Технологии баз данных

Конспект лекций

Москва

1. Аннотация

Цель дисциплины “Технологии баз данных” - ознакомление слушателей с организацией, принципами построения и функционирования современных систем управления базами данных, с методами моделирования данных, принципами построения приложений баз данных.

Курс по выбору для студентов I курса, читается в первом семестре.

1. **Информационные системы**

Общие понятия баз данных. Системы управления базами данных. Функции, возлагаемые на системы управления базами данных. Взаимодействие типа клиент-сервер.

Системы хранения данных исторически являлись одними из первых приложений для вычислительных систем. Прошли эволюцию от специализированных систем до базовых элементов полных комплексов по поддержке всего жизненного цикла бизнес-приложений. Пример – Oracle.

Основные определения:

База данных – именованная совокупность данных, отражающая состояние объектов и их отношения.

СУБД (Система Управления Базами Данных) – совокупность языковых и программных средств, предназначенная для создания, ведения и использования баз данных.

Позиции, по которым сравниваются системы:

А) Поддерживаемая модель данных

Модель данных определяется совокупностью трех позиций:

1. Структура данных (то есть, по сути, типы данных)
2. Операции над данными
3. Ограничения целостности

Последняя позиция является черезвычайно важной – ограничения целостности есть неотъемлемая часть модели.

Б) Средства администрирования

Сюда обычно включаются следующие элементы:

- стандартные данные (словари)
- управление представлением БД в среде хранения
- сбор и анализ статистики
- реорганизация данных (реструктуризация, реформатизация)
- проверка целостности и восстановление данных
- конвертирование данных

В) Средства разработки

В этот раздел традиционно включают следующие позиции:

- поддержка процесса проектирования
- интерфейс с языками программирования
- генерация программ (отчетов, форм ввода-вывода, пользовательских интерфейсов)

Д) Средства для пользователей

- языки запросов
- пользовательские интерфейсы

2. Модели данных

Рассматриваются вопросы построения моделей данных, описываются иерархическая, сетевая, реляционная и объектно-ориентированная модели данных.

Основные моменты:

- класс структур данных (типы)
- операции над ними
- ограничения целостности

Необходимо четко разграничивать модель данных и модель (представление) бизнес процессов (описание предметной области). Задача, например, может моделироваться некоторой иерархической системой (модель организации) и, в то же время, реализовываться с помощью реляционной модели данных

Базовая терминология:

Атрибут (элемент данных) - наименьшая единица структуры данных. Обычно каждому элементу при описании базы данных присваивается уникальное имя. По этому имени к нему обращаются при обработке. Атрибут также часто называют полем.

Запись - некоторая совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются. Экземпляр записи – это конкретная запись с конкретным значением атрибутов.

Исторически первой была модель плоских файлов. Эта модель тесно связана с системой хранения. Данные представляются как набор записей, которые сохраняются последовательно на диске в одном файле. Нет других отношений между данными, кроме физического порядка сохранения.

Класс операций, который хорошо поддерживается такой моделью – это последовательный перебор.

Здесь же впервые возникло понятие индекса – сылки на данные, упорядоченные по некоторому условию.

Развитие модели – записи содержат элементы данных определенного типа. В этой связи говорят о структурированных файлах.

В настоящее время имеет применения в системах с ограниченными ычислительными возможностями. Например, средства сохранения данных в J2ME.

Иерархическая модель данных.

Иерархическая база данных хранит древовидные структуры. Дерево здесь соответствует определению из теории графов – связный граф без циклов.

Иными словами, в системе существуют иерархические отношения между записями двух типов. Родительская запись еще называется исходной записью, а дочерние записи - подчиненными.

Корневая (родительская) запись каждого дерева обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках данного отношения. Каждая запись идентифицируется полным соединенным ключом, под которым понимается совокупность ключей всех записей, начиная от корневой записи. Это соответствует модели обхода дерева.

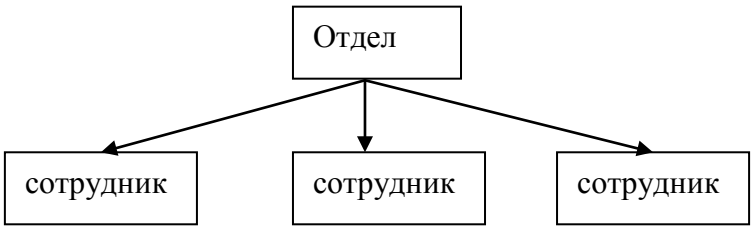
При графическом изображении отношения изображают дугами ориентированного графа, а типы записей - вершинами этого дерева.

Иерархическая модель реализует отношение между исходной и дочерней записью по схеме 1:N, то есть одной родительской записи может соответствовать любое число дочерних записей.

При добавлении подчиненной записи должна, с очевидностью, уже существовать родительская запись. При удалении родительской записи автоматически удаляются все подчиненные ей записи.

Пример. Административная система: предприятие состоит из отделов, в которых работают сотрудники. В каждом отделе может работать несколько сотрудников. Есть некоторые заказчики, по контрактам с которыми работают исполнители из числа сотрудников.

Первое отношение будет включать родительскую запись для отдела и подчиненные записи для сотрудников.



Каждая запись имеет, естественно, свой набор атрибутов. Например:

ОТДЕЛ (НАИМЕНОВАНИЕ_ОТДЕЛА, ЧИСЛО_РАБОТНИКОВ)
СОТРУДНИК (ФАМИЛИЯ, ДОЛЖНОСТЬ, ОКЛАД).

Для учета контрактов с заказчиками необходимо создание еще одной иерархической структуры: заказчик → контракты с ним → сотрудники, работающие над контрактом.

Опять таки, записи будут иметь свой набор атрибутов. Например:

ЗАКАЗЧИК(НАИМЕНОВАНИЕ_ЗАКАЗЧИКА, АДРЕС),
КОНТРАКТ(НОМЕР, ДАТА,СУММА),
ИСПОЛНИТЕЛЬ (ФАМИЛИЯ, ДОЛЖНОСТЬ, НАИМЕНОВАНИЕ_ОТДЕЛА)

Иерархическая база данных и будет содержать набор подобного рода отношений.

На этом примере видно, что нам пришлось копировать информацию между записями СОТРУДНИК и ИСПОЛНИТЕЛЬ. Такие записи называются парными, и в

иерархической модели данных не предусмотрено никакой поддержки для указания соответствия между ними. Это проистекает из поддержки только отношений 1:N

В иерархической модели определяются следующие операции:

ДОБАВИТЬ в базу данных новую запись. Для корневой записи обязательно указание ключа.
ИЗМЕНИТЬ значение данных в записи. Ключевые поля при этом не могут модифицироваться
УДАЛИТЬ некоторую запись и все подчиненные ей записи.
ИЗВЛЕЧЬ запись:

- последовательно читать корневые записи
- извлечь корневую запись по ключу
- извлечь следующую запись в порядке обхода дерева

В операции ИЗВЛЕЧЬ допускается задание условий выборки (предикат над атрибутами)

В качестве ограничений целостности поддерживается только целостность связей между родительскими и подчиненными записями

Сетевая модель данных.

Основные принципы сетевой модели данных были разработаны в середине 60-х годов. Эталонный вариант сетевой модели данных описан в отчетах рабочей группы по языкам баз данных (COnference on DAta SYstem Languages) CODASYL.

Сетевая модель данных определяется в тех же терминах, что и иерархическая. Она состоит из множества записей, которые могут быть владельцами или членами отношений. Связь между записью-владельцем и записью-членом также имеет вид 1:N.

Основное различие этих моделей состоит в том, что в сетевой модели запись может быть членом более чем одного отношения. Вместо дерева, как в иерархической модели, мы имеем граф. При этом граф не обязан быть связным – могут быть записи, не входящие в отношения.

Согласно этой модели каждое отношение именуется и проводится различие между его типом и экземпляром. Тип отношения задается его именем и определяет свойства общие для всех экземпляров данного типа. Экземпляр отношения представляется родительской записью и множеством подчиненных записей.

При этом имеется следующее ограничение: экземпляр записи не может быть членом двух экземпляров отношений одного типа

Возарвщаясь к описанному выше примеру, запись сотрудник входит (в сетевой модели) в два отношения.

Для отображения типа M:N вводится запись СОТРУДНИК_КОНТРАКТ, которая не может не иметь полей и служить только для связи записей КОНТРАКТ и СОТРУДНИК

Каждый экземпляр отношения характеризуется следующими признаками:

А) способ упорядочения подчиненных записей:
- произвольный,
- хронологический /очередь/,
- обратный хронологический /стек/,
- упорядоченный

Если запись объявлена подчиненной в нескольких групповых отношениях, то в каждом из них может быть назначен свой способ упорядочивания.

Б) режим включения подчиненных записей:
- автоматический - невозможно занести в БД запись без того, чтобы она была сразу же закреплена за неким владельцем;
- ручной - позволяет запомнить в БД подчиненную запись и не включать ее немедленно в экземпляр отношения. Отношение может быть установлено позднее

В) режим исключения. Принято выделять три класса членства подчиненных записей в групповых отношениях:
1. Фиксированное. Подчиненная запись жестко связана с записью-владельцем. Ее можно исключить из отношения только удалением. При удалении записи-владельца все подчиненные записи автоматически тоже удаляются.
2. Обязательное. Допускается переключение подчиненной записи на другого владельца, но невозможно ее существование без владельца. Для удаления записи-владельца необходимо, чтобы она не имела подчиненных записей с обязательным членством.
3. Необязательное. Можно исключить запись из отношения, но сохранить ее в базе данных, не прикрепляя к другому владельцу. При удалении записи-владельца ее подчиненные записи - необязательные члены сохраняются в базе, не участвуя более в отношении такого типа.

Набор операций похож на набор операций иерархической модели, но появляется новый параметр - отношение. В сетевой модели поддерживаются следующие операции над данными:

ДОБАВИТЬ - внести запись в БД. В зависимости от режима включения, новая запись может быть включена в некоторое отношение
ВКЛЮЧИТЬ В ОТНОШЕНИЕ - связать существующую запись с родительской записью.
ИСКЛЮЧИТЬ ИЗ ОТНОШЕНИЯ - разорвать связь между записями
ИЗМЕНИТЬ ОТНОШЕНИЕ - связать существующую подчиненную запись с другой записью в том же отношении.
ИЗМЕНИТЬ - изменить значение атрибутов записи.
ИЗВЛЕЧЬ - извлечь записи последовательно по значению ключа. Можно также выбирать записи по отношениям - от владельца можно перейти к записям - членам, а от подчиненной записи к владельцу набора.
УДАЛИТЬ – удалить запись из базы. Если эта запись является владельцем группового отношения, то анализируется класс членства подчиненных записей. Обязательные члены должны быть предварительно исключены из отношения, фиксированные удалены вместе с владельцем, необязательные останутся в БД.

В смысле ограничений целостности здесь все, как и в сетевой модели - обеспечивается только поддержание целостности по ссылкам

Реляционная модель данных

Как и для любой модели, мы должны описать типы (структуры) данных, операции над ними и ограничения целостности.

Единственной структурой данных, используемой в реляционной модели, являются нормализованные n-арные отношения.

В качестве набора операций рассматриваются два эквивалентных способа манипулирования реляционными данными - реляционная алгебра и реляционное исчисление.

В качестве ограничений целостности рассматриваются целостность сущностей и целостность внешних ключей.

Домен – совокупность однотипных значений данных

Атрибут – представляет свойства объекта. Несколько атрибутов могут получать значения из одного домена.

Отношение R определяется как набор упорядоченных n-ок, являющихся подмножеством декартова произведения доменов.

Отношение является нормализованным, если каждая компонента кортежа (n-ки) является простым атомарным значением. Нормализованное отношение может быть представлено в виде таблицы. Имя таблицы – имя отношения. Имена столбцов – имена атрибутов. Строки таблицы – кортежи.

Число атрибутов в отношении называют степенью (или -арностью) отношения.

Число кортежей называется мощностью отношения (кардинальное число).

Реляционной базой данных называется набор изменяющихся во времени отношений.

Схемой реляционной базы данных называется набор заголовков отношений, входящих в базу данных.

Любое отношение можно изобразить в виде таблицы.

1. В отношении нет одинаковых кортежей. Поскольку отношение есть множество кортежей, то, как всякое множество, оно не может содержать неразличимые элементы.
2. Кортежи не упорядочены. Причина та же - отношение есть множество, а множество не упорядочено.
3. Атрибуты в отношении не упорядочены. Так как каждый атрибут имеет уникальное имя в пределах отношения, то порядок атрибутов не имеет значения.

Навигация по отношениям осуществляется путем их соединения с помощью атрибутов, определенных над общими или сравнимыми доменами.

Объектно-ориентированные базы данных

Направление объектно-ориентированных баз данных (ООБД) возникло в середине 1980-х годов.

База для ООДБ вполне естественна – объектно-ориентированные языки программирования. Собственно, одно из основных преимуществ объектно-ориентированных баз данных - это отсутствие разрыва между моделью данных и ее представлением (отражением) в средствах разработки, где господствуют объектно-ориентированные системы (C++, Java, C#).

В классической постановке объектно-ориентированный подход базируется на следующих понятиях:

- объект и его идентификат;
- класс;
- атрибуты и методы;
- иерархии и наследования классов.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом во все время его существования и не меняется при изменении состояния объекта.

Каждый объект характеризуется своим состоянием и поведением. Состояние объекта – это набор значений его атрибутов. Поведение объекта – это набор методов (операций) для данного объекта.

В качестве значения атрибута может также выступать некоторый объект (множество объектов)

Взаимодействие между объектами производится путем выполнения соответствующих методов и/или на основе передачи сообщений между объектами.

Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов.

Допускается порождение нового класса на основе уже существующего класса - наследование. В этом случае новый класс, называемый подклассом существующего класса (суперкласса) наследует все атрибуты и методы суперкласса. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Наследование может быть простым и множественным.

Объектно-ориентированная система БД представляет собой объединение системы программирования и СУБД, основанной на объектно-ориентированной модели данных.

Объекты и значения могут быть именованными. С именованием объекта или значения связана долговременность его хранения (persistency): любые именованные объекты или значения долговременны; любые объект или значение, входящие как часть в другой именованный объект или значение, долговременны.

Средства работы с ООБД включаются в язык программирования. Это означает, что работа с объектами, хранимыми во внешней БД, должна происходить на основе тех же синтаксических конструкций и с такой же семантикой, что и работа с обычными, существующими только во время работы программы объектами.

3. Реляционная алгебра

Рассматриваются положения, лежащие в основе реляционной модели данных, как наиболее часто используемой в коммерческих приложениях. Операции реляционной модели данных.

Реляционная алгебра представляет собой набор операторов, использующих отношения в качестве аргументов, и возвращающие отношения в качестве результата. Важно, таким образом, то, что результат всегда будет отношением.

В качестве аргументов в реляционные операторы можно подставлять другие реляционные операторы, подходящие по типу. Иными словами, реляционные выражения могут быть вложенными. Утверждение, которое может встретиться в этой связи, говорит о том, что реляционная алгебра является замкнутой.

Обычно, следуя оригинальной работе Кодда, определяют восемь реляционных операторов:

- Объединение
- Вычитание
- Декартово произведение
- Выборка
- Проекция
- Соединение
- Пересечение
- Деление

Объединение двух отношений R1 и R2 порождает отношение из кортежей, принадлежащих или R1, или R2 или обоим отношениям

Пересечение двух отношений R1 и R2 порождает отношение из кортежей, принадлежащих одновременно обоим отношениям

Вычитание двух отношений R1 и R2 порождает отношение из кортежей, принадлежащих R1 и не принадлежащих R2

Декартово произведение двух отношений R1 и R2 порождает отношение, заголовок которого получается конкатеницией заголовков R1 и R2, а кортежи – конкатенацией кортежей отношений R1 и R2

Выборка из отношения R1, есть новое отношение, куда входят кортежи из R1, атрибуты которых удовлетворяют некоторому логическому условию

Проекция отношения R есть новое отношение, где кортежи содержать часть атрибутов исходного отношения.

Соединение есть результат последовательного применения операций декартового произведения и выборки.

Результатом деления двух отношений R1 и R2 будет новое отношение R3, включающее такие кортежи r3 , что в отношении R1 содержатся кортежи r3 UNION r2, где r2 есть кортежи из R2

Не все реляционные операторы являются независимыми, т. е. некоторые из реляционных операторов могут быть выражены через другие реляционные операторы.

Некоторые из введенных операторов (соединения, пересечения, деления) можно выразить через другие. Остальные операторы являются примитивными (базовыми) – они не могут быть реализованы с помощью других.

Вместе с тем существует набор запросов (операций над данными), которые не могут быть выражены с помощью набора реляционных операторов. Это приводит к тому, что реализации реляционной алгебры будут всегда дополняться процедурными (не теоретико-множественными) расширениями.

4. Нормальные формы

Нормальные формы описывают ограничения, налагаемые на отношения в базе данных, призванные устранить возможные проблемы с модификацией этих данных. Процесс нормализации это, собственно, процедура (последовательность шагов), связанная с приведением отношений к форме, удовлетворяющей определению “нормальной”.

Отношение находится в первой нормальной форме, если значения в каждом домене являются атомарными.

Определение: если даны два атрибута X и Y некоторого отношения, то говорят, что Y функционально зависит от X, если в любой момент времени каждому значению X соответствует ровно одно значение Y.

Функциональные зависимости представляют собой связи типа "один ко многим", существующие внутри отношения.

Избыточная функциональная зависимость - зависимость, заключающая в себе такую информацию, которая может быть получена на основе других зависимостей, имеющихся в базе данных.

Корректной считается такая схема базы данных, в которой отсутствуют избыточные функциональные зависимости. В противном случае приходится прибегать к процедуре декомпозиции (разложения) имеющегося множества отношений. При этом порождаемое множество содержит большее число отношений, которые являются проекциями отношений исходного множества. Обратимый пошаговый процесс замены данной совокупности отношений другой схемой с устранением избыточных функциональных зависимостей и называется нормализацией.

Определение первой нормальной формы: отношение находится в 1NF если значения всех его атрибутов атомарны.

Определение: неключевой атрибут функционально полно зависит от составного ключа если он функционально зависит от всего ключа в целом, но не находится в функциональной зависимости от какого-либо из входящих в него атрибутов.

Определение второй нормальной формы: отношение находится во 2НФ, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от ключа.

Определение третьей нормальной формы: отношение находится в 3НФ, если оно находится во 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Определение нормальной формы Бойса-Кодда: отношение находится в нормальной форме Бойса-Кодда, если оно находится во 3НФ и в нем отсутствуют зависимости атрибутов первичного ключа от неключевых атрибутов.

Четвертая нормальная форма касается отношений, в которых имеются повторяющиеся наборы данных. Декомпозиция, основанная на функциональных

зависимостях, не приводит к исключению такой избыточности. В этом случае используют декомпозицию, основанную на многозначных зависимостях.

Определение четвертой нормальной формы: отношение находится в 4NF, если оно находится в нормальной форме Бойса-Кодда и в нем отстутсвуют многозначные зависимости, не являющиеся функциональными зависимостями.

Определение пятой нормальной формы: отношение находится в 5НФ тогда и только тогда, когда любая зависимость по соединению в нем определяется только его возможными ключами. Другими словами, каждая проекция такого отношения содержит не менее одного возможного ключа и не менее одного неключевого атрибута.

Процесс нормализации всегда ведет к увеличению количества отношений (таблиц) в реляционной базе данных. Естественно, выборка информации в таком случае будет основана на соединении таблиц. Очевидно, что чем больше таблиц участвует в соединении, тем больше будет затраченное время. Не вдаваясь в детали реализации систем хранения в современных базах данных, чтение из одного файла всегда будет работать быстрее чтения из нескольких. Поэтому нормализация, устраняя возможные проблемы с обновлением данных, ведет (или может приводить, если точнее), в общем случае, к проблемам с производительностью при чтении данных.

На практике, как правило, используются нормальные формы не выше третьей. Многие Case Tools, позволяющие автоматически строить схемы данных по спецификациям, также ориентированы на третью нормальную форму.

Также, в практических приложениях, иногда сознательно идут на нарушения условий нормализации. Делается это с целью повысить производительность системы.

Отступления от требований 1-й нормальной формы ведут к появлению так называемых пост-реляционных баз данных (реляционных баз данных с сетевыми возможностями).

5. SQL – как основа взаимодействия с реляционными базами.

Принципы построения языка SQL. Стандартизация языка SQL. Определение данных (DDL), ограничения целостности. Операции модификации данных (DML). Транзакции. View.

Две ветви SQL:

- описание данных DDL
- собственно работа с данными (манипулирование) DML

DDL описывает, в первую очередь, создание объектов и, самое главное, ограничения целостности

CREATE TABLE имя_таблицы [AS]
(
{ имя_столбца тип_данных [(размер)] [тип_столбца] , …
}
) [ограничения]

тип данных: INTEGER, CHARACTER, DECIMAL, NUMERIC, SMALLINT FLOAT, REAL, DOUBLE, PRECISION, LONG, VARCHAR, DATE, TIME, TIMESTAMP

тип столбца: NOT NULL, UNIQUE, PRIMARY KEY, CHECK (предикат), DEFAULT = значение, REFERENCES имя_таблицы [имя столбца]

ограничения: PRIMARY KEY, CHECK (предикат), REFERENCES имя_таблицы [имя столбца]

Доступ к данным регламентирован для пользователей (операции GRANT, REVOKE)

DML – всего 4 оператора

SELECT – выборка данных
INSERT – добавление
UPDATE – обновление
DELETE - удаление

SELECT * | { [DISTINCT | ALL] <список полей> ,…}
FROM { <имя таблицы> [<алиас>]}
[WHERE <предикат>]
[GROUP BY { <имя столбца> | <целое число>}, …]
[HAVING <предикат>]
[ORDER BY { <имя столбца> | <целое число>}, …]

Оператор UNION позволяет соединить результаты нескольких операторов SELECT

INSERT INTO <имя таблицы> [(<имя столбца>, …)]
{ VALUES (<список полей>) }
| <запрос>

UPDATE <имя таблицы>
SET [(<имя столбца> = значение, …)]

[WHERE <предикат>]

DELETE FROM <имя таблицы>
[WHERE <предикат>]

Встроенные функции.

Могут варьироваться в зависимости от сервера. Общие группы:

- математические функции;
- строковые функции;
- функции для работы с датой и временем;
- функции конфигурирования;
- функции системы безопасности;
- функции управления метаданными;
- статистические функции.

Наример:

ACOS вычисляет арккосинус
ASIN вычисляет арксинус
ATAN вычисляет арктангенс
ATN2 вычисляет арктангенс с учетом квадратов
CEILING выполняет округление вверх
COS вычисляет косинус угла
COT возвращает котангенс угла
DEGREES преобразует значение угла из радиан в градусы
EXP возвращает экспоненту
FLOOR выполняет округление вниз
LOG вычисляет натуральный логарифм
LOG10 вычисляет десятичный логарифм
PI возвращает значение "пи"
POWER возводит число в степень
RADIANS преобразует значение угла из градуса в радианы
RAND возвращат случайное число
ROUND выполняет округление с заданной точностью
SIGN определяет знак числа
SIN вычисляет синус угла
SQUARE выполняет возведение числа в квадрат
SQRT извлекает квадратный корень
TAN возвращает тангенс угла

Строковые функции

CHAR по коду ASCII возвращает символ
CHARINDEX определяет порядковый номер символа, с которого начинается вхождение подстроки в строку
DIFFERENCE возвращает показатель совпадения строк
LEFT возвращает указанное число символов с начала строки
LEN возвращает длину строки
LOWER переводит все символы строки в нижний регистр
LTRIM удаляет пробелы в начале строки
NCHAR возвращает по коду символ Unicode

PATINDEX выполняет поиск подстроки в строке по указанному шаблону
REPLACE заменяет вхождения подстроки на указанное значение
QUOTENAME конвертирует строку в формат Unicode
REPLICATE выполняет тиражирование строки определенное число раз
REVERSE возвращает строку, символы которой записаны в обратном порядке
RIGHT возвращает указанное число символов с конца строки
RTRIM удаляет пробелы в конце строки
SOUNDEX возвращает код звучания строки
SPACE возвращает указанное число пробелов
STR выполняет конвертирование значения числового типа в символьный формат
STUFF удаляет указанное число символов, заменяя новой подстрокой
SUBSTRING возвращает для строки подстроку указанной длины с заданного символа
UNICODE возвращает Unicode-код левого символа строки
UPPER переводит все символы строки в верхний регистр

Функции для работы с датой и временем

DATEADD добавляет к дате указанное значение дней, месяцев, часов и т.д.
DATEDIFF возвращает разницу между указанными частями двух дат
DATENAME выделяет из даты указанную часть и возвращает ее в символьном формате
DATEPART выделяет из даты указанную часть и возвращает ее в числовом формате
DAY возвращает число из указанной даты
GETDATE возвращает текущее системное время
ISDATE проверяет правильность выражения на соответствие одному из возможных форматов ввода даты
MONTH возвращает значение месяца из указанной даты
YEAR возвращает значение года из указанной даты

Представления, или просмотры (VIEW), представляют собой временные, производные (иначе - виртуальные) таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним.

Представление - это предопределенный запрос, хранящийся в базе данных, который выглядит подобно обычной таблице и не требует для своего хранения дисковой памяти.

Представления используются там же, где используются таблицы. Предназначены для упрощения записи сложных запросов, сокрытия структуры базы данных и, самое главное, для разграничения доступа к строкам таблицы.

<определение_просмотра> ::=
{ CREATE| ALTER} VIEW имя_просмотра
[(имя_столбца [,...n])]
AS SELECT_оператор
[WITH CHECK OPTION]

VIEW может быть обновляемым или нет.

Модифицируемое представление определяется следующими критериями:

- основывается только на одной базовой таблице;
- содержит первичный ключ этой таблицы;
- не содержит DISTINCT в своем определении;
- не использует GROUP BY или HAVING в своем определении;
- не использует константы или выражения значений среди выбранных полей вывода;
- оператор SELECT просмотра не использует агрегирующие (итоговые) функции, соединения таблиц, хранимые процедуры и функции, определенные пользователем;
- основывается на одиночном запросе, поэтому объединение UNION не разрешено.

Если просмотр удовлетворяет этим условиям, к нему могут применяться операторы INSERT, UPDATE, DELETE.

Транзакции определяют группы операций, которые либо должны выполниться целиком, либо база данных должна остаться в исходном состоянии.

Группы операций выделяются скобками:

BEGIN TRANSACTION
…
END TRANSACTION

Оператор COMMIT позволяет зафиксировать результаты транзакции
Оператор ROLLBACK позволяет отменить результаты работы

Транзакции могут быть вложенными и именованными

С транзакциями связано понятие уровня изоляции: SET ISOLATION LEVEL:

READ UNCOMMITED
READ COMMITED
READ REPEATABLE
SERIALIZABLE

6. Поддержка SQL операций на уровне сервера.

Курсоры, триггеры, сохраненные процедуры.

SQL программы (фрагменты кода) представляют из себя обычный текст. В этом смысле нет отличия от программ на других языках. Очевидно, что технически нет никаких препятствий для хранения таких текстовых фрагментов вместе с самими данными.

Хранимые процедуры – это фрагменты кода на языке SQL, которые могут использоваться при формировании запросов. Разделяются на две группы:

- процедуры
- функции

Различие – только в одном. Функции вырабатывают значение

Функции (процедуры) пользователя представляют собой самостоятельные объекты базы данных. Функция пользователя располагается в определенной базе данных и доступна только в ее контексте.

Конкретный синтакис хранимых процедур может варьироваться в SQL серверах. Например, для MS SQL:

```
<определение_функции>::=
{CREATE | ALTER } FUNCTION     имя_функции
( [ { @имя_параметра тип_данных
   [=default]}[,...n]])
RETURNS скаляр_тип_данных
 [AS]
BEGIN
<тело_функции>
RETURN скаляр_выражение
END
```

Функция может содержать один или несколько входных параметров либо не содержать ни одного. Каждый параметр должен иметь уникальное в пределах создаваемой функции имя и начинаться с символа "@". После имени указывается тип данных параметра. Дополнительно можно указать значение, которое будет автоматически присваиваться параметру (DEFAULT), если пользователь явно не указал значение соответствующего параметра при вызове функции.

С помощью конструкции RETURNS скаляр_тип_данных указывается, какой тип данных будет иметь возвращаемое функцией значение.

Между ключевыми словами BEGIN...END указывается набор команд, они и будут являться телом функции.

Когда в ходе выполнения кода функции встречается ключевое слово RETURN, выполнение функции завершается и как результат ее вычисления возвращается значение, указанное непосредственно после слова RETURN. Отметим, что в теле функции разрешается использование множества команд RETURN, которые могут возвращать различные значения. В качестве возвращаемого значения допускаются как обычные константы, так и сложные выражения. Единственное условие – тип

данных возвращаемого значения должен совпадать с типом данных, указанным после ключевого слова RETURNS.

Определение триггера в стандарте языка SQL

Триггеры являются частным случаем хранимых процедур. Их исполнение привязано к событиям, происходящим с таблицами. Иными словами, выполнение триггеров (соответствующих хранимых процедур) происходит при выполнении для таблицы определенных операторов SQL.

Триггер всегда связан с конкретной таблицей. Триггер автоматически запускается при модификации данных в ассоциированной таблице (до или после модификации).

При этом очень важно, что действия, определенные в триггере находятся в той де транзакции, что и инициировавший триггер оператор. Это автоматически гарантируется SQL сервером. Соответственно, при откате транзакции из триггера будут отменены как его изменения, так и возможные изменения, произведенные инициировавшим его оператором.

Типичные примеры использования:

- выполнение действий по умолчанию (например, автозаполнение)
- проверка корректности данных
- выполнение (проверка) ограничений целостности, которые сложно (невозможно) описать на декларативном уровне
- репликация данных
- ведение журналов (логов)

Основной формат команды CREATE TRIGGER показан ниже:

<Определение_триггера>::=
CREATE TRIGGER имя_триггера
BEFORE | AFTER <триггерное_событие>
ON <имя_таблицы>
[FOR EACH { ROW | STATEMENT}]
[WHEN(условие_триггера)]
<тело_триггера>

триггерные события состоят из вставки, удаления и обновления строк в таблице. В последнем случае для триггерного события можно указать конкретные имена столбцов таблицы. Точка запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанных с ним событий) или AFTER (после их выполнения).

Выполняемые триггером действия задаются для каждой строки (FOR EACH ROW), охваченной данным событием, или только один раз для каждого события (FOR EACH STATEMENT).

Понятие курсора

Запрос к реляционной базе данных может возвращать несколько записей, но приложение обрабатывает записи по одной. Курсор как объект базы данных позволяет обращаться к результатам выборке с точностью до записи. Курсор по факту – это указатель на текущую запись в выборке. Текущий указатель может перемещаться в двух направлениях по выборке.

Формально, курсор в SQL – это область в памяти базы данных, которая предназначена для хранения последнего оператора SQL. Эта область памяти поименована и доступна для прикладных программ.

Обычно курсоры используются для выбора из базы данных некоторого подмножества хранимой в ней информации. В каждый момент времени прикладной программой может быть проверена одна строка курсора.

Курсоры могут также неявно создаваться сервером базы данных.

В соответствии со стандартом SQL при работе с курсорами можно выделить следующие основные действия:

- создание или объявление курсора;
- открытие курсора, т.е. получение данных
- выборка из курсора и изменение с его помощью строк данных
- закрытие курсора
- освобождение курсора (освобождение памяти)

Курсоры могут быть обновляемыми или доступными только по чтению, скроллируемыми или нет, чувствительными или нет.

Курсоры могут позволять выбирать данные только в одном направлении – от начала к концу. Для скроллируемых (прокручиваемых) курсоров допускается перемещение в обоих направлениях и переход к произвольной строке результирующего набора данных.

Если возможна модификация данных, на которые указывает курсор, то он называется обновляемым (модифицируемым).

Чувствительность курсора относится к способности определять изменения в базе данных в процессе прокрутки (чтения данных). Нечувствительные курсоры еще называют статическими.

В схеме со статическим курсором информация читается из базы данных один раз и хранится в виде моментального снимка (snapshot - по состоянию на некоторый момент времени). Вносимые другими рользователями изменения не видны. Статический курсор не изменяется после создания и всегда отображает тот набор данных, который существовал на момент его открытия.

Чувствительный (динамический курсор), естественно, требует больших ресурсных затрат.

Управление курсором реализуется путем выполнения следующих команд:

- DECLARE – создание или объявление курсора;

- OPEN – открытие курсора, т.е. наполнение его данными;
- FETCH – выборка из курсора и изменение строк данных с помощью курсора;
- CLOSE – закрытие курсора;
- DROP (DEALLOCATE) – удаление курсора как объекта.

Объявление курсора похоже на объявление VIEW:

```
<создание_курсора>::=
  DECLARE имя_курсора
    [INSENSITIVE][SCROLL] CURSOR
  FOR SELECT_оператор
  [FOR { READ_ONLY | UPDATE
    [OF имя_столбца[,...n]]}]
```

При использовании ключевого слова INSENSITIVE будет создан нечувствительный курсор.

При указании ключевого слова SCROLL созданный курсор можно прокручивать в любом направлении, что позволяет применять любые команды выборки. Если этот аргумент опускается, то курсор окажется последовательным, т.е. его просмотр будет возможен только в одном направлении – от начала к концу.

SELECT-оператор задает тело запроса SELECT, с помощью которого определяется результирующий набор строк курсора.

При указании аргумента FOR READ_ONLY создается курсор "только для чтения", и никакие модификации данных не разрешаются. Он отличается от статического, хотя последний также не позволяет менять данные. В качестве курсора "только для чтения" может быть объявлен чувствительный курсор, что позволит отображать изменения, сделанные другим пользователем.

Создание курсора с аргументом FOR UPDATE позволяет выполнять в курсоре изменение данных либо в указанных столбцах, либо, при отсутствии аргумента OF имя_столбца, во всех столбцах.

Для открытия курсора и наполнения его данными из указанного при создании курсора запроса SELECT используется следующая команда:

```
OPEN имя_курсора
```

После открытия курсора происходит выполнение связанного с ним оператора SELECT.

Выборка данных (перемещение указателя) происходит по команде FETCH:

```
FETCH [NEXT | PRIOR | FIRST | LAST
  | ABSOLUTE {номер_строки
  | RELATIVE {номер_строки |
]
  FROM ] имя_курсора
  [INTO @имя_переменной [,...n]]
```

При указании FIRST будет возвращена самая первая строка полного результирующего набора курсора, которая становится текущей строкой.

При указании LAST возвращается самая последняя строка курсора. Она же становится текущей строкой.

При указании NEXT возвращается строка, находящаяся в полном результирующем наборе сразу же после текущей. Теперь она становится текущей. По умолчанию команда FETCH использует именно этот способ выборки строк.

Ключевое слово PRIOR возвращает строку, находящуюся перед текущей строкой. Она и становится текущей.

Аргумент ABSOLUTE номер_строки возвращает строку по ее абсолютному порядковому номеру в полном результирующем наборе курсора. Номер строки можно задать с помощью константы или как имя переменной, в которой хранится номер строки. Переменная должна иметь целочисленный тип данных. Указываются как положительные, так и отрицательные значения. При указании положительного значения строка отсчитывается от начала набора, отрицательного – от конца. Выбранная строка становится текущей. Если указано нулевое значение, строка не возвращается.

Аргумент RELATIVE кол_строк возвращает строку, находящуюся через указанное количество строк после текущей. Если указать отрицательное значение числа строк, то будет возвращена строка, находящаяся за указанное количество строк перед текущей строкой. При указании нулевого значения возвратится текущая строка. Возвращенная строка становится текущей.

В конструкции INTO @имя_переменной [,...n] задается список переменных, в которых будут сохранены соответствующие значения столбцов возвращаемой строки. Порядок указания переменных должен соответствовать порядку столбцов в курсоре, а тип данных переменной – типу данных в столбце курсора. Если конструкция INTO не указана, то поведение команды FETCH будет напоминать поведение команды SELECT – данные выводятся на экран.

Закрытие курсора выполняется командой CLOSE:

CLOSE имя_курсора

7. Доступ к данным из прикладных приложений

DB Lib API, Embedded SQL

DB Lib есть, исторически, первый способ организации доступа к базам данных. Представляет собой библиотеку для доступа к данным из некоторого языка высокого уровня.

Могут различаться в деталях, но в целом будут следовать одной и той же модели:

- Установить соединение с базой данных
- Выполнить запрос
- Получить данные
- Закрыть соединение

Выполнение запроса это в любом случае передача некотрой строки с SQL текстом. Получение данных будет зависеть от типа запроса.

Запросы модификации данных (UPDATE, INSERT, DELETE) будут возвращать целое число, показывающее количество модифицированных строк.

Запросы выборки данных копируют инфорамцию из базы данных в локальный буфер. Дальнейшая работа с выборкой данных осуществляется уже средствами базового языка программирования.

Embedded SQL комбинирует язык высокого уровня (например, C, C++ или Java) и SQL. Реализуется как препроцессор для базового языка. В базовый язык добавляются конструкции языка SQL.

На первом шаге (препроцессирование) выполняется конвертация выделенных SQL операторов в процедурные вызовы. В итоге мы получаем программу на базовом языке программирования, которая уже и обрабатывается штатным компилятором.

Все SQL операторы начинаются с префикса EXEC SQL и заканчиваются символом ";".

```
{
    int a;
    /* ... */
    EXEC SQL SELECT salary INTO :a
            FROM Employee
            WHERE Id=876543210;
    /* ... */
    printf("The salary is %d\n", a);
    /* ... */
}
```

Здесь конструкция :a, используемая в операторе есть так называемая хост-переменная. Это переменная базового языка программирования, которая используется для передачи данных из/в SQL программы.

В SQL операторах используются с префиксом “:”.

Хост переменные описываются внутри секции деклараций:

```
EXEC SQL BEGIN DECLARE SECTION;
   // declarations...
EXEC SQL END DECLARE SECTION;
```

8. Стандартные интерфейсы для доступа

Стандартизация интерфейсов доступа к базам данных является естественным развитием подхода DB Lib. Библиотеки разных производителей концептуально выглядели (и выглядят) абсолютно одинаково. Вместе с тем, формальные различия в синтаксисе не позволяли добиться полной переносимости приложений.

Исторически, первым подходом к стандартизации интерфейсов являлся ODBC (Open Data Base Connectivity).

ODBC

ODBC закрепил следующие стандартные понятия для DBLib:

- используемый синтаксис SQL
- описания интерфейсных методов
- коды сообщений об ошибках
- взаимодействие с окружением (операционной средой). В частности, авторизация пользователей попадает сюда же.

Базовая компонента – драйвер. SQL команды могут, в частности, обрабатываться самим драйвером, что позволяет использовать SQL для запросов к произвольным ресурсам.

ODBC - это стандарт, с помощью которого можно создавать приложения, способные работать со многими СУБД. Условием является наличие драйвера, скрывающего вызовы DB Lib.

Схема работы такая же как и с DB Lib:

- Установить соединение с базой данных
- Выполнить запрос
- Получить данные
- Закрыть соединение

Запросы к базе данных можно выполнять двумя способами.

1. Прямой запрос с помощью SQLExecDirect, когда строка запроса непосредственно передаётся функции со всеми необходимыми параметрами.

2. Запрос сначала подготавливается с помощью SQLPrepare, затем если необходимо специальные символы в строке запроса, в ODBC это знак «?», можно связать с какими либо переменными, это делается с помощью SQLBindParameter, затем запрос отправляется в СУБД методом SQLExecute:

```
SQLPrepare(hstmt, "UPDATE Parts SET Price = ? WHERE PartID = ?", SQL_NTS);

// Связываем 1-й и 2-й параметры с переменными.
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_FLOAT,
   SQL_REAL, 7, 0, &Price, 0, &PriceInd);

SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_ULONG,
```

```
    SQL_INTEGER, 10, 0, &PartID, 0, &PartIDInd);

// выполняем запрос
SQLExecute(hstmt);
```

JDBC – Java ответ

Является аналогом ODBC для языка программирования Java. Поскольку был предложен позже, то была возможность учесть проблемы разработки и предложить более прогрессивные решения.

Общая схема работы также заключается в сокрытии прямого доступа к источнику данных драйвером. С программной точки зрения драйвер JDBC есть реализация интерфейсов предусмотренных API JDBC. По способу реализации драйверы подразделяются на 4 типа:

Тип 1: драйверы реализованные поверх ODBC драйверов. То есть, фактически, все вызовы API JDBC транслируются в вызовы ODBC, а дальше обработку вызова ведет API ODBC. Также называется "JDBC-ODBC bridge".
Тип 2: драйверы использующие программные части написанные на других языках программирования. Обычно в этом случае для доступа к базе данных используются библиотеки, разработанные производителем базы данных (dbLib)
Тип 3: драйвер полностью реализуется на Java, но при этом вызовы JDBC транслируются через Java RMI в специфичный протокол базы данных.
Тип 4: драйвер полностью реализуется на Java напрямую с использованием протокола базы данных.

Последовательность операций при работе через JDBC:

1. Загрузка драйвера JDBC
2. Установка соединения с базой данных
3. Формирование запроса

После получения соединения с базой данных (интерфейс Connection), необходимо создать так называемый SQLStatement — специальный Java объект для выполнения SQL запросов. Статический оператор создается для выполнения статических SQL запросов:

```
Statement st = con.createStatement();
```

Для выполнения SQL запроса используется метод

```
Statement.executeQuery(String query);
```

этот метод возвращает объект типа ResultSet. Это коллекция выбранных записей. текущую запись. Поля ResultSet совпадают с полями обработанного SQL запроса. ResultSet может быть:

- нескроллируемый,
- скроллируемый
- неизменяемый
- изменяемый.

Для доступа к полям записи используется или имя поля или его порядковый номер. Функции доступа различаются по типу запрашиваемых данных:

```
rs.getInt(“id”)
rs.getString(2)
```

4. Закрытие объектов (ResultSet, Statement, Соединение)

Для эффективного использования повторяющихся SQL запросов в JDBC API используется специальный механизм PreparedStatement. Он позволяет хранить в памяти прекомпилированный образ SQL выражения. Это, естественно, экономит время на компиляцию SQL выражений. Например:

```
PreparedStatement pst = con.preparedStatement(
        "UPDATE Order SET amount = ? WHERE id = ?"
        );
pst.setInt(1, 100);
pst.setString(2, “abc”);
pst.executeUpdate();

// повторное использование
pst.setInt(1, 200);
pst.setString(2, “efg”);
boolean result = pst.executeUpdate();
```

Символами “?” в запросе отмечены параметры прекомпилированного SQL выражения.

Для вызовов хранимых процедур и функций используется специальный синтаксис и отдельная конструкция CallbleStatement. Например, вызов функции возвращающей вещественное число и имеющей целочисленный параметр:

```
CallableStatement cst = con.callableStatement("{? = call getMyFunction(?)}");

cst.registerOutParameter(1, java.sql.Types.DOUBLE);
cst.setInt(2, 100);
cst.execute();
```

ADO (ADO.NET)

ADO.NET (ActiveX Data Object.NET) - набор классов, используемый для доступа к источникам данных в платформе .NET. ADO.NET представляет собой новую объектную модель, которая использует стандарт XML для передачи данных.

С точки зрения работы с базами данных – это Object Relational mapping. Производитель предлагает объектный интерфейс для работы данными. XML используется как общий деноминатор, позволяющий работать и с не-реляционными данными по общей схеме.

Основа модели – так называемые отсоединенные наборы данных. Отсоединенный набор данных можно рассматривать как коллекция (DataSet) в базовом языке

программирования, которая содержит моментальный слепок данных. При этом на системные средства возлагается поддержка взаимодействия этого набора с источником данных.

DataSet - независимый от источника данных объект, который не имеет собственных средств для работы с источниками данных. Прикладная программа, таким образом, ничего не знает о соединении с источником данных.

Работа с данными таким образом происходит полностью в памяти приложения. Обновленные данные прозрачно для приложения сохраняются в базе данных. При этом реальное использование соединения с источником данных занимает очень короткое время. Это повышает масштабируемость системы.

Object-relational mapping (O/R mapping) – современные средства

Технология, используемая для представления данных в объектно-ориентированных языках программирования. Иными словами, как объекты и отношения между ними отображаются в таблицах реляционной базы данных.

Типичные примеры:
- сохранить всю иерархию классов в одной таблице
- сохранять каждый конкретный класс в собственной таблице

Естественно, что отображение один класс в собственную таблицу является простейшим и наиболее понятным. Оно упрощается, если атрибуты класса и домены таблицы схожи.

При отображении объектов нам необходимо поддерживать скрытую информацию. Например:
- первичный ключ, который будет идентифициовать строки (экземпляры объектов) в таблице
- признак сохранения данных (что использовать при обновлении: INSERT или DELETE ?)

Отношения между объектами моделируются с помощью внешних ключей

Общие шаблоны:

Таблица наследования (Class Table Inheritance) – каждый класс в цепочке наследования сохраняется в своей таблице

Отображение анешних ключей (Foreign Key Mapping) – отношения между объектами реализуются с помошью внешних ключей

Идентификация объектов (Identity Field) – первичный ключ (object ID) как атрибут базы данных

Поздняя инициализация (Lazy Initialization) – реальное чтение атрибутов большого размера (например, BLOB полей) при первом обращении к ним, а не при начальном чтении объекта.

Позднее чтение (Lazy Read) – чтение объекта только при непосредственном обращении к нему

Подбор типов (Map Similar Types) – отображение атрибутов объекта в походящие по типу домены SQL (например, integer -> numeric)

Подбор столбцов (Map Simple Property to Single Column) – по возможности, отображайте простые атрибуты в один столбец таблицы базы данных

Подбор таблиц (Representing Objects as Tables) – по возможности, следуйте схеме отображения один класс –> одна таблица. Это повышает производительность.

Отдельные таблицы для объектов класса (Separate Tables for Class-Scope Properties) – сохранение атрибутов класса (например, статических переменных) в отдельных таблицах

Скрытая информация (Shadow Information) – классам необходимо сохранять специальные данные (ключи, флаги состояния и т.п.)

Таблица наследования (Single Table Inheritance) – отображайте все классы иерархии в одной таблице

9. Распределенные базы данных

Определения и подходы, относящиеся к распределенным базам данных. Репликация данных. Многоуровневые транзакции

Распределенные базы данных связаны с децентрализованным хранением и обработкой информации. Например, для реляционного отношения возможно хранение строк на разных узлах или хранение доменов на разных узлах сети.

Под распределенной базой (Distributed DataBase - DDB) понимают базу данных, включающую фрагменты данных, располагающихся на различных узлах сети компьютеров. Топология сети при этом может быть различной.

Распределенность отражает способ организации базы данных. С точки зрения внешнего же представления основная задача состоит как раз в том, чтобы эту децентрализацию скрыть и представить базу (для прикладных программ) как обычную локальную базу данных.

К. Дэйт предложил следующие характеристики (свойства) для распределенной базы данных:

- Локальная автономия (local autonomy)
- Независимость узлов (no reliance on central site)
- Непрерывные операции (continuous operation)
- Независимость от расположения (location independence)
- Независимость от фрагментации (fragmentation independence)
- Независимость от тиражирования (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)
- Независимость от сети (network independence)
- Независимость от баз данных (database independence)

Локальная автономия означает, что управление данными на каждом из узлов распределенной системы выполняется локально. Данные узла, с одной стороны, являются фрагментом общей системы, а с другой стороны могут трактоваться как обычная локальная база данных, с которой можно работать независимо от других узлов сети.

Независимость узлов (независимость от центрального узла) означает равноправие всех узлов системы.

Непрерывность операций означает возможность непрерывного доступа к данным вне зависимости от их расположения и операций на локальных узлах.

Независимость от расположения означает независимость данных (операций над ними) от их физического расположения. Независимость здесь и далее может пониматься как выполнение соответствующих операций системными средствами и, соответственно, сокрытие деталей от прикладных программ.

Независимость от фрагментации означает возможность распределенного хранения логически единых данных. Например, для реляционной базы данных – хранение одной таблицы на разных узлах. При этом оперировать с таблицей мы будем как с логически целой.

Тиражирование (или репликация) данных - это процесс переноса изменений объектов базы данных в одном узле в базы, расположенные на других узлах нашей распределенной системы. Репликации могут быть синхронными и асинхронными. Независимость от репликаций означает прозрачное для прикладных программ выполнение этих репликаций.

Обработка распределенных запросов есть возможность выполнения операций языка запросов над распределенной базой данных, сформулированных в рамках обычного запроса на данном языке. Например, для реляционных баз данных мы должны быть способны использовать тот же самый SQL.

Распределенная транзакция – это транзакция, которая охватывает данные на нескольких узлах сети. Соответственно, обработка распределенных транзакций есть возможность выполнения операций обновления распределенной базы данных, не разрушающих целостность и согласованность данных.

Это решается применением двухфазнового или многофазного протокола фиксации транзакций. Обычно используется двухфазный протокол (two-phase commit protocol или 2PC). На первой фазе опрашивается готовность узлов, на второй происходит изменение данных.

Независимость от оборудования и независимость от операционных систем исключают зависимость от деталей реализации узлов.

Независимость от сети исключает зависимость распределенной базы данных от сетевых протоколов.

Независимость от баз данных предполагает наличие различных СУБД в узлах распределенной системы.

Если распределенная база 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 распределенной базы для обеспечения согласованных изменений в нескольких базах данных используют менеджеры распределенных транзакций.

В спецификациях X/Open определен общий протокол XA

Адьтернатива многофазным транзакциям – репликация данных. Репликация связана с необходимостью разрешения конфликтов по обновлению данных. Идея репликции заключается в отказе от физического распределения данных. При репликации любая база данных (как для СУБД, так и для работающих с ней пользователей) всегда является локальной; данные размещаются локально на том узле сети, где они обрабатываются; все транзакции в системе завершаются локально.

Тиражирование данных - это асинхронный перенос изменений объектов исходной базы данных в базы, принадлежащим различным узлам распределенной системы. Функции тиражирования выполняет, как правило, специальный модуль СУБД.

Детали тиражирования данных полностью скрыты от прикладной программы.

10. Многоуровневые модели доступа к данным

Мониторы транзакций, сервера приложений. 3-х уровневая модель и ее отличие от модели клиент-сервер.

В традиционной модели клиент-сервер (2-х звенная модель) есть выделенная компонента (сервер), отвечающая за обработку данных. При этом общение приложения с сервером происходит на языке обработки (SQL для реляционных баз данных).

В трехуровневой модели интерфейс с пользователем полностью независим от компоненты обработки данных. В трехуровневой модели явно присутствуют:

- интерфейс с пользователем
- программное обеспечение промежуточного слоя
- компонента управления данными (управление базами данных попадает сюда же)

Программное обеспечение промежуточного слоя (middleware), выполняет функции управления транзакциями, коммуникациями, транспортировку запросов, управление кэшем и т.п. В частности, middleware ишрает определеяющую роль в управлении распределенными системами.

В случае клиент-сервер систем клиент явным образом запрашивает данные, зная структуру базы данных. Например, клиент передает СУБД SQL-запрос, в ответ получает данные. Имеет место жесткая связь типа "точка- точка".

В случае трехуровневой схемы клиент явно запрашивает один из сервисов (предоставляемых прикладным компонентом), передавая ему некоторое сообщение (например) и получает ответ также в виде сообщения. Для клиента база данных (в том числе и DDB) закрыта слоем сервисов. Клиент может ничего не знать о ее существовании, так как все операции над базой данных выполняются внутри сервисов.

Клиент-сервер системы работают в синхронном режиме, при техуровневой модели мы можем реализовать асинхронное взаимодействие.

SQL, когда он создавался, был задуман как декларативный язык запросов, а не как средство взаимодействия "клиент-сервер". Описания собственно сервисов там нет. Соответственно, трехуровневые системы позволяют преодолеть ограничения модели SQL.

Примерами middleware систем могут быть:

- мониторы транзакций (Tuxedo)
- J2EE серверы (см. EJB. Entity Beans как раз и введены для взаимодействия с источниками данных)

ЛИТЕРАТУРА (Основная)

1. Томас Коннолли, Каролин Бегг, Анна Страчан. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. Издательство: Вильямс, 2000 г. 264 с. ISBN 5-8459-0109-X, 0-201-34287-1

2. Джен Л. Харрингтон. Проектирование реляционных баз данных Издательство: Лори, 2006 г. 230 стр. ISBN 0-12-326425-1, 5-85582-082-3

3. К. Дж. Дейт. Введение в системы баз данных. Издательство: Вильямс, 2005 г. 1328 стр. ISBN 5-8459-0788-8, 0-321-19784-4

4. Гектор Гарсиа-Молина, Джеффри Ульман, Дженнифер Уидом. Системы баз данных. Полный курс.Издательство: Вильямс, 2003 г. 1088 стр. ISBN 5-8459-0384-X

5. Мартин Грабер. SQL. Описание SQL92, SQL99 и SQLJ. Издательство: Лори, 2003 г. 643 стр. ISBN 5-85582-109-9, 0-7821-2538-7

6. Кузнецов С.Д. SQL. Язык реляционных баз данных. Издательство: Майор, 2001 г. 192 стр. ISBN 5-901321-03-0

дополнительная:

1. Джен Харрингтон. Проектирование объектно-ориентированных баз данных. Издательство: ДМК пресс, 2001 г. 272 стр. ISBN 5-94074-097-9, 0-12-326428-6

2. К. Дж. Дейт, Хью Дарвен. Основы будущих систем баз данных: третий манифест. Издательство: "Янус-К". 2004 г. 656 стр. ISBN: 5-8037-0183-1