1. Порядок обработки запроса СУБД.



Исходный текст запроса, переданный по сети из клиентского приложения, сначала подвергается проверке на правильность всех синтаксических конструкций и наличие таблиц и столбцов с именами, заданными в тексте запроса. Для запроса, который признан синтаксически правильным, затем формируется *план его исполнения,* представляющий собой описание (во внутреннем формате СУБД) наиболее оптимального способа реализации тех реляционных операций, которые содержатся в тексте запроса. Все эти действия выполняет специальный компонент СУБД, который называется *оптимизатором запроса* **(**Query Optimizer**).**

Оптимизатор запроса передает план исполнения запроса другому компоненту СУБД, который называется *процессором SQL* (SQL Engine). Процессор SQL исполняет все необходимые операции в соответствии с планом. В результате для запроса select формируется *виртуальная таблица* с выходными данными, которая возвращается клиенту в ответ на его запрос. Для запросов insert, delete, update выполняются необходимые действия с таблицами, которые в них указаны – клиент при этом получает сообщение об успешном или неуспешном выполнении операции. Для результата запроса select термин «виртуальная таблица» вполне уместен, поскольку клиентское приложение действительно получает результат в виде таблицы, но на сервере при этом никакой реальной таблицы не создаётся.

Если запрос, поступивший на сервер, уже исполнялся ранее, и для него имеется готовый план исполнения, этап построения плана опускается, и управление сразу передаётся процессору SQL, что позволяет сократить общее время обработки запроса.

1. Представления и их роль в базе данных

*Представления* (другие варианты перевода – просмотры, виды) - это именованные запросы на выборку, сохранённые в базе данных, которые при любом обращении к ним по имени выполняются, создавая виртуальную таблицу, наполненную актуальными данными. Для каждого представления хранится исходный текст запроса, лежащего в его основе, и план исполнения.

Использование представлений имеет глубокий смысл, который формулируется в правиле №6 Кодда для реляционных баз данных:

«База данных должна быть доступна конечным пользователям только через представления. Все представления, которые теоретически можно обновить, должны быть доступны для обновления».

Иными словами, механизм представлений позволяет предоставлять каждому пользователю только ту часть базы данных, которая ему действительно требуется для работы, скрывая от него концептуальную схему, доступную только администратору базы данных.

Для разработчика использование представлений упрощает разработку сложных SQL-запросов, которые можно строить на основе представлений и отлаживать по частям.

1. Плюсы и минусы хранимого кода

Хранимые подпрограммы обеспечивают приложениям баз данных следующие преимущества:

* сокращение объема программирования при разработке приложений, так как однажды созданная подпрограмма может использоваться разными приложениями;
* уменьшение сетевого трафика, так как, если подпрограмма включает в себя несколько обращений к базе данных, по сети передается не каждый запрос и его результат, а только вызов процедуры и ее конечный результат;
* повышение производительности, так как на сервере есть больше возможностей оптимизации локально выполняющихся запросов, и здесь могут быть применены средства, недоступные для клиентской части;
* гарантия того, что задача, решаемая хранимой процедурой, будет одинаково выполняться для всех клиентских приложений и для всех клиентских платформ при любых настройках клиентов.

Основным недостатком хранимого кода является его непереносимость между различными СУБД.

Следует отметить, что хранимые процедуры и функции не следует рассматривать как альтернативу сложным SQL-запросам. По сравнению с «чистым» SQL любая процедурная альтернатива является худшим вариантом с точки зрения производительности, поскольку SQL-запросы многие СУБД обрабатывает с наивысшей эффективностью

1. Основные конструкции языка PL/SQL

Основной программной единицей в языке PL/SQL является блок – совокупность операторов, заключенная в операторные скобки BEGIN ... END.

При выполнении процедурных действий в блоке, как правило, необходимы переменные для хранения промежуточных значений. Объявление переменных предшествует блоку и образует секцию объявления, которая начинается ключевым словом DECLARE. Есть возможность создавать пользовательские типы.

В PL/SQL имеется полезная возможность указать тип переменной, явно ссылаясь на определенный столбец какой-либо таблицы или таблицу целиком. Например:

DECLARE

x INTEGER;

fio students.name\_st%type;

sub subjects%ROWtype;

BEGIN

. . .

[EXCEPTION ……………..]

END;

В приведенном примере для переменной x тип указан явно, переменная fio имеет такой же тип, как столбец name\_st таблицы students. Переменная sub имеет тип запись (RECORD), структура которой идентична строке таблицы subjects (содержит два поля cod\_sub и name\_sub, к которым можно обратиться с помощью составного имени, например, sub.name\_sub).

**1. Оператор присваивания**

переменная:= выражение;

или

SELECT выражение INTO переменная FROM dual;

**2. Условный оператор**

IF условие THEN операторы

[ELSIF условие THEN операторы]

[ELSE операторы]

END IF;

**3. Операторы цикла**

Как и в других языках высокого уровня, в PL/SQL имеется несколько операторов цикла, так что у разработчиков хранимого кода есть выбор.

**Бесконечный цикл, условие выхода задается в теле цикла**

LOOP

операторы

[EXIT WHEN условие выхода из цикла;]

……..

END LOOP;

**Цикл с предусловием**

WHILE условие LOOP

операторы

END LOOP;

**Цикл с параметром**

FOR параметр IN множество\_значений LOOP

операторы

END LOOP;

**5. Оператор безусловного перехода:**

GOTO метка;

...

метка: оператор;

**6. Оператор возврата из процедуры/функции**

RETURN;

RETURN выражение; (только функции)

**7. Комментарии**

Однострочный комментарий обозначается так:

-- далее следует текст комментария до конца строки

Многострочный комментарий обозначается аналогично языку С:

/\* текст комментария может располагаться где угодно и занимать сколько угодно строк \*/

**8. Средства обработки исключений**

Общий подход к обработке исключительных ситуаций состоит в том, что для каждой ситуации определяется ее обработчик, и при возникновении ситуации выполняется код соответствующего обработчика.

DECLARE

err EXCEPTION;

BEGIN

INSERT INTO … VALUES … ;

SELECT ... INTO …;

IF ... THEN RAISE err;

...

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN ...

WHEN TOO\_MANY\_ROWS THEN ...

WHEN err THEN ...

WHEN OTHERS THEN ...

END;

Исключения, возникающих в процессе выполнения команд INSERT (нарушено ограничение уникальности) и SELECT …INTO (запрос, который должен вернуть одну строку, вернул больше одной строки).

1. Хранимые процедуры и функции

Хранимые процедуры и функции являются стандартными объектами базы данных. Их понимание в SQL не отличается от общепринятого: хранимой подпрограммой (процедурой или функцией) называется именованная, отдельно описанная, повторно используемая программная единица, выполняющая, как правило, определенную прикладную функцию.

Хранимая процедура создается оператором SQL

CREATE [OR REPLACE] PROCEDURE имя[(список\_параметров)]

AS

блок PL/SQL

Необязательная конструкция OR REPLACE позволяет заменять процедуру с таким же именем. Это очень удобно в процессе отладки.

Аналогично создается хранимая функция:

CREATE [OR REPLACE] FUNCTION имя[(список\_параметров)]

RETURN тип\_результата, возвращаемого функцией

AS

блок PL/SQL, обязательно содержащий оператор

RETURN выражение

В списке параметров должен быть описан режим использования каждого параметра: IN (только входной – используется по умолчанию), OUT (только выходной), IN OUT (параметр, который изменяет своё значение при выполнении подпрограммы). Режим использования указывается после имени параметра. Типы параметров, как и типы переменных, можно указывать явно или с помощью ссылки на соответствующий столбец или таблицу.

При описании локальных переменных подпрограммы разрешено опускать ключевое слово DECLARE.

Удалить хранимую процедуру или функцию можно при помощи стандартной команды DROP.

1. Курсоры. Явные и неявные курсоры. Команды SQL для работы с курсорами

Курсор представляет собой результат выборки из базы данных, который предназначен для дальнейшей построчной обработки.

Различают неявный и явный курсоры. **Неявный курсор** можно использовать только в том случае, если *запрос на выборку возвращает ровно одну строку*. Тогда этот результат можно поместить в обычные переменные, используя расширенный синтаксис команды SELECT:

SELECT список\_выражений INTO список\_переменных ...

остальная часть оператора SELECT

Количество переменных в списке и их типы должны в точности соответствовать списку выражений оператора SELECT.

При выполнении команды SELECT ... INTO … в различных случаях её применения могут возникнуть две разные исключительные ситуации:

* TOO\_MANY\_ROWS возникает в том случае, если запрос SELECT вместо одной строки вернул несколько строк (в этом случае возвращаемые данные невозможно разместить в заданном списке переменных)
* NO\_DATA\_FOUND возникает в том случае, если запрос SELECT вообще не вернул данных. Тогда переменные в списке не могут получить никаких значений.

При наличии обработчиков для каждой из указанных ситуаций применение неявного курсора является простым, эффективным и безопасным способом обработки результатов однострочной выборки из базы данных.

**Явный курсор** является более универсальным средством обработки произвольной выборки из базы данных. Он должен быть явно объявлен в разделе DECLARE. В объявлении курсора определяется его имя и запрос, на котором основан курсор.

DECLARE CURSOR имя\_курсора IS SELECT ... далее идет запрос на выборку

Например:

DECLARE CURSOR cur IS

SELECT name\_st FROM students WHERE name\_st LIKE ‘A%’;

Следует отметить, что приведённое выше объявление курсора, принятое в Oracle, не совсем соответствует стандарту. Согласно стандарту, объявление курсора выглядит так:

имя\_курсора CURSOR FOR SELECT ….

Все остальные операции с курсором вполне соответствуют стандарту.

Объявление курсора не является выполнимым оператором. Выполнение запроса, выделение памяти под курсор и его наполнение результатами запроса выполняется при открытии курсора. Например:

OPEN CURSOR cur

После открытия курсора можно последовательно выбирать строки курсора, используя оператор FETCH. Например:

FETCH cur INTO fio

Переменная fio должна быть предварительно объявлена, например, так:

fio students.name\_st%type;

Каждое следующее выполнение FETCH выбирает значения столбцов из следующей строки курсора в переменные заданного списка. Оператор FETCH, как правило, применяется в цикле. Например:

LOOP

FETCH cur INTO fio;

...

EXIT WHEN NOT cur%FOUND;

END LOOP;

или

FETCH cur INTO fio;

WHILE cur%FOUND LOOP

FETCH cur INTO fio;

…

END LOOP;

После того как выбраны все нужные строки, курсор должен быть закрыт. Например: CLOSE cur;

**Цикл по курсору**

Некоторые СУБД, в том числе Oracle, поддерживают цикл с параметром по курсору;

FOR параметр IN имя\_курсора LOOP

...

END LOOP;

Использование такого цикла не требует операций открытия и закрытия курсора – они выполняются неявно. Параметр цикла не требуется объявлять в секции DECLARE, его тип определяется автоматически как RECORD, а имена полей записи соответствуют именам столбцов или выражений в объявлении курсора. Например:

FOR cur\_rec IN cur LOOP

... cur\_rec.name\_st…

END LOOP;

Наконец, самый простой способ реализации цикла по курсору вообще не требует явного объявления курсора (такой способ на практике используется часто). Предыдущий пример можно записать так:

FOR cur\_rec IN

(SELECT name\_st FROM students WHERE name\_st LIKE ‘A%’) LOOP

... cur\_rec.name\_st…

END LOOP;

Из этих примеров понятно, что использование цикла по курсору – очень простой и удобный способ обработки курсора.

1. Триггеры. Причины для использования триггеров

Триггеры – особый вид хранимых процедур, которые запускаются автоматически при наступлении определенных событий в базе данных.

1. Особенности триггеров (на примере Oracle)

Являясь по сути хранимой процедурой, триггер обладает теми же преимуществами и недостатками, что и весь хранимый код. К преимуществам следует добавить то, что триггеры являются прекрасным инструментом для администратора БД, поскольку работают независимо от того, какое из клиентских приложений вызвало активизирующее их событие. Эта особенность превращает триггеры также в удобное средство для добавления новой функциональности в существующую систему без всякого изменения ее программного кода. Достаточно только выбрать подходящее событие и создать триггер.

Однако, нужно отметить, что использовать триггеры следует с особой осторожностью, ведь клиентские приложения «ничего не знают» о существовании тех или иных триггеров на сервере, и важно не допустить никаких конфликтов и противоречий в слаженной работе всей информационной системы.

Событий, которые могут активизировать триггеры, довольно много, например, Oracle поддерживает триггеры уровня базы данных, уровня схемы и уровня таблицы. В рамках данного курса рассмотрим только триггеры уровня таблицы, которые обеспечивают автоматическое выполнение некоторых действий при каждой модификации данных таблицы.

Такой триггер характеризуется следующими признаками, которые должны быть заданы при его создании:

* уникальное имя триггера (задание параметров не требуется, поскольку триггер – процедура без параметров);
* активизирующее действие – команда, которая вызывает запуск триггера, такими командами являются INSERT, DELETE, UPDATE;
* время активизации – выполнение триггера до (BEFORE) или после (AFTER) выполнения активизирующего действия;
* область действия – выполнение триггера либо один раз для каждого оператора модификации таблицы, либо для каждой строки (в последнем случае следует добавить фразу FOR EACH ROW);
* условие активизации – необязательное дополнительное условие, которое должно выполняться для запуска триггера (фраза WHEN);
* тело триггера – действия, выполняемые триггером (блок PL/SQL).

На каждое событие может быть создано и несколько триггеров. Однотипные триггеры выполняются в порядке их создания. Можно и в одном триггере указать несколько активизирующих его событий (но только одной таблицы!).

Действие, выполняемое в триггере, может включать в себя операции INSERT, DELETE, UPDATE, которые, в свою очередь, могут запускать выполнение того же или других триггеров. Такое явление называется каскадированием триггера.

Триггер создается при помощи команды SQL:

CREATE [OR REPLACE] TRIGGER имя\_триггера

время\_активизации активизирующая\_команда ON имя\_таблицы

[FOR EACH ROW] – чтобы триггер выполнялся для каждой строки

[WHEN дополнительное условие запуска триггера]

AS

Блок PL/SQL

В теле триггера можно использовать *любые операторы PL/SQL, кроме операторов SQL, которые изменяют таблицу, для которой был создан данный триггер.* Любые другие таблицы изменять можно.

В теле триггера в Oracle можно использовать две предопределенные переменные, которые обозначают ту строку, которая в данный момент подвергается модификации:

:new – новое значение строки, применяется для команд INSERT и UPDATE

:old – старое значение строки (до модификации), применяется для команд DELETE и UPDATE

Если триггер благополучно создан, далее он будет запускаться сам при любом наступлении активизирующего события. Удалить триггер можно при помощи команды

DROP TRIGGER имя\_триггера

Иногда бывают ситуации, когда по каким-либо причинам автоматическое срабатывание триггера не нужно, но и удалять его нельзя, поскольку в дальнейшем он потребуется. Для временного отключения триггера в Oracle можно применить команду:

ALTER TRIGGER имя\_триггера DISABLE

Чтобы снова включить существующий триггер, используют команду:

ALTER TRIGGER имя\_триггера ENABLE

1. Безопасность данных. Пользователи, привилегии и роли.

Подсистема безопасности (Security) предназначена для сохранности данных от злонамеренной порчи либо от несанкционированного доступа. Обычно система безопасности включает две составляющие:

1. Разграничение доступа пользователей.
2. Аудит (контроль) действий пользователей в базе данных.

Для создания учетной записи каждого нового пользователя требуется создать стандартный объект user с обязательным указания пароля данного пользователя (пароли в словаре данных Oracle хранятся в зашифрованном виде, поэтому если пользователь забыл пароль, его нельзя посмотреть, но можно сменить). В Oracle команда CREATE USER … содержит еще ряд дополнительных параметров:

Create user имя\_пользователя

Identified by пароль

[default tablespace имя\_табличного\_пространства]

[temporary tablespace имя\_временного\_табл.\_пространства]

[quota unlimited|размер\_дискового\_пространства]

[profile имя\_профиля]

Несколько слов о профилях пользователей (подробные сведения выходят за рамки курса). Создание индивидуальных профилей для пользователей базы данных позволяет:

1. Вести индивидуальную политику паролей (время жизни пароля, количество попыток входа в систему с неправильным логином или паролем и т.д.)
2. Ограничить ресурсы (количество одновременно открытых сессий, процессорное время, отводимое данному пользователю, или время для некоторого конкретного запроса, время ожидания ответа сервера и ряд других важных ресурсов).

При создании учетной записи в Oracle каждый пользователь одновременно получает в распоряжение свою собственную схему в базе данных с тем же именем, то есть одновременно неявно выполняется команда

Однако, получив в распоряжение собственную учетную запись и собственную схему базы данных, только что созданный пользователь не может даже подключиться к базе данных, не говоря уж о создании объектов. Для выполнения любого действия в базе данных пользователю требуются права на выполнение подобных действий. Для этих целей вводятся два понятия – привилегии и роли.

Привилегии позволяют использовать определенные команды языка SQL по отношению к определенным ее объектам и предоставляются с помощью команды grant… Например, если пользователь user1 владеет таблицей students и выполняет команду

grant select on students to PUBLIC

все пользователи (PUBLIC) смогут выполнять команду SELECT по отношению к таблице students, используя для этого составное имя user1.students или короткий синоним, если он будет создан при помощи команды:

Упрощают работу с привилегиями роли – именованные группы привилегий, которые можно предоставлять пользователем с помощью той же команды GRANT, что и отдельные привилегии. В приложениях с большим числом пользователей применение ролей намного уменьшает количество команд grant. Можно создать заранее определенный набор ролей, с помощью которых пользователям предоставляются только те привилегии, которые им необходимы.

Роль – это стандартный объект базы данных, поэтому она создается при помощи стандартной команды CREATE…

CREATE ROLE имя\_роли

Точно так же, как и только что созданный пользователь, новая роль ещё не содержит никаких привилегий, поэтому присваивать эту роль пользователям бессмысленно.

Наполнить роль набором конкретных привилегий в Oracle можно при помощи уже упоминавшейся команды GRANT. Теперь уже можно представить эту команду в общем виде:

GRANT список\_привилегий/ролей TO список\_пользователей/ролей/PUBLIC

Отменить привилегию или роль, которые присвоены командой GRANT, можно с помощью другой команды:

Revoke список\_привилегий/ролей FROM список\_пользоват./ролей/PUBLIC

Как видим, команды GRANT и REVOKE являются универсальными, а сама система присвоения и отмены привилегий – очень гибкой. Новой роли можно присвоить как конкретные привилегии, так и другие роли. Например, команда:

GRANT connect, resource TO student

присвоит роли student две стандартные роли, которые имеются во всех версиях Oracle: роль connect позволяет подключаться к базе данных, а роль resource позволяет создавать некоторые объекты в своей схеме (таблицы, индексы, представления, хранимый код). Если в целях успешного обучения пользователя с ролью student ему необходимо предоставить все права администратора, это можно сделать при помощи команды:

1. Системные и объектные привилегии. Понятие «владельца » объекта. Аудит действий пользователей в базе данных

Системные привилегии получают администраторы БД и, частично, разработчики приложений. Если разработчикам приложений даётся в распоряжение собственная тестовая база данных, они получают на нее любые системные привилегии, которые им необходимы в процессе разработки и тестирования приложений.

Системные привилегии – это права на выполнение команд CREATE, ALTER, DROP применительно к различным объектам базы данных в своей (или любой) схеме. Если пользователи или роли наделяются привилегией на выполнение DDL в любой схеме, к имени объекта добавляется ключевое слово ANY. Например, команда:

GRANT Create table TO student

позволит включить в роль студент право создавать таблицы в своей схеме, а

GRANT Create any table TO student with grant option

право создавать таблицы в любой схеме, при этом роль student сможет передать эту привилегию другим пользователям или ролям.

Объектные привилегии позволяют пользователям выполнять конкретные действия на конкретном объекте. Такова, например, привилегия удалять строки в указанной таблице. Объектные привилегии назначаются конечным пользователям, так что они могут использовать приложения базы данных для выполнения конкретных задач.

Объектные привилегии выдаются при помощи уже известной универсальной команды GRANT…

GRANT список\_объектных\_привилегий

ON имя таблицы/представления/процедуры

TO список\_ролей/список\_пользователей/PUBLIC

[WITH GRANT OPTION]

В ORACLE существуют следующие привилегии объектного уровня:

* SELECT позволяет другому пользователю выполнить запрос на выборку к данным указанной таблицы или представления.
* INSERT позволяет вставлять строки в таблицу (возможно, используя для этих целей обновляемое представление) с помощью команды INSERT.
* UPDATE позволяет обновлять строки в таблице (обновляемом представлении) вне зависимости от того, были ли эти строки созданы этим пользователем или нет.
* DELETE позволяет удалять из таблицы любые существующие строки. С использованием представления можно ограничить то, какие строки будут удалены.
* EXECUTE даёт возможность пользователю, владеющему хранимым кодом базы данных (процедурами, функциями или пакетами), позволить другому пользователю ORACLE вызывать его процедурные объекты.
* ALTER даёт возможность пользователю ORACLE изменить определение заданной таблицы или последовательности. Не следует путать с системной привилегией ALTER TABLE, которая дает возможность изменять структуру любой таблицы в своей схеме.
* INDEX позволяет пользователю создавать индексы на указанную таблицу, владельцем которой он не является. Владелец эту привилегию имеет по умолчанию.

В конце оператора grant привилегии объектного уровня может быть определена фраза with grant option, которая позволяет пользователю, получившему эту привилегию, передать её другому пользователю ORACLE.

Разграничение прав доступа пользователей вовсе не отменяет необходимость контроля (аудита) их действий в базе данных.

Разрешается проводить аудит операций трех разных типов:

* попыток регистрации в базе данных (аудит подключений)
* обращения к определенным объектам (аудит объектов)
* определенных операций с базой данных (аудит действий).

Во время аудита база данных по умолчанию должна регистрировать как успешные, так и неуспешные команды. Этот режим можно изменить при установке аудита любого типа.

**Аудит подключений**

Можно осуществлять аудит всех попыток соединения с базой данных. Аудит попыток регистрации задается командой:

audit session

**Аудит операций**

Любая команда DDL, оказывающая воздействие на некоторый объект базы данных, например на таблицу, представление или индекс, может быть подвергнута аудиту. При этом нетрудно сгруппировать операции create, alter и drop, воздействующие на объекты. Группирование команд снижает объем административной работы, необходимой для установки и поддержки параметров аудита.

Так, для аудита всех команд, воздействующих на роли, нужно ввести команду:

audit role

Чтобы отменить заданную установку, следует ввести:

noaudit role

**Аудит объектов**

Помимо системных операций, выполняемых над объектами, аудиту можно подвергать операции select**,** insert**,** update и delete, выполняемые над конкретными таблицами.

Конструкция, добавляемая для аудита объектов – это by session (на сеанс) или by access (по доступу). Она определяет, нужно ли вносить запись аудита однажды для каждого сеанса или каждый раз при обращении к объекту. Например, если пользователь выполнил над одной и той же таблицей четыре различных оператора update, результатом аудита по доступу будет внесение четырех записей аудита – по одной на каждое обращение к таблице. Однако, если в той же самой ситуации применить конструкцию by session, то будет внесена только одна запись аудита.

Поэтому аудит по доступу может намного увеличить частоту внесения записей аудита. Он используется достаточно редко и, как правило, для измерения числа отдельных операций, выполняемых в течение определенного временного интервала; после завершения тестирования следует установить для аудита состояние by session.

Ниже рассмотрены примеры использования рассмотренных способов аудита. В первой команде производится аудит всех команд insert, выполняемых над таблицей students, находящейся в схеме user1. Во второй команде аудиту подвергается каждая команда, воздействующая на таблицу marks. В третьей команде осуществляется аудит операций delete, выполняемых над таблицей subjects в течение сеанса:

audit insert on use1.students

1. Понятие транзакции. Свойства транзакций .

*Транзакция –* единица работы СУБД, которая может быть выполнена либо целиком, либо вообще не выполнена. Объем транзакции может варьироваться от одного SQL-оператора до всех действий с базой данных, выполняемых приложением. Для того, чтобы понять суть механизма транзакций, рассмотрим основные свойства, характеризующие транзакцию.

Транзакция характеризуется четырьмя основными свойствами, часто называемыми свойствами АСИД – Атомарность, Согласованность, Изолированность, Долговечность (устойчивость). На английском языке эта аббревиатура также обозначается ACID – Atomicity, Consistency, Isolation, Durability. Поясним каждое из свойств по отдельности.

**Атомарность**

Транзакция является неделимой, она выполняется полностью или не выполняется вообще; если транзакция прерывается на середине, то база данных должна остаться в том состоянии, которое она имела до начала транзакции.

**Согласованность (целостность)**

Транзакция переводит базу данных из одного согласованного (целостного) состояния в другое, также целостное. В ходе выполнения транзакции база данных может временно пребывать в нецелостном состоянии.

**Изоляция**

До сих пор мы не рассматривали особенности работы СУБД в многопользовательской среде. Но в реальных информационных системах на основе сервера Oracle могут одновременно работать десятки тысяч пользователей, каждый из которых выполняет свои собственные транзакции. Свойство изоляции обозначает, что *все транзакции выполняются изолированно друг от друга.*

**Долговременность**

После того как транзакция завершена и зафиксирована, результат ее выполнения гарантированно сохраняется в базе данных. При любых аварийных ситуациях, используя возможности сервера, можно восстановить *все зафиксированные транзакции.* Восстановление незафиксированных транзакций сервер, разумеется, не гарантирует.

1. Команды SQL для работы с транзакциями. Средства СУБД для реализации механизма транзакций

Для обеспечения всех перечисленных выше свойств транзакции необходима как языковая поддержка процессов инициализации и завершения транзакции, так и механизмы их реализации сервером. Рассмотрим данные вопросы по порядку.

Языковые правила поддержки транзакций для различных СУБД несколько различаются. Например, в Microsoft SQL Server поддерживается команда начала транзакции BEGIN TRANS[ACTION]. В Oracle такой команды нет, но существуют четкие правила, регламентирующие моменты начала и завершения транзакции:

1). Любая команда ddl выполняется как отдельная транзакция. Иными словами, поступление на сервер команды DDL автоматически фиксирует результаты предыдущих команд DML этого сеанса (если таковые были) и начинает новую транзакцию, а при завершении команды DDL автоматически фиксируются ее результаты. Таким образом, одна команда DDL вызывает те же действия, что и последовательность команд:

COMMIT

команда DDL

COMMIT

Получается, что несколько команд DDL нельзя объединить в единую транзакцию и откатить команду DDL при помощи стандартной команды ROLLBACK в Oracle также нельзя.

2). Результаты выполнения команд dml автоматически фиксируются только при включенном режиме AUTOCOMMIT (например, в настройках утилиты SQL\*Plus есть возможность включения этого режима). По умолчанию этот режим отключен. Таким образом, все идущие подряд команды DML воспринимаются как одна транзакция.

Инициализация транзакции (неявная команда BEGIN TRANSACTION) происходит в следующих случаях:

* первая команда в сеансе связи
* первая команда после команд COMMIT или ROLLBACK
* первая команда после команды DDL

Завершение транзакции происходит при поступлении команд commit (завершение транзакции с фиксацией изменений) или rollback (завершение транзакции с откатом изменений). Можно неявно зафиксировать команду транзакцию из последовательности команд dml любой следующей за ней командой ddl.

Стандарт SQL и многие СУБД, в том числе Oracle, предусматривают так называемые точки сохранения. Точка сохранения задается оператором

SAVEPOINT имя\_точки\_сохранения

и в операторе ROLLBACK имеется возможность отката транзакции не к началу, а к указанной точке сохранения:

ROLLBACK TO имя\_точки\_сохранения

Данная команда выполняет откат только тех изменений, которые были сделаны после точки сохранения, и *не завершает транзакцию*.

Поддержка транзакций требует значительных ресурсов и существенно (во много раз!) замедляет производительность сервера. Однако, в современных условиях допустить потерю или порчу информации в базе данных абсолютно недопустимо, поэтому правила АСИД реализуются всеми СУБД. Несмотря на особенности конкретных реализаций, имеется ряд универсальных механизмов поддержки транзакций.

**Журнализация транзакций**

Ведение журналов транзакций преследует одновременно две цели:

1). Возможность отката транзакции.

2). Восстановление БД в случае аварийных ситуаций или сбоев.

Сервер ведёт 2 вида журналов транзакций:

**Undo-журналы** используются для отката и ведутся для каждой транзакции отдельно. Как только очередная транзакция зафиксирована или откачена, то информация из соответствующего Undo-журнала удаляется.

**Redo-журнал** необходим для повторного выполнения транзакций при восстановлении данных. Это единый системный журнал, в который записываются результаты всех зафиксированных транзакций.

**Сериальный график исполнения транзакций. Монитор транзакций**

Для обеспечения свойства изолированности транзакций при максимально возможной в этих условиях производительности сервера используется механизм сериализации транзакций. Сериальный график исполнения транзакций обеспечивает параллельное исполнение сервером нескольких транзакций (то есть, команды различных транзакций могут исполняться на сервере «вперемешку»), но результат работы гарантированно должен быть точно таким же, как если бы эти транзакции исполнялись последовательно.

**Механизмы блокировки**

Большинство СУБД позволяют любому числу транзакций одновременно осуществлять доступ к одной и той же базе данных, и в них существуют механизмы управления параллельными процессами, предотвращающие нежелательные воздействия одних транзакций на другие. По сути, это механизм блокирования, главная идея которого достаточно проста. Если транзакции нужны гарантии, что некоторый объект (база данных, таблица или строка), в котором она заинтересована, не будет изменен каким-либо непредсказуемым образом в течение требуемого промежутка времени, она устанавливает блокировку этого объекта. Результат блокировки заключается в том, чтобы изолировать этот объект от других транзакций и предотвратить его изменение средствами этих транзакций.

1. Пути настройки производительности СУБД
2. План запроса. Стоимость запроса. Сбор статистики для оптимизатора запросов.
3. Подсказки для оптимизатора запросов.
4. Резервирование данных. Способы резервирования. Восстановление данных.