса целиком и полностью возлагается на разработчика.

Зарегистрированные классы могут быть двух типов - встраиваемые и хранимые. Встраиваемые классы наследуют свое поведение от системного класса «%SerialObject». Основной особенностью хранения встраиваемого класса является то, что объекты встраиваемых классов существуют в памяти как независимые экземпляры, однако могут быть сохранены в базе данных, только будучи встроены в другой класс.

Основным преимуществом использования встроенных классов является минимум издержек, связанных с возможным в будущем изменением набора одинаковых свойств классов, представленных в виде встраиваемого объекта.

Хранимые классы наследуют свое поведение от системного класса «%Persistent». «%Persistent» предоставляет обширный набор функций своим наследникам, включающий: создание объекта, подкачку объекта из БД в память, удаление объекта и т.п. Каждый экземпляр хранимого класса имеет 2 уникальных идентификатора - OID и OREF. OID (object ID) характеризует объект, записанный в БД, т.е. на физическом носителе, а OREF (object reference) характеризует объект, который был подкачен из БД и находится в памяти.

2.2 GemStone

Данная система была разработана компанией Servio-Logic совместно с OGI. В исходном варианте системы разработчики GemStone опирались на язык Smalltalk. Хотя в первых выпусках системы ее основной язык назывался Opal, сразу было видно, что в действительности этого всего лишь Smalltalk с поддержкой стабильного хранения объектов, и вскоре название языка было заменено на GemStone Smalltalk. Впоследствии в GemStone была обеспечена поддержка языков C и C ++, но во все времена базовым языком оставался Smalltalk, а все остальные интерфейсы строились поверх базового. И серверная, и клиентская части системы могут работать под управлением всех основных ветвей ОС UNIX и всех развитых вариантов Windows. В настоящее время продукт поддерживается, развивается и распространяется компанией GemStone Systems Inc[8].

Система основана на трехзвенной архитектуре клиент-сервер, в которой прикладная обработка данных производится на среднем уровне между процессом взаимодействия с пользователем и процессом, поддерживающим хранилище данных. Важность этого подхода состоит в том, что, если в приложении используется много данных, то код приложения целесообразно расположить на стороне хранилища данных, а если в приложении производится много изменений над небольшим объемом данных, то имеет смысл разместить код приложения на стороне пользователя. Тем самым, архитектура позволяет уменьшить объем сетевого трафика без перегрузки сервера, что повышает скорость обработки данных.

Для управления мультидоступом используется механизм транзакций. Механизм основан на так называемом оптимистическом подходе, при котором каждая сессия работает с собственной локальной копией хранилища объектов, и слияние произведенных в сессиях изменений хранилища происходит при завершении транзакции. Если при завершении транзакции обнаруживается, что произведенные в ней изменения конфликтуют с изменениями других ранее зафиксированных транзакций, то фиксация транзакции не производится, транзакция не завершается, и решение проблемы возлагается на пользователя. Автоматическая блокировка объектов не производится, но пользователь может явно запросить блокировку, что повышает шансы на успешную фиксацию транзакции.

Для обеспечения безопасности данных поддерживается механизм авторизации доступа на уровне владельца объекта и его группы пользователей, так что может быть ограничен доступ к некоторым объектам или некоторым методам объектов. К каждому объекту приписывается авторизационный объект, содержащий данные о том, какие пользователи и в каком режиме (чтения или изменения) имеют доступ к объекту.

Для восстановления базы данных после сбоев аппаратуры используются механизмы репликации, резервного копирования и журнализации. Любой авторизованный пользователь может запросить выполнения полного или частичного копирования. Восстановление базы данных после сбоя системы или дисковых устройств начинается с использования последней по времени резервной копии. После этого при помощи данных, сохраненных в журнальных файлах, хранилище объектов приводится к состоянию, соответствующему последней до момента сбоя зафиксированной транзакции. Авторизованные пользователи могут также запросить поддержку реплицирования областей хранилища данных.

В системе поддерживается целостность по ссылкам между всеми объектами, поскольку все ссылки основываются на использовании объектных идентификаторов, и объекты, для которых существуют ссылки от других объектов, не могут быть удалены. Для повышения скорости доступа к часто используемым коллекциям обеспечивается средство построения индексов.

Объекты делаются стабильными (т.е. сохраняются в базе данных) путем использования своего рода стабильного корня, называемого коннектором. Все объекты, прямо или косвенно достижимые по объектным ссылкам от коннектора, являются стабильными. В GemStone у каждого класса, в котором существует хотя бы один стабильный объект, поддерживается эквивалентная серверная версия класса. Другими словами, один вариант класса служит классом в контексте программирования, а другой – в контексте базы данных. Такие пары поддерживаются автоматически: если создается класс в смысле Smalltalk, и некоторый объект этого класса становится стабильным, то автоматически создается серверный класс этого объекта (класс в смысле GemStone). Создание коннектора приводит к появлению экземпляра класса GemStone, эквивалентного классу объекта, который должен быть сделан стабильным. Аналогично, любой объект, достижимый от коннектора, автоматически становится стабильным.

В GemStone поддерживается динамическая сборка мусора (garbage collection). Процесс-«мусорщик» автоматически освобождает память, занимаемую объектами, на которые отсутствуют ссылки.

В среде GemStone можно использовать различные реализации Smaltalk, а также языки C и C++. Классы и объекты можно создавать с использованием любого из этих языков, и объекты, созданные на одном языке можно использовать в приложениях, написанных на любом другом языке. Реализация языка C представляет собой набор функций и набор компонентов, преобразующих объекты GemStone в указатели и литералы C и наоборот. Реализация C++ включает препроцессор в чистый С и библиотеку классов[7].

Подключения к реляционным системам (например, Oracle или IBM DB2) производятся через шлюзы. Для синхронизации состояния локальной (управляемой GemStone) и внешних копий данных обеспечивается автоматическая модификация данных. В зависимости от среды и требований к уровню синхронизованности эти обновления выполняются немедленно или же в пакетном режиме.

GemStone можно также использовать для управления данными, соответствующими стандартам OLE и CORBA. Для работы с данными в реляционном стиле поддерживаются стандарты SQL и ODBC.

2.3 ITASCA

Распределенная ООСУБД ITASCA основана на результатах проекта Orion, выполнявшегося в MCC. Разработка серии из трех прототипов завершилась выпуском системы, основанной на архитектуре «много клиентов - много серверов». Система была доведена до уровня коммерческого продукта начинающей техасской компанией, которая в 1995г. была приобретена компанией IBEX Corp[9].

В распределенной архитектуре ITASCA частные и совместно используемые базы данных разнесены по узлам локальной UNIX -ориентированной сети. Каждой значение данных хранится в одном узле, но централизованное управление отсутствует; все серверы автономны. На каждом сервере поддерживаются кэш страниц и кэш объектов, и каждый сервер множество клиентов с обеспечением мультидоступа на основе блокировок. На клиентах поддерживается только кэш объектов.

Для управления мультидоступом в ITASCA используется двухфазный протокол синхронизационных блокировок с сериализацией транзакций и обнаружением тупиков. Также поддерживаются долгие транзакции на основе перемещения объекта из совместно используемой базы данных в частную базу данных (check -out). Для обеспечения совместной работы допускается участие нескольких пользователей в одной долгой транзакции.

Для всей распределенной базы данных поддерживается единая схема с использование подсхем для частных фрагментов базы данных. Модель данных включает следующие аспекты:

* множественное наследование;
* представление классов в виде объектов;
* наличие свойств и операций классов;
* поддержка ограничений целостности;
* возможность перегрузки операций.

В любое время могут добавляться новые данные, классы, свойства и операции. Для обеспечения контроля над распространением таких операций как удаление объекта имеется возможность определения составных объектов. Для поддержки мультимедийных приложений имеется возможность использования линейных массивных объектов, которые предназначены прежде всего для хранения последовательных данных, таких как текст или аудиоданные. В пространственных массивных объектах имеются два измерения, и они подходят, например, для хранения изображений.

Восстановление базы данных после сбоев производится на основе журнала, предназначенного для аннулирования результата выполненных операций (undo log). Это позволяет в процессе восстановления устранить эффект всех транзакций, не завершившихся к моменту сбоя. Фиксация транзакции заключается в том, что на сервере все объекты, измененные данной транзакцией, перемещаются из буфера объектов в буфер страниц.

Поддерживается механизм индексирования, основанный на использовании техники B +-деревьев. Можно создавать индексы для одного класса и одного свойства или для нескольких свойств нескольких классов.

Имеется возможность создания классов, поддерживающих оповещение. Имеются две формы оповещения – пассивная и активная. Пассивное оповещение состоит в том, что сохраняется информация о модификации или удалении экземпляров класса. Приложение может обратиться классу с запросом данных о таких событиях. Активное оповещение приводит к вызову некоторой операции при выполнении операций модификации, удаления, создания версии, перемещения объекта из общей базы данных в частную базу данных (check -out) или наоборот (check -in). По умолчанию выполнение операции оповещения приводит к отправке электронной почты привилегированному пользователю (администратору системы), но допускается замена поведения этой операции.

Допускается создание временной, рабочей или «выпускной» версии объекта. Для динамического или статического связывания разных версий поддерживается иерархия происхождения версий. При использовании динамического связывания версий иерархия автоматически модифицируется при создании новых версий.

Безопасность данных обеспечивается на основе механизма авторизации доступа, в котором конкретная привилегия (доступ по чтению, доступ по записи или создание) предоставляется роли, за которой может стоять один пользователь или группа пользователей. Привилегии могут быть подсоединены к базам данных, классам, экстентам, объектам, операциям и свойствам. Имеется авторизация по умолчанию, которая подразумевается для любой роли и может быть дополнена явной авторизацией, положительной (с добавлением привилегий) или отрицательной (с изъятием привилегии).

При использовании C++ стабильность достигается путем доступа к библиотеке классов, поддерживающих стабильность. В CLOS (Common Lisp Object System) обеспечивается метакласс стабильности. Стабильные объекты должны быть экземплярами классов, являющихся экземплярами этого метакласса. Кроме того, можно указать, что некоторые свойства стабильного класса являются недолговечными.

В ITASCA поддерживаются C, C++, Smalltalk, CLOS. Акцент делается на возможности динамического изменения схемы без остановки действия системы и без потребности в массовой повторной компиляции и редактирования связей. Доступ к программам на каждом из языков производится через функциональный API. В случае использования C++ автоматически создается файл заголовков, который сливается с исходными файлами программного кода при генерации приложения.

Собственный механизм запросов ITASCA позволяет запрашивать данные в частной базе данных, общей базе данных или сразу в обеих базах данных. Для повышения производительности применяются оптимизация запросов и методы распараллеливания.

2.4 Objectivity /DB

Компания Objectivity была образована в 1988г. В 1990 г. была выпущена первая версия системы, которая представляла собой распределенную СУБД, основанную на использовании объектной технологии, высокопропускных локальных сетей и симметричных мультипроцессоров. Система работает на всех основных UNIX -платформах и в среде Windows[10].

Система основана на клиент-серверной архитектуре, в которой единицей обмена между сервером и клиентом является страница базы данных. Тем самым многие системные функции, включая кэширование, поиск объектов и преобразование их форматов, выполняются на стороне клиента. Объектные идентификаторы представляются в 64-разрядном формате, что обеспечивает потенциальную возможность работы со сверхбольшими базами данных.

Поддерживается четырехуровневая структура хранения данных. Объекты содержатся в контейнерах, каждый из которых представляет собой часть локальной базы данных. Локальные же базы данных могут комбинироваться в федеративную (распределенную) базу данных. Надежность хранения данных поддерживается механизмом репликации.

Обеспечивается возможность изменения классов и автоматического образования новых версий существующих объектов. Поддерживается механизм управления иерархиями версий объектов.

Допускаются как короткие, так и долгие транзакции. Управление короткими транзакциями основано на синхронизационных блокировках и распознавании тупиков. Долгие транзакции и коллективная работа с базами данных основаны на многоверсионности объектов и механизме check -in / check -out.

В системе поддерживаются языки C++ и Smalltalk, но способы использования языков сильно различаются. Это относится и к механизмам стабильности, и к средствам определения данных. В среде C++ стабильными являются объекты всех классов, являющихся наследниками специального системного класса. В среде Smalltalk стабильными являются все объекты, достижимые от именованных корневых объектов-словарей. Соответственно, в Smalltalk для удаления объектов используется механизм сборки мусора, а в C++ – явные операции.

Естественно, базы данных, управляемые Objectivity /DB, основаны на одной объектной модели. Эта модель достаточно близка к модели ODMG и, в частности, включает возможность определения двунаправленных связей. Поддерживается возможность управления составными объектами с распространением на подобъекты операции удаления. Однако способы определения данных в средах C++ и Smalltalk различаются.

В C++ включено специальное расширение языка, предназначенное для определения данных. Соответствующие конструкции обрабатываются специальным препроцессором, который генерирует код C++, содержащий соответствующие обращения к СУБД. В среде Smalltalk схема базы данных определяется как набор классов Smalltalk. Другими словами, при использовании Smalltalk приложения, работающие с базами данных Objectivity /DB, организуются более прозрачным образом, чем в случае C++.

Имеется поддержка языка SQL /89 и, частично, SQL /92. Реляционный доступ к базам данных, управляемых Objectivity /DB, возможен через интерактивный SQL-ориентированный интерфейс, через имеющийся драйвер ODBC и через API.

Последняя версия Objectivity /DB идеально, по заявлениям фирмы, подходит для приложений, которые работают в распределенных средах, требуют гибкой модификации данных, организации сложных связей, а также нуждаются в высокой производительности и работы с большими объемами данных. Более содержательно, Objectivity обеспечила интеграцию инструментария СУБД и разработки приложений с такими средствами программирования, как SoftBench и C++. Благодаря интегрированному графическому интерфейсу разработки схемы БД и инструментам отладки и анализа упрощается задание модели базы данных и, соответственно, разработки приложений для Objectivity /DB.

2.5 ObjectStore

Компания Object Design была основана в 1988 г. с экстренной целью разработать и вывести на рынок ООСУБД, которую стали называть ObjectStore. В конце 90-х у Object Design установились тесные партнерские отношения с IBM, что позволило привлечь к ObjectStore тысячи разработчиков приложений. На основе технологии ObjectStore компанией был разработана одна из первых коммерческих СУБД – Excelon, ориентированная на управление XML -данными. С начала 2003г. компания является подразделением компании Progress Software[11].

ООСУБД ObjectStore основана на архитектуре клиент-сервер, в которой каждый сервер отвечает за регулирование доступа к хранилищу объектов и управляет журнализацией обновлений, блокировками, установкой контрольных точек, разрешением конфликтов по данным, резервированием данных и восстановлением базы данных после сбоев. Каждый сервер поддерживает множество клиентов. В клиентском процессе используется представление данных более высокого уровня, и клиентская часть ObjectStore отвечает за управление коллекциями, выполнение запросов и управление транзакциями.

Серверная часть спроектирована в расчете на использование механизма отображения виртуальной памяти, которая распространяется на всю сеть и может охватывать несколько серверов. Извлекаемые из базы данных объекты могут объединяться в пакеты для передачи по сети, что позволяет снизить объем сетевого трафика. Для сокращения времени доступа в серверах используется техника кластеризации. На каждой клиентской части имеется локальный кэш, в котором сохраняются используемые объекты. Когда объект перемещается в адресное пространство клиента, ссылки на него перерабатываются таким образом, что для каждого к объекту доступа достаточно одной команды компьютера.

Управление мультидоступом основано на использовании прозрачного для пользователя механизма блокировок, включающего возможность блокирования по чтению и по записи. Поддерживаются разные уровни детализации (гранулированности) блокировок от уровня страниц внешней памяти базы данных до конфигураций (указываемых программистом групп объектов).

Надежность хранения данных обеспечивается за счет поддержания журнала произведенных изменений. Подсистема управления транзакциями отвечает за журнализацию всех произведенных изменений на основе протокола WAL (Write Agead Log – упреждающей записи в журнал). Дополнительно поддерживается архивный журнал, в котором авторизованный пользователь может произвести архивное копирование базы данных.

Имеется средство поддержки версий, которое обеспечивает возможность коллективной работы с базами данных на основе механизмов check -in / check -out. На этом подходе основывается поддержка долгих транзакций. Для каждой конфигурации объектов можно создать историю версий, независимую от типов объектов.

В ObjectStore стабильность хранения объектов поддерживается за счет наличия именованных корневых стабильных объектов класса база данных. База данных создается с помощью вызова метода new этого класса. Имеются методы для открытия и закрытия базы данных. Кроме того, в классе содержатся методы для создания стабильных корневых объектов, обычно являющихся коллекциями, в которых размещаются стабильные объекты.

Поддерживаются языки C, C++ и Smaltalk. Свойство стабильности обеспечивается за счет включения в библиотеку классов специальных системных классов. Имеются классы, поддерживающие коллекции – списки, множества, мультимножества и массивы. Методы этих классов поддерживают выборку объектов из коллекций, вставку и удаление объектов.

Поддерживаются шлюзовые объекты, поддерживающие доступ к реляционным данным, а также инструментальные средства для отображения реляционной схемы в эквивалентное объектно-ориентированное представление. Таким образом, с реляционными базами данных можно работать в интерфейсе ODBC на основе SQL или в собственном интерфейсе ObjectStore.

2.6 Versant

С 1988 г. компания Versant предлагает решения, основанные на хорошо масштабируемой объектно-ориентированной архитектуре и принадлежащем компании алгоритме кэширования. ООСУБД Versant является одной из немногих объектно-ориентированных систем, допускающих масштабирование уровня любого предприятия. Решения на базе Versant применяются в телекоммуникациях, обороне, на транспорте и т.д. Система работает как на основных UNIX-платформах, так и в среде Windows[12].

Архитектура Versant в большей степени ориентирована на логическое управления данными, т.е. объектами, а не на физическое представление данных в виде, например, страниц. Управление размещением объекта осуществляется системой способом, полностью прозрачным для пользователей. Для поддержки локальных хранилищ объектов используется кэширование.

Система обладает свойством отказоустойчивости. Для этого допускается синхронная репликация базы данных на двух серверах, которые могут находиться в одной локальной сети или разнесены в разные точки глобальной сети. В одной базе данных Versant может храниться около трехсот триллионов объектов, размер каждого из которых неограничен. Для архивации данных может использоваться третичная внешняя память с автоматическим оповещением оператора в случае потребности извлечения объектов из архива.

Поддерживаются кластеры совместно используемых объектов, причем встроенные объекты хранятся внутри своих объектов-предков, что способствует уменьшению уровня фрагментации памяти. Кластеризация применяется и при внешнем кэшировании. Кроме того, в системе Versant поддерживается возможность использования персональных баз данных, установленных на мобильных компьютерах. Они могут быть отсоединены от сервера центральной базы данных, использоваться автономно и зафиксировать свои изменения в центральной базе данных после восстановления соединения.

Управление транзакциями основывается, главным образом, на синхронизационных блокировках на уровне объектов, хотя возможны блокировки классов и версий объектов. Имеется целый ряд разновидностей блокировок: короткие блокировки для коротких транзакций, стабильные блокировки для долгих транзакций и т.д. Допускается даже возможность расширения модели блокировки правилами, желательными для пользователей. Система избегает тупиковых синхронизационных ситуаций, не удовлетворяя запросы на блокировку, которые могут привести к тупику.

Фиксация распределенных транзакций основывается на двухфазном протоколе фиксации. Поддерживаются частичная фиксация кэшей, механизмы контрольных точек и точек сохранения. Обеспечивается и возможность образования вложенных транзакций. При реализации долгих транзакций используется механизм check -in / check -out с установкой стабильной блокировки на требуемые объекты, предотвращающей доступ к этим объектам со стороны других транзакций до завершения данной долгой транзакции.

Имеется возможность регистрации на сервере событий, которые интересуют приложения. При регистрации серверу сообщается вид события и операция, которую следует выполнять при возникновении события. К событиям, которые разрешается регистрировать, относятся обновление и удаление классов, создание, обновление и удаление объектов.

Для повышения надежности хранения баз данных поддерживаются два вида журналов – логический и физический. При необходимости восстановления базы данных по архивной копии все зафиксированные к моменту сбоя транзакции повторно воспроизводятся по логическому журналу.

Обеспечивается ссылочная целостность базы данных и прозрачность месторасположения объектов в распределенной среде. Объекты могут мигрировать по узлам сети, что способствует балансировке нагрузки, и оставаться полностью доступными для приложений. Допускается динамическая модификация классов, приводящая к автоматической модификации всех существующих в базе данных объектов этих классов. При этом система все время остается в рабочем состоянии, и приложения продолжают выполняться. Поддерживается развитый механизм версий. По известной версии объекта можно получить доступ к его предкам, потомкам и братьям.

Для представления связей между объектами базы данных используется единый стабильный указательный тип. В системе поддерживаются скрытые от пользователей преобразования указателей базы данных в обычные указатели C ++ и наоборот. Поэтому объекты создаются и ликвидируются с помощью стандартных конструкторов и деструкторов классов.

Для программирования можно использовать языки C++ и Smalltalk, причем безо всяких расширений. Поддерживаются возможности, специфичные для работы с базами данных. Например, имеется средство автоматической генерации схемы базы данных прямо по файлам заголовков C++. Это позволяет обойтись без использования специализированных препроцессоров или компиляторов. Специальные системные классы позволяют работать со всеми разновидностями типов коллекций, специфицированными в стандарте ODMG. Любой объект, созданный в среде C++, доступен в среде Smalltalk и наоборот.

Запросы к базам данных Versant можно задавать с помощью специального системного класса, позволяющего обходить объекты коллекций. Поддерживается расширенный вариант SQL /89. Имеется драйвер ODBC. Обеспечивается доступ из среды Versant к внешним реляционным базам данных.

ЗАКЛЮЧЕНИЕ

С недавнего времени наметился заметный сдвиг в области освоения объектных СУБД. Уже существуют примеры практического их использования крупными биржами, банками, страховыми компаниями, а также в сфере производства и телекоммуникаций, где базам данных, содержащим гигабайты информации, приходится обслуживать сотни пользователей. Они оказались хорошей альтернативой в тех случаях, когда применение реляционных БД вынуждало строить сложную схему с чрезмерно большим числом межтабличных связей.

Благодаря значительному прогрессу в развитии объектной технологии, за последние пять лет производителям удалось довести свои ООСУБД до такого уровня, что они стали вполне отвечать реальным требованиям рынка.

Несмотря на то, что технология объектных СУБД созрела для крупных проектов, для действительно массового ее распространения необходим специальный инструментарий.

В настоящий момент ощущается настоятельная потребность в интеграции ООСУБД с существующими инструментальными средствами. Разработчики уже сегодня могли бы продуктивно использовать версии Visual Basic, Power Builder, Forte или Delphi, поддерживающие ООСУБД. Большинство продуктов для создания приложений в той или иной мере являются объектно-ориентированными, но работают по-прежнему с реляционными БД. Специалисты считают, что партнерство производителей ООСУБД и средств программирования способно привести к появлению столь необходимого инструментария.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Андреев А.М. Среда и хранилище: ООБД. – М.: «Мир ПК», 1997.

2. Аткинсон М. Манифест систем объектно-ориентированных баз данных, СУБД, N 4, 1995, с.142-155.

3. Буч Г. Объектно-ориентированный анализ и проектирование. 2-ое издание. - М.: «Бином», 1997.

4. Замулин А.В. Системы программирования баз данных и знаний. — Новосибирск: Наука; Сиб. отд-ние, 1993.

5. Кирстен В. СУБД Cache'. Объектно-ориентированная разработка приложений, - СПб.: «Питер», 2001.

6. Кречетов Н. Постреляционная технология Cache' для реализации объектных приложений. – М.: МИФИ, 2001.

7. Страуструп Б. Язык программирования C++. 3-е издание. - М.: «Бином», 1997.

8. Материалы с сайта разработчика. [Электронный ресурс]. – Режим доступа: http://www.gemstone.com/.

9. Материалы с сайта разработчика. [Электронный ресурс]. – Режим доступа: http://www.ibex.ch/.

10. Материалы с сайта разработчика. [Электронный ресурс]. – Режим доступа: http://www.objectivity.com/.

11. Материалы с сайта разработчика. [Электронный ресурс]. – Режим доступа: http://www.objectstore.net/.

12. Материалы с сайта разработчика. [Электронный ресурс]. – Режим доступа: http://www.versant.com/.