Задание

Изучить теоретический материал, законспектировать все что выделено синим цветом. Уметь читать синтаксис инструкций SQL в примерах к лекции.

С уважением, Хвастова Светлана Ивановна

!!! Если возникнут вопросы обращаться по телефону 0721389311. Электронная почта: xvsviv@rambler.ru

Лекция на тему: «Подзапросы и выражения с запросами. Внесение изменений в базу данных. Условия целостности данных. Обязательность данных. Условия на значения. Целостность таблицы. Ссылочная целостность**.»**

План

1. Что такое подзапросы в SQL?
2. Подзапросы SQL-примеры
3. Графическое представление подзапроса SQL
4. Подзапросы в SQL (вложенные запросы SQL): общие правила
5. Правила целостности данных
6. Обработка данных в многопользовательской СУБД
7. Атомарность SQL-выражений при работе с данными
8. Распараллеливание операций
9. Обеспечение максимальной производительности
10. Транзакции
11. Блокировки

Цель: изучить инструкции создания подзапросов, инструкции внесения изменений. Выучить условия целостности данных.

1. Что такое подзапросы в SQL?

- **SQL подзапрос** – это запрос, вложенный в другой запрос;
- **Подзапрос** может использоваться:

– В инструкции **SELECT**;

– В инструкции **FROM**;

– В условии **WHERE**.

- Подзапрос может быть вложен в инструкции **SELECT**, **INSERT**, **UPDATE** или **DELETE**, а также в другой подзапрос;
- Подзапрос обычно добавляется в условие **WHERE** оператора **SQL SELECT**;
- Можно использовать операторы сравнения, такие как >, <, или =. IN, **ANY** или **ALL**;
- Подзапрос также называется внутренним запросом. Оператор, содержащий подзапрос, также называется внешним;
- Внутренний запрос выполняется перед родительским запросом, чтобы результаты его работы могли быть переданы внешнему.

Подзапрос можно использовать в инструкциях **SELECT**, **INSERT**, **DELETE** или **UPDATE** для выполнения следующих задач:

- Сравнения выражения с результатом запроса;
- Определения того, включено ли выражение в результаты запроса;
- Проверки того, выбирает ли запрос любые строки.

Синтаксис:

```
SELECT    select_list
FROM      table
WHERE     expr operator
                (SELECT    select_list
                FROM       table);
```

- Подзапрос **SQL** (*внутренний запрос*) выполняется перед выполнением основного запроса (*внешнего запроса*);
- Основной запрос использует результат выполнения подзапроса.

2. Подзапросы SQL-примеры

В этом разделе мы рассмотрим, как использовать подзапросы. У нас есть следующие две таблицы: '***student***' и '***marks***' с общим полем '***StudentID***':

StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos

студенты

Запросы к базе данных (команда select)

StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

отметки

Теперь нужно составить запрос, определяющий всех студентов, которые получают лучшие отметки, чем студент со **StudentID** - «***V002***». Но мы не знаем отметок студента «***V002***».

Поэтому нужно составить два **SQL** подзапроса в **Select**. Один запрос возвращает отметки (*хранятся в поле «**Total_marks**»*) для «***V002***», а второй запрос выбирает учеников, которые получают лучшие оценки, чем результат первого запроса.

Первый запрос:

```
SELECT *
FROM `marks`
WHERE studentid = 'V002';
```

Результат запроса:

StudentID	Total_marks
V002	80

Результатом запроса будет **80**.

Используя результат этого запроса, мы написали еще один запрос, чтобы определить учеников, которые получают оценки лучше, чем **80**.

Второй запрос:

```
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks >80;
```

Результат запроса:

studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

Два приведенных запроса определяют студентов, которые получают лучше оценки, чем студент **StudentID** «***V002***» (**Abhay**).

Можно объединить эти два запроса, вложив один запрос в другой. Подзапрос - это запрос внутри круглых скобок. Рассмотрим **подзапроса в SQL пример**:

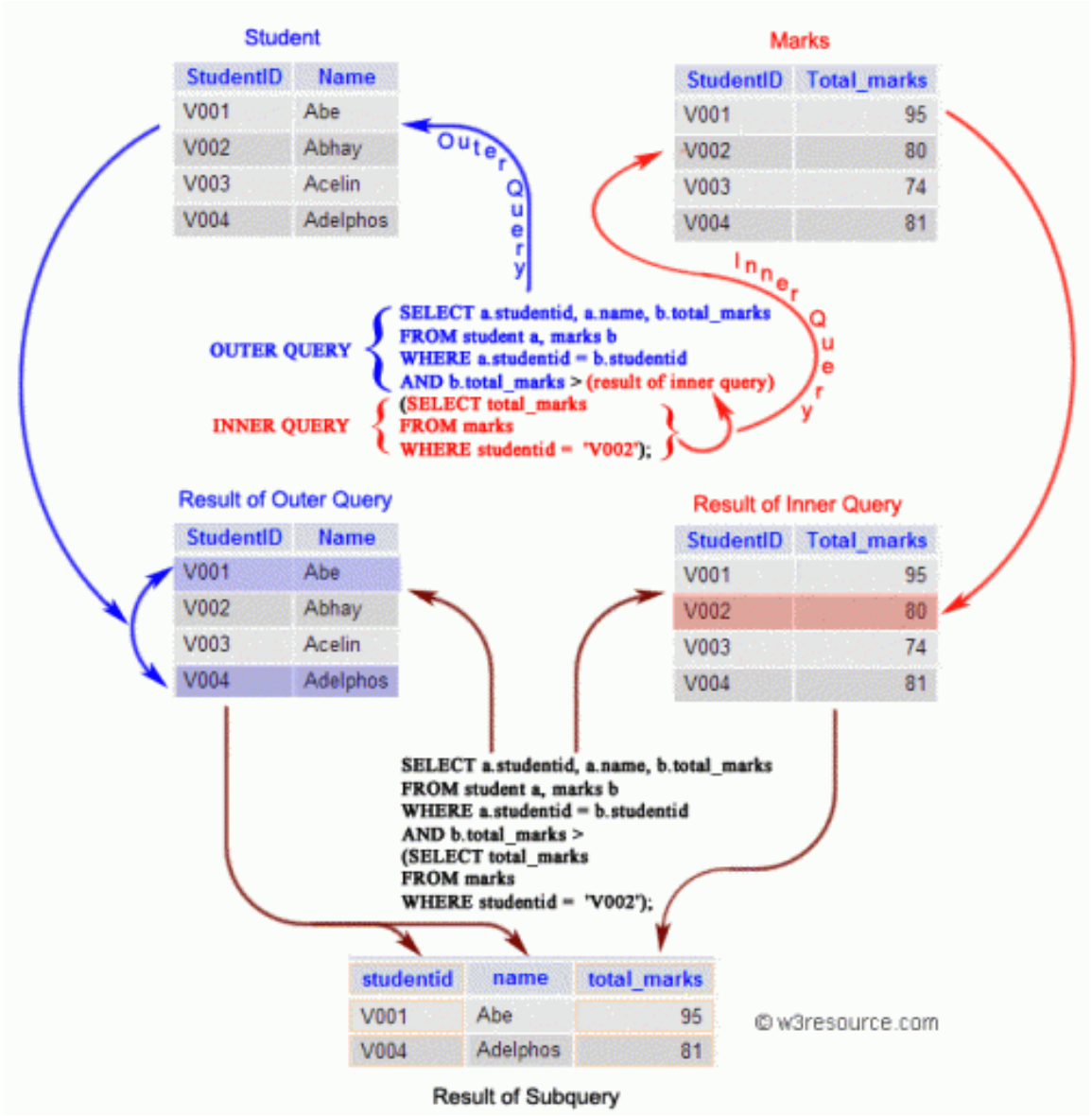
Код SQL:

```
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid AND b.total_marks >
(SELECT total_marks
FROM marks
WHERE studentid =  'V002');
```

Результат запроса:

studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

3. Графическое представление подзапроса SQL:



4. Подзапросы в SQL (вложенные запросы SQL): общие правила

Ниже приведен синтаксис подзапроса:

Синтаксис:

```
(SELECT [DISTINCT] аргументы_подзапроса_для_отбора
FROM {имя_таблицы | имя_представления}
{ имя_таблицы | имя_представления } ...
[WHERE условия_поиска]
[GROUP BY выражение_объединения [,выражение_объединения] ...]
[HAVING условия_поиска])
```

Подзапросы SQL (вложенные запросы SQL): рекомендации по использованию

Ниже приведен ряд рекомендаций, которым нужно следовать при использовании **SQL** подзапросов:

- Подзапрос должен быть заключен в круглые скобки;
- Подзапрос должен указываться в правой части оператора сравнения;
- Подзапросы не могут обрабатывать свои результаты, поэтому в подзапрос не может быть добавлено условие **ORDER BY**;
- Используйте однострочные операторы с однострочными подзапросами;
- Если подзапрос возвращает во внешний запрос значение **null**, внешний запрос не будет возвращать никакие строки при использовании операторов сравнения в условии **WHERE**.

Подзапросы SQL (вложенные запросы SQL) - основные типы

- **Однострочный подзапрос**: возвращает ноль или одну строку;
- **Многострочный подзапрос**: возвращает одну или несколько строк;
- **Многостолбцовый подзапрос**: возвращает один или несколько столбцов;
- **Коррелированные подзапросы**: указывают один или несколько столбцов во внешней инструкции **SQL**. Такой подзапрос называется коррелированным, поскольку он связан с внешней инструкцией **SQL**;
- **Вложенные подзапросы**: подзапросы помещенные в другой подзапрос.

Также можно использовать подзапрос внутри инструкций **INSERT**, **UPDATE** и **DELETE**.

Подзапросы SQL с инструкцией INSERT

Инструкция **INSERT** может использоваться с подзапросами **SQL**.

Синтаксис:

```
INSERT INTO имя_таблицы [ (столбец1 [, столбец2 ]) ]
SELECT [ *|столбец1 [, столбец2 ]
```

```
FROM таблица1 [, таблица2 ]
[ WHERE VALUE OPERATOR ];
```

Если мы хотим вставить заказы из таблицы '**orders**', для которых в таблице «***neworder***» значение **advance_amount** составляет **2000** или **1500**, можно использовать следующий код **SQL**:

Пример таблицы: orders

```
ORD_NUM ORD_AMOUNT ADVANCE_AMOUNT ORD_DATE CUST_CODE AGENT_CODE ORD_DESCRIPTION
-------- ---------- ------------- --------- -------------- --------------- -----------------
200114 3500 2000 15-AUG-08 C00002 A008
200122 2500 400 16-SEP-08 C00003 A004
200118 500 100 20-JUL-08 C00023 A006
200119 4000 700 16-SEP-08 C00007 A010
200121 1500 600 23-SEP-08 C00008 A004
200130 2500 400 30-JUL-08 C00025 A011
200134 4200 1800 25-SEP-08 C00004 A005
200108 4000 600 15-FEB-08 C00008 A004
200103 1500 700 15-MAY-08 C00021 A005
200105 2500 500 18-JUL-08 C00025 A011
200109 3500 800 30-JUL-08 C00011 A010
200101 3000 1000 15-JUL-08 C00001 A008
200111 1000 300 10-JUL-08 C00020 A008
200104 1500 500 13-MAR-08 C00006 A004
200106 2500 700 20-APR-08 C00005 A002
200125 2000 600 10-OCT-08 C00018 A005
200117 800 200 20-OCT-08 C00014 A001
200123 500 100 16-SEP-08 C00022 A002
200120 500 100 20-JUL-08 C00009 A002
200116 500 100 13-JUL-08 C00010 A009
200124 500 100 20-JUN-08 C00017 A007
200126 500 100 24-JUN-08 C00022 A002
200129 2500 500 20-JUL-08 C00024 A006
200127 2500 400 20-JUL-08 C00015 A003
200128 3500 1500 20-JUL-08 C00009 A002
200135 2000 800 16-SEP-08 C00007 A010
200131 900 150 26-AUG-08 C00012 A012
200133 1200 400 29-JUN-08 C00009 A002
200100 1000 600 08-JAN-08 C00015 A003
200110 3000 500 15-APR-08 C00019 A010
200107 4500 900 30-AUG-08 C00007 A010
200112 2000 400 30-MAY-08 C00016 A007
200113 4000 600 10-JUN-08 C00022 A002
```

200102 2000 300 25-MAY-08 C00012 A012

Код SQL:

```
INSERT INTO neworder
SELECT * FROM  orders
WHERE advance_amount in(2000,1500);
```

Результат:

```
2 row(s) inserted.

0.71 seconds
```

Подзапросы SQL с инструкцией UPDATE

В инструкции **UPDATE** можно установить новое значение столбца, равное результату, возвращаемому однострочным подзапросом. Ниже приводится синтаксис и пример **UPDATE** с подзапросом **SQL**.

Синтаксис:

```
UPDATE таблица  SET имя_столбца = новое_значение
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

Если мы хотим изменить параметры **ord_date** в таблице '**neworder**' с '**15-JAN-10**', для которых разница между **ord_amount** и **advance_amount** меньше минимальной **ord_amount** в таблице '**orders**',то можно использовать следующий **код SQL**:

Пример таблицы: neworder

ORD_*NUM* *ORD*_AMOUNT ADVANCE_*AMOUNT* *ORD*_DATE CUST_*CODE AGENT*_CODE ORD_DESCRIPTION

-------- ---------- ------------- --------- -------------- -------------- -----------------

200114 3500 2000 15-AUG-08 C00002 A008

200122 2500 400 16-SEP-08 C00003 A004

200118 500 100 20-JUL-08 C00023 A006

200119 4000 700 16-SEP-08 C00007 A010
200121 1500 600 23-SEP-08 C00008 A004
200130 2500 400 30-JUL-08 C00025 A011
200134 4200 1800 25-SEP-08 C00004 A005
200108 4000 600 15-FEB-08 C00008 A004
200103 1500 700 15-MAY-08 C00021 A005
200105 2500 500 18-JUL-08 C00025 A011
200109 3500 800 30-JUL-08 C00011 A010
200101 3000 1000 15-JUL-08 C00001 A008
200111 1000 300 10-JUL-08 C00020 A008
200104 1500 500 13-MAR-08 C00006 A004
200106 2500 700 20-APR-08 C00005 A002
200125 2000 600 10-OCT-08 C00018 A005
200117 800 200 20-OCT-08 C00014 A001
200123 500 100 16-SEP-08 C00022 A002
200120 500 100 20-JUL-08 C00009 A002
200116 500 100 13-JUL-08 C00010 A009
200124 500 100 20-JUN-08 C00017 A007
200126 500 100 24-JUN-08 C00022 A002
200129 2500 500 20-JUL-08 C00024 A006
200127 2500 400 20-JUL-08 C00015 A003
200128 3500 1500 20-JUL-08 C00009 A002
200135 2000 800 16-SEP-08 C00007 A010
200131 900 150 26-AUG-08 C00012 A012
200133 1200 400 29-JUN-08 C00009 A002
200100 1000 600 08-JAN-08 C00015 A003
200110 3000 500 15-APR-08 C00019 A010
200107 4500 900 30-AUG-08 C00007 A010
200112 2000 400 30-MAY-08 C00016 A007
200113 4000 600 10-JUN-08 C00022 A002
200102 2000 300 25-MAY-08 C00012 A012

Код SQL:

```
UPDATE neworder
SET ord_date='15-JAN-10'
WHERE ord_amount-advance_amount<
(SELECT MIN(ord_amount) FROM orders);
```

Результат:

```
7 row(s) updated.

0.06 seconds
```

Подзапросы SQL с инструкцией DELETE

Ниже приводится синтаксис и пример использования **SQL** подзапросов с инструкцией **DELETE**.

Синтаксис:

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

Если нужно удалить заказы из таблицы «***neworder***», для которых **advance_amount** меньше максимального значения **advance_amount** из таблицы «***orders***», можно использовать следующий **код SQL**:

Пример таблицы: neworder

```
ORD_NUM ORD_AMOUNT ADVANCE_AMOUNT ORD_DATE CUST_CODE AGENT_CODE ORD_DESCRIPTION
-------- ---------- -------------- --------- --------------- --------------- -----------------
200114 3500 2000 15-AUG-08 C00002 A008
200122 2500 400 16-SEP-08 C00003 A004
200118 500 100 20-JUL-08 C00023 A006
200119 4000 700 16-SEP-08 C00007 A010
200121 1500 600 23-SEP-08 C00008 A004
200130 2500 400 30-JUL-08 C00025 A011
200134 4200 1800 25-SEP-08 C00004 A005
200108 4000 600 15-FEB-08 C00008 A004
200103 1500 700 15-MAY-08 C00021 A005
200105 2500 500 18-JUL-08 C00025 A011
200109 3500 800 30-JUL-08 C00011 A010
200101 3000 1000 15-JUL-08 C00001 A008
200111 1000 300 10-JUL-08 C00020 A008
200104 1500 500 13-MAR-08 C00006 A004
200106 2500 700 20-APR-08 C00005 A002
200125 2000 600 10-OCT-08 C00018 A005
200117 800 200 20-OCT-08 C00014 A001
200123 500 100 16-SEP-08 C00022 A002
200120 500 100 20-JUL-08 C00009 A002
200116 500 100 13-JUL-08 C00010 A009
200124 500 100 20-JUN-08 C00017 A007
200126 500 100 24-JUN-08 C00022 A002
```

200129 2500 500 20-JUL-08 C00024 A006
200127 2500 400 20-JUL-08 C00015 A003
200128 3500 1500 20-JUL-08 C00009 A002
200135 2000 800 16-SEP-08 C00007 A010
200131 900 150 26-AUG-08 C00012 A012
200133 1200 400 29-JUN-08 C00009 A002
200100 1000 600 08-JAN-08 C00015 A003
200110 3000 500 15-APR-08 C00019 A010
200107 4500 900 30-AUG-08 C00007 A010
200112 2000 400 30-MAY-08 C00016 A007
200113 4000 600 10-JUN-08 C00022 A002
200102 2000 300 25-MAY-08 C00012 A012

Код SQL:

```
DELETE FROM neworder
WHERE advance_amount<
(SELECT MAX(advance_amount) FROM orders);
```

Результат:

```
34 row(s) deleted.

0.04 seconds
```

5. Правила целостности данных

Главная особенность SQL-технологий наличие у сервера СУБД специальных средств контроля целостности данных, не зависящих от клиентских программ и привязанных непосредственно к таблицам. Т.е. принципиально не важно, каким образом осуществляется доступ к базе данных: через SQL-консоль, через ODBC-драйвера из приложения Windows, через WWW-connector из Internet-браузера или через DBI-интерфейс Perl. В любом из этих случаев, за контролем целостности данных следит сервер, и при нарушении правил целостности данных сервер известит клиента об ошибке.

К структурам контроля целостности данных относятся ограничители (constraint), которые привязаны к столбцам и триггеры (trigger), которые могут быть привязаны как к столбцам, так и к строкам в таблице.

Ограничители – это элементарные проверки или условия, которые выполняются для операций вставки и модификации значения столбца. Если данная проверка не проходит или условие не выполняется, то вставка или модификация отменяется, а в программу клиента передается ошибка.

SQL-серверы, как правило, поддерживают следующие ограничители.

NOT NULL - проверка на непустое значение. NULL - специальное понятие в СУБД, которое означает "пусто". "Пусто" и "0(ноль)" не равны друг другу!

UNIQUE - проверка на уникальность. Вставляемое значение должно быть уникально для данного столбца по всей таблице. Может содержать пустые значения.

PRIMARY KEY - первичный ключ. Значение в столбце считается первичным ключом, если оно непустое и уникально в пределах столбца данной таблицы. Первичный ключ может быть составным и представлять собой комбинацию столбцов. Тогда чтобы считаться первичным ключом, каждое из группы значений не должно быть пустыми и формируемые строки значений первичного ключа должны быть уникальны в пределах таблицы. Первичный ключ - основа для построения индексов по таблице.

SQL-технология позволяет на уровне столбца задавать домены значений, т.е. строго определенные наборы или диапазоны значений, для помещаемых в столбец данных. В частности можно реализовывать ограничения ссылочной целостности (referential integrity constraint) и проверки фиксированного условия. Ограничение ссылочной целостности не позволяет значениям из столбца одной таблицы принимать значения кроме как из присутствующих в столбце другой таблицы. Это делается при помощи ограничителей FOREIGN KEY (внешний ключ) и REFERENCES (указатель ссылки). Таблица, содержащая FOREIGN KEY, считается родительской таблицей. Таблица, содержащая REFERENCES, считается дочерней таблицей. Внешний ключ и указатель ссылки могут находиться в одной таблице, т.е. родительская таблица одновременно является дочерней.

FOREIGN KEY - внешний ключ. Назначает столбец или комбинацию столбцов в текущей (родительской) таблице в качестве внешнего ключа для ссылки из других таблиц.

REFERENCES - указатель ссылки (или родительский ключ). Указывает на столбец (комбинацию столбцов) в родительской таблице, ограничивающую значения в текущей (дочерней) таблице.

Для использования ограничений ссылочной целостности должны выполняться некоторые условия. В частности, родительская и дочерняя таблицы должны находиться в пределах одного аппаратного сервера базы данных, они не могут находиться на различных узлах распределенной базы данных. Столбцы, участвующие в отношении ограничения ссылочной целостности обязаны иметь один и тот же тип данных.

Ограничения ссылочной целостности используются при каскадном удалении, т.е. при удалении записи в родительской таблице удаляются все записи с указанным ключом из дочерних таблиц, и наоборот при запрете удаления/модификации, т.е. при наличии зависимых записей в дочерних таблицах, значение ключа записи в родительской таблице нельзя удалить или модифицировать.

CHECK - проверка фиксированного условия. В данном ограничителе явно указывается условие, которое должно выполняться для вставляемого или модифицируемого значения в столбце. Например: check (user in 'ALEX','JUSTAS') - в столбце user могут содержаться только значения 'ALEX' и 'JUSTAS', попытка вставки значения 'SHTIRLITZ' будет интерпретирована как ошибочная , check (user_salary between 1000 and 5000) - столбец user_salary может принимать целочисленные значения в диапазоне от 1000 до 5000 и т.д. При формировании условий с некоторыми ограничениями могут использоваться функции, например check (user = upper(user)), в данном случае имя пользователя должно вводиться только в верхнем регистре. Есть и ограничения, например, CHECK не может содержать подзапросы (SELECT).

Обычно ограничители задаются при создании таблиц. Но в дальнейшем их можно изменять, удалять или временно запрещать при помощи соответствующих команд СУБД.

Триггеры – это сохраненная откомпилированная процедура, которая связана с определенной таблицей. Триггеры, в отличие от ограничителей, могут выполнять сколь угодно сложные манипуляции над данными. Помимо операций модификации и вставки, триггеры могут срабатывать и при удалении данных из таблицы. Можно также задавать порядок срабатывания триггера относительно операции, т.е. выполниться ли триггер перед операцией вставки/модификации/удаления значения из столбца (или всей строки) или непосредственно после такой операции.

Некоторые типовые применения триггеров:

- Прозрачный аудит (не зависящий от клиентских программ и невидимый для них) и регистрация событий, связанных с доступом к определенным таблицам или столбцам в таблицах.
- Генерация значений в столбцах на основе значений в других столбцах при вставке/модификации строки данных.
- Манипуляции над зависимыми таблицами в особенности, если они находятся на других узлах распределенной базы данных, чего нельзя сделать при помощи ограничителей.

В случае необходимости триггеры можно запрещать, а затем разрешать. Запрещение триггеров применяется обычно при массовых загрузках данных в таблицы извне, с целью уменьшения времени загрузки. Понятие триггера как выполнение кода по событию в том же Oracle используется весьма широко. В частности, оно является основным при разработке клиентских программ при помощи SQL*Forms. Триггеры пишутся на процедурных расширениях SQL.

6. Обработка данных в многопользовательской СУБД

Основное требование к многопользовательским СУБД - обеспечение непротиворечивости данных в системе, при сохранении максимальной производительности и конкуренции в доступе к данным для пользователей.

Конкуренция в доступе к данным означает, что каждый из пользователей независим от остальных пользователей в потребности обработки данных. Система, во избежание порчи данных, самостоятельно устанавливает очередность работы с данными для пользователей. В случае необходимости пользователи могут ожидать своей очереди для работы с данными. Одной из главной целей многопользовательской СУБД является максимальное уменьшение этого времени ожидания до такой степени, чтобы оно (в идеале) стало незаметным для пользователя.

Кроме того, сервер СУБД должен предотвращать взаимно разрушающие манипуляции с данными нескольких пользователей при их одновременной работе. Например, если система не предусматривает такую возможность, то менеджеры принимающие заказы от клиентов на поставку товара, и выполняющие их резервирование на складе, могут зарезервировать товара больше чем фактически имеется в наличии. В этом случае обеспечен неприятный разговор с клиентом, заказ которого будет впоследствии отменен.

Более неприятная ситуация возможна в банке: если одновременно исполняется несколько клиентских платежных поручений с одного счета, то при неконтролируемом списании с клиентского счета возможен отрицательный остаток, что недопустимо.

Контроль нужен также в системах резервирования билетов на транспорте, чтобы билет на одно и то же место не был продан разными кассирами разным пассажирам.

Несмотря на различия в реализации, серверы СУБД используют общие способы управления данными и доступом к ним.

7. Атомарность SQL-выражений при работе с данными

Под атомарностью выражения понимается неизменность (фиксация во времени) набора данных, с которыми это выражение работает на всем протяжении своего исполнения. Т.е. если мы выполняем оператор UPDATE над определенной таблицей, то состояние таблицы на момент начала выполнения операции фиксируется во времени и не изменяется до конца выполнения оператора. Этот набор данных для текущего выполняемого выражения не может быть изменен другим пользователем или даже другой сессией этого же пользователя, которая пытается выполнить операцию модификации этих же данных в этой же таблице.

8. Распараллеливание операций

Типовые операции с таблицей в базе данных состоят из многих однотипных операций, например оператор UPDATE, который модифицирует 5000 строк в таблице, по своей сути состоит из 5000 операций, каждая из которых может быть выполнена независимо. В связи с этим такие операторы очень хорошо распараллеливаются при использовании многопроцессорных систем. Это позволяет выровнять нагрузку в системе между разными процессорами, при том условии что СУБД умеет работать в многопроцессорной конфигурации, и уменьшить время ответа системы.

9. Обеспечение максимальной производительности

С целью сокращения времени различных пользователей на манипуляции с данными используется ряд следующих методов. Их работа находится на уровне, скрытом даже от программиста СУБД, но о них стоит упомянуть т.к. они иллюстрируют серьезные различия с xBase-технологией.

Строго говоря, эта информация справедлива лишь в отношении Oracle, но другие СУБД используют подобные принципы.

– Процессы, выполняющие чтение блоков данных, никогда не ожидают процессов, выполняющих запись тех же блоков данных.

– Процессы, выполняющие запись блоков данных, при отсутствии явных блокировок со стороны пользователя, не ожидают процессов, выполняющих чтение тех же блоков данных.

– Процессы, выполняющие запись блоков данных, ожидают другие процессы, выполняющие запись, только в случае если они пытаются выполнить запись данных в одни и те же блоки данных.

Данные приемы позволяют существенно уменьшить время ожидания ответа системы и увеличить ее производительность.

10. Транзакции

Многопользовательские системы широко используют понятие транзакций. Транзакция - это логическая единица работы, которая состоит из одного или нескольких SQL-выражений. Группа выражений отмеченных как транзакция рассматривается как единое и неделимое целое. В случае если в одном из выражений обработки данных происходит ошибка, то транзакция отменяется целиком. Таким образом, система возвращается в состояние предшествующее началу транзакции, обеспечивая при этом физическую и логическую непротиворечивость данных.

Например, мы пытаемся модифицировать таблицу при помощи оператора UPDATE. В одном из столбцов этим оператором устанавливается недопустимое значение с точки зрения правил целостности для этой таблицы. Срабатывание ограничителя приведет к тому, что сервер СУБД не позволит выполнить такую модификацию и известит нас ошибкой, а механизм контроля транзакций вызывает отмену всего выполняемого выражения и производит откат к предыдущему состоянию таблицы, сохраняя, таким образом, целостность и непротиворечивость данных. В данном примере транзакция работает с одним SQL-выражением. В случае если выражений несколько, то

откатывается результат работы всех выражений составляющих единую транзакцию.

Чтобы транзакциями можно было пользоваться, в системе должен быть включен режим регистрации транзакций. После этого система сохраняет информацию о предыдущих состояниях объектов в системе в так называемых журналах транзакций. Журналы транзакций - это специальные файлы, управляемые сервером СУБД, в которых записываются все изменения произошедшие с момента начала транзакций для всех транзакций в системе.

Если происходит явное сохранение изменений в системе (по команде COMMIT) или неявное сохранение изменений (по завершению группы SQL-выражений, формирующих транзакцию или по завершению сеанса пользователя), то все изменения произошедшие с момента начала транзакции вносятся в систему, и информация о данной транзакции удаляется из журнала.

Для облегчения управления системой в режиме регистрации транзакций существует возможность задания так называемых промежуточных точек сохранения. Промежуточная точка сохранения по команде SAVEPOINT явно помечает состояние системы и предоставляет возможность восстановления состояния БД на момент ее сохранения по команде ROLLBACK. В данном случае ROLLBACK откатывает систему к указанной точке. Обычно промежуточных точек сохранения для одного пользователя может быть несколько.

В случае если транзакция по каким-то причинам не может быть завершена, то происходит неявный откат. Его причиной, могут быть, например, ошибка при выполнении одного из SQL-выражений, составляющих транзакцию, или обрыв связи с инициатором транзакции. При этом по информации из журнала восстанавливается предыдущее состояние объектов, которые пыталась модифицировать текущая транзакция, после чего информация о транзакции удаляется из журнала. Откат может быть и явным - по команде ROLLBACK.

Данная схема справедлива для Oracle, где транзакция начинается с выполнением первого оператора, прочие сервера могут работать по-другому. Например в Informix DS, транзакция начинается явно, при помощи команды BEGIN WORK.

В SQL-бочке меда есть своя ложка дегтя. Для всех SQL-серверов использующих журнальный режим регистрации транзакций существует проблема, так называемых "длинных" транзакций. Это транзакции, которые затрагивают очень большой объем данных (сопоставимый с количеством свободного места на дисках) и в этом случае журналы регистрации транзакций могут переполниться. Если их рост ничем неограничен, то они могут израсходовать у ОС всю доступную дисковую память, что не есть хорошо, т.к. операционная система и сервер СУБД в этом случае остаются в непредсказуемом состоянии. Если их рост ограничен, то при переполнении журналов СУБД выдает соответствующую ошибку и операция откатывается. Чтобы избежать таких ситуаций программист должен разделить длинную транзакцию на короткие транзакции.

11. Блокировки

Для того чтобы пользователи не искажали взаимно используемые данные, сервер СУБД, при многопользовательской работе, использует механизм блокировок. Блокировки по аналогии с базами данных на основе файлов могут быть как разделяемые, так и исключительные. Блокировки могут устанавливаться как на таблицу целиком, так и на строку в таблице. Аналогично в xBase-технологиях: блокировки могут устанавливаться как на xBase-файл, так и на запись в xBase-файле.

Блокировки связаны с транзакциями. Если выполняется отмена транзакции, то снимаются все связанные с этой транзакцией блокировки.

Многие блокировки выполняются неявно для пользователя, они выставляются, например, операторами UPDATE, INSERT. Существуют явные

операторы задания блокировок, например, LOCK TABLE или операторы, имеющие клаузы блокировки, например SELECT : FOR UPDATE. Соответственно есть операторы и для снятия блокировок.

Многие SQL-серверы имеют специальные способы обнаружения и предотвращения взаимных блокировок (deadlocks), которые могут занимать ресурсы СУБД на неопределенное время.

Способы, которыми обеспечиваются блокировки, зависят от реализации сервера, и описываются в его документации. Виды блокировок также зависят от используемого сервера. В Informix существуют, т.н. promotional locks, это означает, что если клиентский процесс не может блокировать в исключительном режиме ресурс, то такая блокировка ставится в очередь и ей предоставляется ресурс после снятия текущей исключительной блокировки другого процесса. Oracle7 так не делает, если он не может установить исключительную блокировку в течение указанного сервером времени, то клиент извещается об ошибке.