Лекция 14

Тема. Технологии уровня данных.

План лекции

Уровень данных. Универсальное хранилище

Число способов хранения информации постоянно и весьма быстро увеличивается. Не так давно данные хранились только на мэйнфреймах и в системах управления базами данных (СУБД). Теперь же информация располагается и в почтовых хранилищах, и в файловых системах, и в Web -документах, и в графических файлах.
При поиске наилучшего метода распространения данных по отделам организации вы, вероятно, захотите поместить всю информацию одном универсальном хранилище, предназначенном для данных всех типов.

Универсальное хранилище

Универсальное хранилище решает проблему выбора методов доступа. Однако при его использовании возникают дополнительные трудности в создании механизма извлечения данных любых типов. Универсальное хранилище также не справляется с обработкой информации, расположенной в других местах, если ее объем измеряется в терабайтах. Кроме того, всегда существует риск, что все хранилище, откажет из-за одного сбоя.
Вообще-то существует общий метод доступа — интерфейс открытого доступа к базам данных (Open Database Connectivity , ODBC), но хотелось бы, чтобы универсальный метод поддерживал все типы данных, а не только реляционные базы данных и запросы SQL , как ODBC .

Интерфейсы прикладного программирования

Интерфейс прикладного программирования (Application Programming Interface , API) - это набор вызовов, используемых приложениями для запроса выполнения низкоуровневых сервисов операционной системы. Каждая компания-разработчик СУБД, чтобы упростить работу с ней, готовит специальный API . Доступ к данным, не поддёрживаемым СУБД, обеспечивают специальные API . Методы доступа, предназначенные для соответствующих хранилищ, позволяют добиться полного контроля над информацией. Однако для этого разработчик должен знать, как применять каждый из этих методов, и понимать все детали работы API , связанного с определенным хранилищем. Поэтому, если предполагается работа с несколькими хранилищами данных придется обучить персонал всем способам работы с ними, что резко увеличивает затраты.
Разработчикам доступен и другой вариант — работать с общими нейтральными API , например ODBC. Преимущество такого метода заключается в том, что для доступа к СУБД требуется изучить только один API, а приложения смогут обращаться к данным нескольких СУБД.

Компоненты доступа на основе UDA.

Разработанная компанией Microsoft универсальная архитектура данных (Universal Data Architecture, UDA) предназначена для организации высокопроизводительного доступа к данным любого типа — структурированным и неструктурированным, связанным и несвязанным — хранящимся на предприятии. UDA — набор COM-интерфейсов, реализующих концепцию доступа к данным. Она основана на интерфейсах OLEDB для создания компонентов, работающих с базами данных. OLEDB позволяет хранилищам информации открывать свои функции без необходимости имитировать реляционный источник данных. Кроме того, в рамках этой технологии универсальным сервисным компонентам (например, специализированным обработчикам запросов) разрешено расширять функции более простых поставщиков данных. Поскольку основная цель OLEDB — эффективный доступ к информации, а не простота использования, в UDA добавлен интерфейс прикладного уровня Microsoft ActiveX Data Object (ADO). ADO поддерживает двойственные интерфейсы, которые можно применять в языках сценариев, в C++ и других средствах разработки.

UDA — платформа и набор средств разработки, определяющий стандарты и технологии, связанные с доступом к хранилищам данных масштаба предприятия. Эта архитектура лежит в основе концепция разработки приложений Microsoft. Кроме того, UDA обеспечивает высокопроизводительный доступ к различным реляционным и нереляционным информационным хранилищам и обладает простым интерфейсом, не зависящим от языка реализации.

При использовании UDA вам не придется перемещать данные в одно хранилище и работать с продуктами только одного производителя, так как UDA поддерживает все основные базы данных.

Компоненты доступа на основе UDA

Microsoft Data Access Component (MDAC) — это реализация UDA , включающая как ADO, так и компонент доступа OLEDB для ODBC. Это позволяет ADO обращаться к любой базе данных, снабженной драйвером ODBC (фактически ко всем основным базам данных). Существуют компоненты доступа OLEDB для других типов хранилищ.. Как показано на рис.14.1, ADO как механизм доступа к информации позволяет написать приложения для уже существующих и новых структурированных неструктурированных данных независимо от их местонахождения.

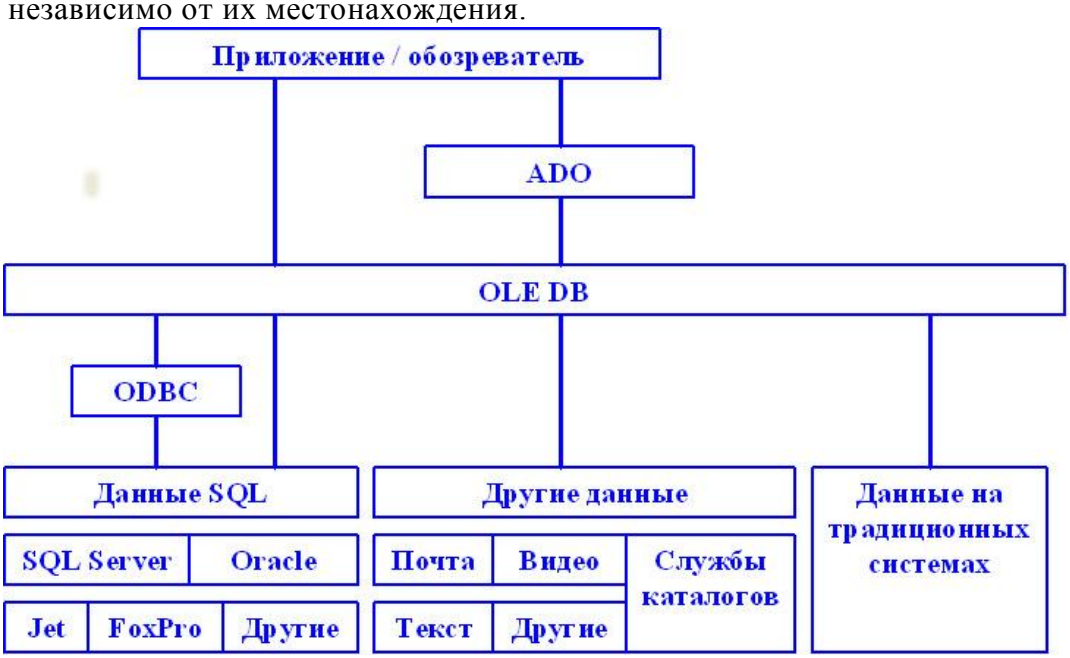

Рис. 14.1. Структура UDA

Моделирование данных

Моделирование данных — процесс идентификации, документирования и реализации требований к данным, при котором существующие модели и процессы пересматриваются с целью определения возможности их повторного использования. Кроме того, создаются новые модели и процессы, реализующие требования к приложению. Основные этапы моделирования данных таковы:

- определение структуры данных и связанных с ними процессов (например, логическая организация данных);
- определение типов данных, их размеров и значений по умолчанию;
- обеспечение целостности данных с помощью бизнес-правил и ограничений;
- подготовка операционных процессов, например проверок защиты и резервного копирования;
- выбор технологии хранения данных (реляционной, иерархической, индексируемой и т. д.).

Важно понимать, что моделирование данных тесно связано со структурой процессов в рамках организации. Так, новое понимание специфических элементов данных, используемых организацией, часто ведет к трудностям, связанным с принадлежностью информации, включая ответственность за ее хранение, точность и своевременность. Иногда при проектировании данных приходится изучать взаимодействие информационных систем предприятия — при скоординированном планировании возрастает эффективность, сокращаются затраты и появляются новые стратегические возможности.

Определение структуры данных

Данные характеризуют реальные информационные ресурсы приложения: людей, продукты, заказчиков, имущество, отчеты и, наконец, структуры данных, которые приложение распределяет по категориям, организует и поддерживает. Процесс определения структуры данных — итерационный. Сначала разработчики в общих чертах представляют механизм обработки информации приложением. По мере расширения знаний о бизнес-процессах он уточняется и становится ясней. Описание каждого элемента данных обычно состоит из:

- названия;
- общего описания;
- владельца (кто за него отвечает);
- характеристик;
- логических событий, процессов и связей (как и. когда данные создаются, изменяются и используются).

Кроме того, следует изучить способы количественного определения элементов данных. Перечислим их основные спецификации или атрибуты:

- местонахождение (адрес, страна, полка на складе);
- физические параметры (вес, размеры, объем, цвет, материал, плотность ткани);
- концептуальные параметры (название, категория, серийный номер);
- связи (деталь, состоящая из нескольких элементов, автор, написавший несколько книг);
- ценность (стоимость).

На данном этапе выполняется анализ существующей структуры данных, документирование и тщательный пересмотр. В конечном результате получается документ, в котором есть ответы на вопросы: «Кто, что, где, когда и как?». Таков, в общих чертах, первый этап исследования данных.

Определение характеристик данных

По мере исследования структур данных можно группировать некоторые их элементы, выявляя характеристики и связи:

- определять таблицы, строки и столбцы;
- выбирать ключевые поля;
- создавать связи между таблицами;
- определять типы данных.

Определение таблиц, строк и столбцов

Независимо от способа физического хранения, данные, как правило, организуются в виде таблиц или файлов, состоящих из нескольких строк (записей) и столбцов (полей). Своим видом они напоминают электронные таблицы.

Выбор ключевых полей

С помощью ключевого поля создается индекс, позволяющий быстро получать информацию. Такое поле бывает уникальным, так и повторяющимся.

Создание связей между таблицами

Базы данных, как правило, состоят из нескольких таблиц, связанных друг с другом. Связи бывают: «один-ко-многим», «один-к-одному», «многие-ко-многим».

Определение типов данных

Тип данных - это именованная категория, характеризующая набор значений, способ обозначения этого набора, а также интерпретирующие и модифицирующие его операции. Типы данных бывают:

- предопределенными — они встроены в СУБД. Например, в SQL Server предопределены такие типы, как integer , datetime , bit , char и varchar ;
- производные — определяются с помощью таких средств СУБД, как язык моделирования данных (Data Modeling Language , DML). Производные типы основываются на предопределенных и уже существующих производных типах данных. Разработчики придумывают их структуру и название. Производные типы позволяют добиться согласованности типов данных, хранящихся в выбранных столбцах, переменных или параметрах.

Типы данных важны еще и потому, что обеспечивают соответствие значения правильному типу и гарантируют, что оно не выходит за рамки определенного диапазона. Различные хранилища информации и языки программирования поддерживают множество типов данных. При определении типов данных разработчики должны убедится, что диапазон значений этих типов соответствует хранимой информации сейчас и будет соответствовать впоследствии.

Выбрав для полей соответствующий тип данных, вы сэкономите место в базе данных и число операций объединения таблиц. Возьмите на вооружение общее правило: выбирайте тип данных наименьшего размера.

При определении типов данных следует учитывать:

- допустимый минимум и максимум значений;
- значения по умолчанию;
- пустые (NULL) значения;
- ожидаемое увеличение значений;
- ожидаемые и, по возможности, неожиданные изменения.

Обеспечение целостности данных

Для обеспечения целостности данных, хранимых и используемых в структурах приложения, следует контролировать все процессы, обращающиеся к информации.

Реализовать это позволяют следующие основные концепции:

- нормализация данных;
- определение бизнес-правил доступа к данным;
- обеспечение целостности ссылок;
- проверка данных.

Нормализация данных

Проектировщик базы данных должен структурировать данные, устранив, таким образом, ненужное дублирование информации и ускорив ее поиск. Такой процесс упорядочения

таблиц, ключевых полей, столбцов и связей, повышающий эффективность базы данных, называется нормализацией. Эта процедура применима как к реляционным, так и к индексируемым файлам.
Нормализация — сложный процесс, подчиняющийся множеству специальных правил и проходящий с разными уровнями интенсивности. Дадим ее полное определение: нормализация - это процесс удаления повторяющихся групп, минимизации избыточности, устранения составных ключевых полей для частичных зависимостей и отделения неключевых атрибутов.

Определение бизнес-правил доступа к данным. Компоненты доступа к данным.

Доступ к данным контролируется бизнес-правилами, причем каждое новое приложение должно применять те же правила, что и первое, то есть уже готовые зависимости и связи. В общем, бизнес-правила, отвечающие за доступ к данным, нужно проектировать с особой тщательностью, создавая независимые, хорошо скоординированные процессы.
Бизнес-правила доступа к данным требуются приложениям в следующих случаях:

- при вставке, обновлении, удалении и просмотре данных;
- при проверке данных;
- при управлении защитой данных;
- при доступе к данным из нескольких источников;
- для обеспечения целостности ссылок.

Бизнес-правила стоит использовать каждый раз, когда приложение вставляет, обновляет, удаляет или просматривает данные. Таким образом обеспечивается полный контроль за обновляемой информацией.
Обеспечение целостности данных — это процесс проверки значений полей и связанных с ними значений в файлах. Благодаря ему можно быть уверенным, что в числовых полях находятся именно числа, причем они не выходят за рамки соответствующего диапазона, а все связи с другими файлами не нарушены. Поместив такие алгоритмы проверки в бизнес-правила, вы создадите приложение, возвращающее корректные данные и легко адаптирующееся к новым требованиям.
Обеспечение ссылочной целостности
Для обеспечения ссылочной целостности необходимо, чтобы все внешние ключи одной таблицы указывали на существующие строки другой. При этом гарантируется синхронизация обеих таблиц при операциях обновления и удаления.
Проверка данных
Проверка данных гарантирует правильность и точность информации. Её можно реализовать по-разному: в коде пользовательского интерфейса, в коде приложения, средствами СУБД и бизнес-правил. Существует несколько типов проверки данных.

- Проверка типов данных: одна из простейших форм проверки данных, позволяет получить ответ на вопросы «Состоит ли строка из символов алфавита?» или «Состоит ли число из цифр?». Обычно такие проверки выполняет пользовательский уровень приложения.
- Проверка диапазона значений: дополняя проверку типа, гарантирует, что значение находится между допустимыми минимумом и максимумом. Например, при работе с символами могут быть разрешены только латинские буквы от A до Z . Все остальные — недопустимы. Как и в случае проверки типа, проверку диапазона обычно выполняет пользовательский уровень, хотя можно создать и соответствующее бизнес-правило.
- Проверка кода: иногда очень сложна. Как правило, для нее нужна специальная справочная таблица. Например, при создании приложения, рассчитывающего

налог с продаж, может потребоваться таблица, содержащая региональные коды налогов. Ее стоит включить в бизнес-правило или реализовать непосредственно в базе данных.

- Комплексная проверка: применяется, когда простой и основанной на справочных таблицах проверок недостаточно. Обычно реализуется в виде бизнес-правил. Например, приложение, обрабатывающая медицинские страховки может получить запрос на выплату 123,5 долларов при максимальном годовом накоплении в 1500 долларов. В такой ситуации проверка данных распространяется на оценку способов оплаты, зависящих от ограничений полиса и годовых начислений.

Операционные процессы

Регулярные операционные процессы, проверяющие целостность данных — обязательная составляющая любого приложения, нуждающегося в сопровождении. Операционные процессы включают:

- обслуживание баз данных;
- резервное копирование данных.

Обслуживание базы данных

Если разработчики отвечают и за обслуживание баз данных, в их обязанности входят периодическое выполнение определенных действий. В случае реляционных баз данных это очистка журнала, проверка пиковой загрузки памяти и процедурного кэша, сжатие файлов и проверка связей между таблицами и индексами. В иерархических базах данных надо проверять разорванные связи, тщательно исследуя структуру всех записей.

Резервное копирование данных

Резервное копирование позволяет восстановить поврежденные данные. Существует несколько способов реализации резервного копирования:

- создание копии;
- зеркализация;
- репликация.

При первом способе данные копируются на внешнее устройство, такое, как диск или лента, в особом формате. Так как при этом сохраняется вся база данных, изменения, сделанные после резервного копирования, не учитываются (если в копию не включен журнал транзакций). Этот метод очень популярен. Однако есть и недостаток: резервное копирование следует проводить по крайней мере раз в сутки.

Большинство реляционных баз данных поддерживают создание зеркальной копии — постоянное дублирование всех транзакций с одного устройства на другое, называемое зеркалом. Такой метод незаметен и не требует обслуживания. Основное его преимущество — мгновенное восстановление после сбоев. Однако если сетевое соединение с зеркалом нарушится, база данных перестанет функционировать. Кроме того, этот метод не исключает периодическое копирование на обычное резервное устройство.

Репликация — это копирование всей базы данных или ее частей на несколько удаленных компьютеров. Это не только метод резервного копирования информации, но и способ распространения данных по сети, распределения нагрузки и минимизации риска сбоев. Основное назначение репликации — поддержка согласованности информация в распределенных базах данных, поэтому обычно ее не рассматривают как средство резервного копирования.

Выбор технологии хранения данных

Существует огромное число технологий хранения данных. Самые популярные - индексируемая, иерархическая и реляционная Такие типы хранилищ отличаются не столько способом физического хранения и получения данных, сколько своими концептуальными моделями.

Индексируемые базы данных, обеспечивающие исключительно быструю выборку информация, удобны для реализация последовательных списков, произвольных выборок

данных и сложных файловых взаимосвязей. С помощью индекса легко считать файл полностью или одну запись. Индексированные базы данных, как правило, поддерживают только хранение и выборку информации. За целостность ссылок и проверки данных должно отвечать приложение.

Иерархические базы данных подходят для реализация обращенных древовидных структур, например списков материалов или штатного расписания. Доступ к иерархическим данным чрезвычайно быстр, так как структуры данных связаны напрямую. Кроме того, в них встроен механизм обеспечения целостности ссылок. Однако для их реализация иногда нужен опытный разработчик, который сможет компенсировать характерное для таких баз данных отсутствие возможностей моделирования сложных связей.

Реляционные базы данных давно стали стандартом хранения информация. Они намного предпочтительнее остальных технологий из-за их удобства. К тому же они обладают стандартным интерфейсом — языком структурированных запросов (Structure Query Language, SQL) — в рамках которого возможна согласованная работа многочисленных инструментальных средств. Кроме того, в реляционных базах данных предусмотрены механизмы обеспечения ссылочной целостности, проверки данных и сопровождения БД.

Работа над приложением начинается с создания концептуального проекта, включающего модель объектов, определение специфических данных и требований к связям между ними. Зная эти требования, вы распределите объекты по компонентам, соблюдая следующие правила:

- за точность, полноту и согласованность данных отвечают владеющие ими объекты данных;
- объекты данных должны работать корректно независимо от того, является ли обращающийся к ним объект транзакционным;
- следует внимательно изучить способы использования объектов данных и издержки доступа к информации, связанные с сохранением состояния во время вызова методов. Этот пункт особенно важен, так как объекты данных не способны сохранять состояние вне границ транзакции;
- чем меньше двусторонних перемещений объектов, тем лучше, поэтому объекты данных должны поддерживать сетевые операция. Кроме того, сокращение таких перемещений улучшает производительность приложения.

К каждому хранилищу информация определен присущий только ему метод доступа. Всякий разработчик баз данных создает свой API доступа к информации. Например, к данным, хранящимся не в базе данных, можно обратиться средствами API службы каталогов Windows NT , MAPI (если это почтовые сообщения) или API файловой системы. Использовать все возможности хранилища разрешается только в одном случае – применяя его собственный API. Однако при этом приходится изучать множество методов доступа. К тому же, разработчики должны разобраться во всех функциях API , выбрать из них наиболее эффективные, научиться пользоваться средствами диагностики и настройки хранилища. Естественно, затраты на обучение персонала всем API доступа к данным, применяемым в вашей организации, становятся очень высокими.

Разработчикам распределенных приложений приходится решать две технические проблемы, связанные с доступом к данным. Во-первых, им редко предоставляется возможность начать работу «с нуля». Ведь, как правило, приложения должны обращаться к уже существующим данным, хранящимся в различных форматах. Например, какая-то часть информации содержится в СУБД, а другая — в менее структурированной форме. На рынке программного обеспечения представлено множество СУБД, в том числе и традиционные СУБД для мэйнфреймов (Information Management Systems (IMS) и DB2), и клиент-серверные СУБД (Oracle и SQL Server), и СУБД для персональных компьютеров (Microsoft Access и Paradox). Одно приложение может использовать несколько СУБД, а для хранения других данных - текстовые файлы,

файлы индексно-последовательного доступа, электронные таблицы, сообщения электронной почты и других форматов. Разработчикам же распределенных приложений придется придумывать способ объединения различных источников информации.
Во-вторых, распределенные приложения обращаются к удаленным источникам данных. В большинстве случаев пользователи работают на разных компьютерах, причем не на тех, где хранится информация. Поэтому, чтобы минимизировать сетевой трафик, необходим эффективный механизм доступа к удаленным данным. Это становится особенно важным при увеличении числа пользователей приложения до нескольких тысяч или миллионов, многие из которых подключены к сети как через глобальные вычислительные сети с довольно высокой пропускной способностью, так и с помощью медленных модемных соединений.
В состав компонентов доступа к данным MDAC входят:

- ADO — программный интерфейс данных уровня приложения;
- Remote Data Service (RDS) — механизм кэширования данных на стороне клиента;
- компонент доступа Microsoft OLE DB для ODBC — компонент доступа к данных ODBC средствами OLE DB ;
- диспетчер драйверов ODBC — библиотека, реализующая ODBC API; непосредственно вызывает соответствующие драйверы ODBC;
- драйверы ODBC — драйверы баз данных SQL Server , Access и Oracle .

ODBC
Избежать применения «родных» методов доступа к данным можно посредством независимых API , таких, как Microsoft ODBC. ODBC — это программный интерфейс на основе языка C для доступа к информации СУБД средствами языка SQL. Диспетчер драйверов ODBC преобразует вызовы ODBC в вызовы собственного API базы данных, позволяя приложению найти необходимый драйвер и использовать поддерживаемые им методы доступа к базе данных.
Основное преимущество ODBC заключается в том, что разработчикам приходится изучать только один API, посредством которого они смогут получить доступ ко многим СУБД. Кроме того, приложения могут получать информацию из нескольких СУБД. К тому же тип используемой базы данных не обязательно определять заранее – окончательное решение не поздно принять и при развертывании приложения.
К сожалению, у такого метода есть и недостатки. Во-первых, для каждого хранилища информации должен существовать собственный драйвер ODBC, причем он должен поддерживать запросы SQL, даже если собственный API базы данных их не использует. Во-вторых, API ODBC работает с данными в реляционной форме. Оба эти ограничения способны вызвать проблемы с неструктурированными и нереляционными хранилищами информации. И наконец, API ODBC — стандарт; другими словами, независимо от возможностей СУБД драйверу будет доступен только ограниченный набор ее функций, включенный в стандарт. Ведь модификация API — очень сложный процесс. Комиссия должна принять предложения, определить способы их реализации в драйверах и способы определения новых возможностей приложениями. Только затем можно обновить драйверы тем не менее приложениям приходится либо проверять их наличие, либо работать со старыми драйверами.
На практике механизм ODBC используется довольно широко, он отлично подходит для приложений, работающих с традиционными реляционными базами данных. Эту технологию поддерживает большинство крупных производителей СУВД. Однако, как только приложение выходит за рамки реляционных СУБД для его реализации требуются более сложные методы.